HOTEL TECHNOLOGY – NEXT GENERATION

Property Web Services

Framework 2.0 Specification
Version 1.0.8
Release 2

FINAL (Synchronous)
DRAFT (Asynchonous)

Table of Contents

PREFACE

Hotel Technology Next Generation ("HTNG") is a nonprofit organization with global scope, formed in 2002 to facilitate the development of next-generation, customer-centric technologies to better meet the needs of the global hotel community. HTNG's mission is to provide leadership that will facilitate the creation of one (or more) industry solution set(s) for the lodging industry that:

- Are modeled around the customer and allow for a rich definition and distribution of hotel products, beyond simply sleeping rooms;
- Comprise best-of-breed software components from existing vendors, and enable vendors to collaboratively produce world-class software products encompassing all major areas of technology spending: hotel operations, telecommunications, in-room entertainment, customer information systems, and electronic distribution;
- Properly exploit and leverage a base system architecture that provides integration and inter operability through messaging; and that provides security, redundancy, and high availability;
- Target the needs of hotel companies up to several hundred properties, that are too small to solve the issues themselves;
- Will reduce technology management cost and complexity while improving reliability and scalability; and
- Can be deployed globally, managed remotely, and outsourced to service providers where needed.

In June 2005, HTNG announced the first-ever "Branding and Certification Program" for hotel technology. This program will enable vendors to certify their products against open HTNG specifications, and to use the "HTNG Certified" logo in their advertising and collateral materials. It will enable hotels to determine which vendors have completed certification of their products against which specific capabilities, and the environments in which performance is certified. HTNG's vision is to achieve a flexible technical environment that will allow multiple vendors' systems to interoperate and that will facilitate vendor alliances and the consolidation of applications, in order to provide hotels with easily managed, continually evolving, cost-effective solutions to meet their complete technology needs on a global basis.

Framework 2.0

A key group of technologists from across the industry were formed into a workgroup to revise the methodology for system interconnection via Web Services.

This group included the following people representing the following companies:-

Name	Company Represented	Email
Kristofer Agren	OpenCourse Solutions	kagren@opencourse.com
Sophie Grigg	PAR Springer-Miller Systems	sophie_grigg@springermiller.com
Alex Lobakov	PAR Springer-Miller Systems	alex_lobakov@springermiller.com
Alex Shore	Newmarket International	AShore@newmarketinc.com
Tom Gresham	MICROS Systems, Inc.	tgresham@micros.com
Andreas Hagedorn	Trust International	ahagedorn@trustinternational.com
Brad More	Theodatus	brad.more@theodatus.com
Mark Pullen	InfoGenesis	mpullen@infogenesis.com
Doug Rice	HTNG	douglas.rice@htng.org

The group met together during 2006 & 2007 to prepare this specification material. During Hitec (June 2006) various vendors were able to demonstrate the use of the new Framework 2.0

methodology to interface systems for various HTNG initiatives. In addition a number of workgroups have either used this Framework as their connectivity mechanism or plan to do so.

This specification describes the Framework 2.0 methodology it remains a working document until a number of vendors are live with Framework 2.0.

Overview

This specification outlines a set of existing open standards, patterns and practices that have gained significant acceptance throughout the IT industry that must be supported by an implementer to claim HTNG Framework 2.0 compliance. The framework prescribes a service oriented architecture implemented using Web Services.

An HTNG Framework 2.0 compliant Web Service MUST adhere to the following:

1. Support both SOAP 1.1 and SOAP 1.2
2. Be expressed fully in WSDL and XML Schema
3. Support WS-Addressing (Web Services Addressing 1.0 - Core W3C Working Draft: 2004-12-08 - http://www.w3.org/TR/2004/WD-ws-addr-core-20041208)
4. Be accessible using synchronous and/or asynchronous HTTP as described below
5. There is an assumption that you are processing XML messages correctly utilizing best practices and known conventions

The following is RECOMMENDED

1. Usage of WS-Security to authenticate messages and secure message content. Only the WS-Security 1.0 specification is supported within the context of this specification.
2. Usage of asynchronous HTTP as described below
3. Usage of SOAP faults as described below
4. Usage of the XML Schema and WSDL construction best practices as described below

WS Addressing

The group reviewed many of the specifications in place and in use across many industries. It was felt that the use of existing standards was the optimal way to create the most effective communication standard. This documentation uses the standards described in the Web Services section of the http://www.w3.org

The group decided that guidelines defined within Web Services Addressing were appropriate for Framework 2.0. The W3C Working Draft of December 2004 (2004-12-08 - http://www.w3.org/TR/2004/WD-ws-addr-core-20041208) was chosen as the WS-Addressing version to use because it provided the a platform that was well supported by many of the common software platforms the members were using to develop their software.

Synchronous and Asynchronous communications are in use actively by many HTNG members using the existing HTNG header. Framework 2.0 provides support for both communication patterns. In addition we have provided guidelines that should make the communication process more robust and standardized.

HTTP Communication Patterns

The synchronous pattern, whilst straightforward, is included here for reference and to aid in communicating a best practice approach. As different web service frameworks handle asynchronous communication differently, the approach described in the 'Async pattern' section is the recommended one and while an implementer may choose to support other async patterns, at the very least this async pattern must be supported in future in order to claim HTNG compliance. The full details of Asynchronous communication and the associated fault handling should be considered in draft form within this specification and are described in Appendix 1 of this document. Synchronous communications are defined as follows:-

The Synchronous Communication Process

Synchronous communication is defined as a request/response over the same channel. The overall principals in this communication are defined as follows:-

- Each message will supply a unique MessageID. Should the same message be transmitted with a different MessageID (many organizations utilize a GUID for this purpose), this will be treated as a separate message by the recipient.
- If reply to element contains the anonymous URI then the response will be provided on the same channel, by definition this becomes a Synchronous transaction. Please see http://www.w3.org/TR/2004/WD-ws-addr-core-20041208 (Section 3 – Message Addressing Properties)

Diagrammatically this can be represented via the following sequence diagram.
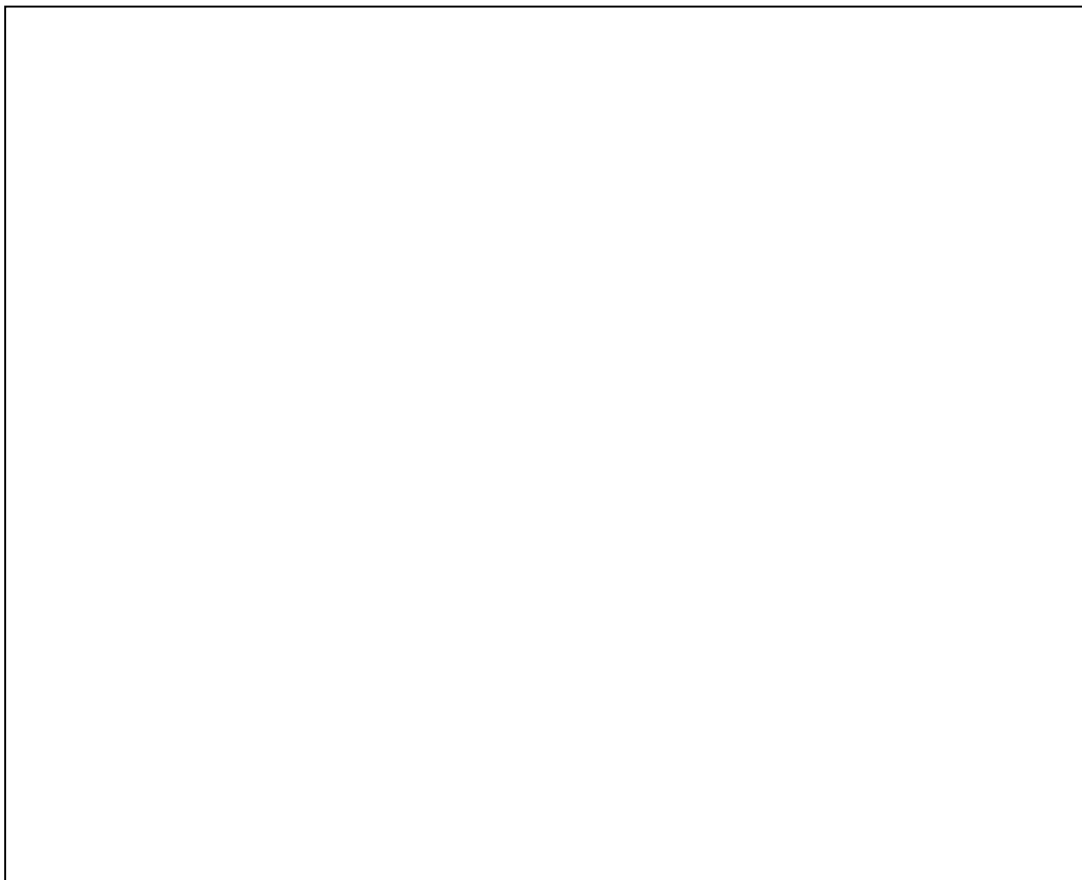
SYNC1

The caller sends the request message, and indicates that the response should be sent on the same connection by supplying the WS-Addressing "anonymous" URI in the wsa:ReplyTo element. Note that if the anonymous URI is used and there is both a wsa:ReplyTo and wsa:FaultTo element, both wsa:ReplyTo and wsa:FaultTo elements MUST use the anonymous URI, i.e. it is not possible to use the anonymous URI for wsa:ReplyTo but not wsa:FaultTo, or vice versa.

Sync - Sample message with HTTP header

```
POST /MyService.asmx HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
Connection: close
```

```
SOAPAction: http://xyz/MyService/SayHello

(01) <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing";>
(02)     <soap:Header>
(04)         <wsa:MessageID>uuid:214A50B2-E62E-4f8b-BD97-
62ABE31E15C2</wsa:MessageID>
(05)         <wsa:ReplyTo>
(06)
<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymou
s</wsa:Address>
(07)         </wsa:ReplyTo>
(08)         <wsa:To>http://xyz/MyService</wsa:To>
(09)         <wsa:Action>http://xyz/MyService/SayHello</wsa:Action>
(10)     </soap:Header>
(11)     <soap:Body>
(12)         <m:SayHello xmlns:m="http://xyz/MyService">
(13)             <m:MyName>John Doe</m:MyName>
(14)         </m:SayHello>
(15)     </soap:Body>
(16) </soap:Envelope>
```

SYNC2

The Web Service sends back the response on the same connection that the request came in on.

Sync - Sample message with HTTP response header

```
HTTP/1.1 200 OK
Date: Wed, 10 May 2006 11:30:07 GMT
Content-Length: nnnn
Content-Type: text/xml; charset="utf-8"

(01) <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(02)     <soap:Header>
(03)         <wsa:MessageID>uuid:5ED743DD-C051-43e8-9287-
D349DEAD38FB</wsa:MessageID>
(04)         <wsa:RelatesTo>uuid:214A50B2-E62E-4f8b-BD97-
62ABE31E15C2</wsa:RelatesTo>
(05)         <wsa:To>http://abc:1234/MyClient </wsa:To>
(06)         <wsa:Action>http://xyz/MyService/SayHelloResponse</wsa:Action>
(07)     </soap:Header>
(08)     <soap:Body>
(09)         <m:SayHelloResponse xmlns:m="http://xyz/MyService">
(10)             <m:Greeting>Hello John Doe</m:Greeting>
(11)         </m:SayHelloResponse>
(12)     </soap:Body>
(13) </soap:Envelope>
```

Sync - Sample fault with HTTP response header

```
HTTP/1.1 200 OK
Date: Wed, 10 May 2006 11:30:07 GMT
Content-Length: nnnn
Content-Type: text/xml; charset="utf-8"
```

```
(01) <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(02)      <soap:Header>
(03)          <wsa:MessageID>uuid:AF6F10EE-B2A8-4080-BEA0-
0A5F03100C60</wsa:MessageID>
(04)          <wsa:RelatesTo>uuid:214A50B2-E62E-4f8b-BD97-
62ABE31E15C2</wsa:RelatesTo>
(05)          <wsa:To>http://abc:1234/MyClient</wsa:To>
(06)
<wsa:Action>http://www.w3.org/2005/08/addressing/fault</wsa:Action>
(07)      </soap:Header>
(08)      <soap:Body>
(09)          <soap:Fault xmlns:m="http://xyz/MyService">
(10)             <faultcode>m:MyNameNotSet </faultcode>
(11)             <faultstring>No name was specified, unable to say
hello</faultstring>
(12)             <detail>
(13)                 <m:MyNameNotSet>
(14)                     <m:SomeElement>Some additional
information</m:SomeElement>
(15)                 </m:MyNameNotSet>
(16)             </detail>
(17)          </soap:Fault>
(18)      </soap:Body>
(19) </soap:Envelope>
```

Overall Fault Handling

SOAP Faults will be provided as a mechanism for handling error conditions. We highly recommend that faults are declared in the WSDL. These may include faults that are business dependent. For example, methods that create reservations may want to return failures for defined failure reasons like "missing arrival or departure date".
Since faults relating to transportation are not (typically) known ahead of time, these would not normally be declared in the WSDL.

A response that contains a fault should be sent to the same address as a reply would, unless a specific wsa:FaultTo element was specified in the header of the request message

Documentation describing an example of SOAP fault handling is attached in Appendix 2.

WSDL Construction and Change Management

We strongly recommended the use of HTTP for transporting messages. This specification primarily focuses on Web Services using HTTP as the transport medium.

Please review
http://www.opengroup.org/htng/propws.pma/protected/upreviews/30/1948/Release_and_Change
_Management_of_HTNG_Specifications_Involving_Web_Services.htm.base regarding the appropriate practices regarding release and change management of HTNG specifications involving Web Services.

Security (if in use)

We recommend the use of WS Security as part of the normal development process. Below is a sample Synchronous message that uses the UsernameToken element as per WS-Security 1.0. WS-Addressing headers are also included. Note that the message is only using WS-Security for authentication, therefore no in-message signature or encryption is performed. The password is provided in clear text, and implies that the communication is secured at the transport level.

Security XML Example

```
(01) <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">
(02)      <soap:Header>
(03)
<wsa:Action>http://htng.org/PWSWG/2006/05/BanquetEventOrder#BeoRequest</wsa
:Action>
(04)           <wsa:MessageID>urn:uuid:29f43cdc-a621-4e2c-80af-
3653545d5502</wsa:MessageID>
(05)           <wsa:ReplyTo>
(06)
<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymou
s</wsa:Address>
(07)           </wsa:ReplyTo>
(08)
<wsa:To>http://127.0.0.1/NIIS/BeoService/BeoService.asmx</wsa:To>
(09)           <wsse:Security soap:mustUnderstand="1">
(10)             <wsu:Timestamp wsu:Id="Timestamp-9f540437-93c5-4b9d-9d57-
afad42eb007b">
(11)                 <wsu:Created>2006-10-30T16:07:46Z</wsu:Created>
(12)                 <wsu:Expires>2006-10-30T16:12:46Z</wsu:Expires>
(13)             </wsu:Timestamp>
(14)             <wsse:UsernameToken
                xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd"
                wsu:Id="SecurityToken-9d1092c8-afdd-421c-9d78-
044f6c25d777">
(15)                 <wsse:Username>TestUserName</wsse:Username>
(16)                 <wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordText">TestPasswordValue</wsse:Password>
(17)                 <wsse:Nonce>+c1BfJJI3Slsm6J4l9Wk2w==</wsse:Nonce>
(18)                 <wsu:Created>2006-10-30T16:07:46Z</wsu:Created>
(19)             </wsse:UsernameToken>
(20)           </wsse:Security>
(21)      </soap:Header>
(22)      <soap:Body>
(23)      </soap:Body>
(24) </soap:Envelope>
```

Routing

Limited discussion has taken place to date on routing. The discussions to date have determined:-

1. If routing (or any type of intermediary forwarding mechanism) is used, a next-hop approach is highly recommended, i.e. every entity in the chain know only of the next hop in the chain.
2. Every end point should be treated in the same way, i.e there is no discernible difference between an intermediary (e.g, a router) and the logical final recipient.

Appendix 1 - The Asynchronous Communication Process

Note: This appendix, Asychronous Communications Process, is considered to be in draft status and is subject to change in the next release.

Asynchronous behavior is accomplished by implementing three "normal" web service methods, one on the service side that gets called by the consumer to start the asynchronous process (the method should have the _SubmitRequest suffix), and two methods that are implemented by the consumer to receive the result (suffixed _SubmitResult) or a fault (_SubmitFault).

The response/reply message in each message exchange (the _SubmitRequest message exchange and _SubmitResult/_SubmitFault message exchange) are treated like normal and are correlated using WS-Addressing.

The asynchronous response or fault is correlated using a custom HTNG SOAP header that the consumer creates. The correlation id is considered an opaque string and must be unique (the use of a UUID is recommended but not required). The address to where the asynchronous reply or fault is sent to is also controlled using custom HTNG SOAP headers.

For an asynchronous operation that completes without a fault, the process is as follows, assuming a fictitious method called "XYZ" part of a fictitious specification that has the namespace http://htng.org/abcspec/

ASYNC1

The consumer generates a unique string that will be used for correlation, sets it in the SOAP header, and creates the _SubmitRequest message. In this sample, a UUID was used for CorrelationID :

```
(01)  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
             xmlns:htng="urn:tobedetermined">
(02)      <soap:Header>
(03)          <wsa:MessageID>uuid:214A50B2-E62E-4f8b-BD97-
62ABE31E15C2</wsa:MessageID>
(04)          <wsa:ReplyTo>
(05)
<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymou
s</wsa:Address>
(06)          </wsa:ReplyTo>
(07)          <wsa:To>http://htng.org/abcspec</wsa:To>
(08)
<wsa:Action>http://htng.org/abcspec/XYZ_SubmitRequest</wsa:Action>
(09)          <htng:CorrelationID>uuid:2D11751F-916F-4c1f-B1FD-
9D6D051AC90A</htng:CorrelationID>
(10)          <htng:ReplyTo>
(11)              <wsa:Address>http://abc:1234/MyClientAsync</wsa:Address>
(12)          </htng:ReplyTo>
(13)          <htng:FaultTo>
(14)              <wsa:Address>http://abc:1234/MyClientAsync</wsa:Address>
(15)          </htng:FaultTo>
(16)      </soap:Header>
(17)      <soap:Body>
(18)          <m:XYZ_SubmitRequest xmlns:m="http http://htng.org/abcspec">
(19)              ... The request ...
(20)          </m:XYZ_SubmitRequest>
(21)      </soap:Body>
(22)  </soap:Envelope>
```

ASYNC2

The provider receives the message, initiates the asynchronous process and sends back an "empty" SOAP message

```
(01) <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" >
(02)     <soap:Header>
(03)         <wsa:MessageID>uuid:9971BF72-F42D-49ee-99DB-
BEC28B6EDAF7</wsa:MessageID>
(04)         <wsa:RelatesTo>uuid:214A50B2-E62E-4f8b-BD97-
62ABE31E15C2</wsa:RelatesTo>
(05)
<wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</ws
a:To>
(06)
<wsa:Action>http://htng.org/abcspec/XYZ_SubmitRequestResponse</wsa:Action>
(07)     </soap:Header>
(08)     <soap:Body/>
(09) </soap:Envelope>
```

ASYNC3

The provider completes the asynchronous process, and invokes the _SubmitResult method on the consumer (the address was specified by the htng:ReplyTo element in the original _SubmitRequest method).

```
(01) <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
            xmlns:htng="urn:tobedetermined" >
(02)     <soap:Header>
(03)         <wsa:MessageID>uuid:C15EE2B2-B41C-44c4-901E-
1032159CCC6A</wsa:MessageID>
(04)         <wsa:ReplyTo>
(05)
<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymou
s</wsa:Address>
(06)         </wsa:ReplyTo>
(07)         <wsa:To>http://htng.org/abcspec</wsa:To>
(08)         <wsa:Action>
http://htng.org/abcspec/XYZ_SubmitResult</wsa:Action>
(09)         <htng:RelatesToCorrelationID>uuid:2D11751F-916F-4c1f-B1FD-
9D6D051AC90A</htng:RelatesToCorrelationID>
(10)     </soap:Header>
(11)     <soap:Body>
(12)         <m:XYZ_SubmitResult xmlns:m="http://htng.org/abcspec">
(13)             ... The result ...
(14)         </m:XYZ_SubmitResult>
(15)     </soap:Body>
(16) </soap:Envelope>
```

ASYNC4

The consumer sends back an "empty" SOAP response message

```
(01) <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(02)     <soap:Header>
(03)         <wsa:MessageID>uuid:6F400F52-8912-4dab-BEEB-
FEDEC356979F</wsa:MessageID>
(04)         <wsa:RelatesTo>uuid:C15EE2B2-B41C-44c4-901E-
1032159CCC6A</wsa:RelatesTo>
```

11

```
(05)
<wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</ws
a:To>
(06)
<wsa:Action>http://xyz/MyService/XYZ_SubmitResultResponse</wsa:Action>
(07)      </soap:Header>
(08)      <soap:Body/>
(09) </soap:Envelope>
```

Diagrammatically this can be represented via the following sequence diagram:
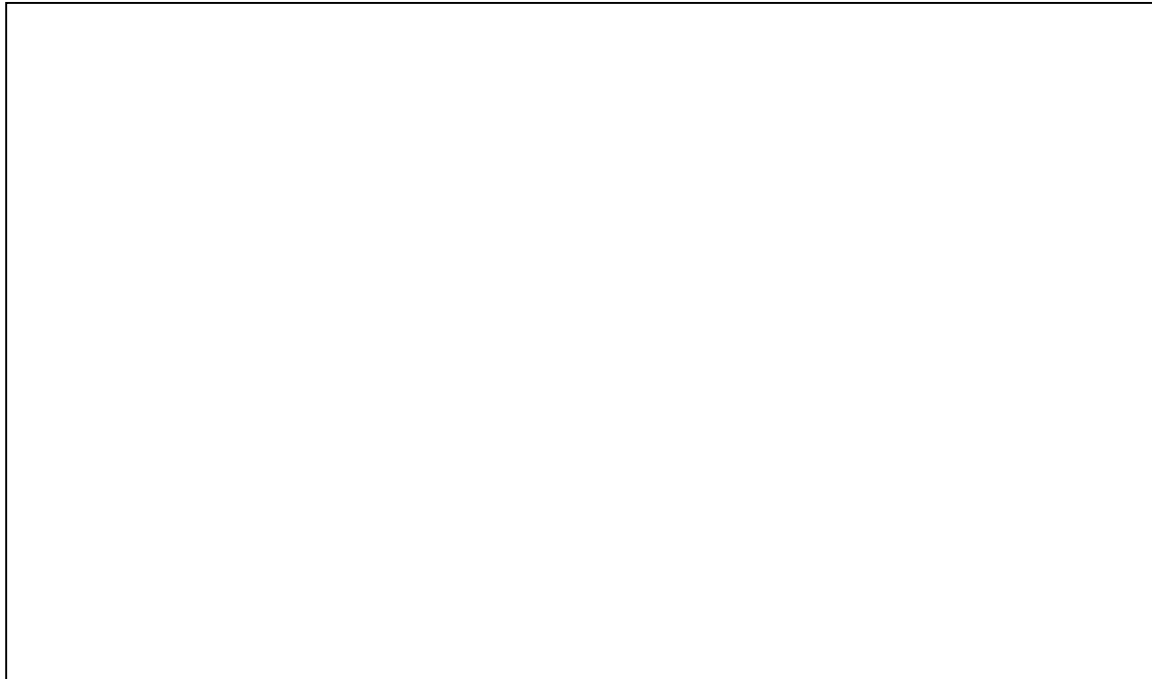
Asynchronous Fault handling

There are four places where a fault can be reported:

1. In the response to _SubmitRequest method, i.e. the provider throws a fault back to the consumer.
   This fault will be sent back instead of the "empty" message.
   If this happens, the consumer can assume that NO asynchronous process was started and there will thus be no _SubmitResult or _SubmitFault call.
2. In the _SubmitFault method, i.e. the provider signals the consumer that the asynchronous operation completed as the result of a fault.
3. In the response to the _SubmitResult method, i.e. the consumer returns a fault back to the provider as a response to the _SubmitResult method.
4. In the response to the _SubmitFault method, i.e. the consumer returns a fault back to the provider as a response to the _SubmitFault method.

Asynchronous WSDL Structure

Create two port types, one that will hold the methods implemented by the providers and one implemented by the consumer to receive completion and error callbacks. For each _SubmitRequest method in the provider, create two callback methods, _SubmitResult and _SubmitFault, on the consumer's side.

Asynchronous WSDL Example

```
(01) <!--"Normal" port type that the provider implements.
        Contains both synchronous methods and the initiating method of
asynchronous methods-->
(02) <portType name="ReservationProviderPortType">
(03)     <!--This is the normal synchronous version-->
(04)     <operation name="CreateReservation">
(05)         <input/>
(06)         <output/>
(07)         <fault/>
(08)     </operation>
(09)     <!--This is the asynchronous version-->
(10)     <operation name="CreateReservation_SubmitRequest">
(11)         <input
message="tns:CreateReservation_SubmitRequestInputMessage"/>
(12)         <output message="tns:EmptyMessage"/>
(13)     </operation>
(14) </portType>
(15) <!--"Callback" port type that the caller implements to be able to
receive completion
        callbacks (successes and failures) on asynchronous methods-->
(16) <portType name="ReservationAsyncCompletionPortType">
(17)     <operation name="CreateReservation_SubmitResult">
(18)         <input
message="tns:CreateReservation_SubmitResultInputMessage"/>
(19)         <output message="tns:EmptyMessage"/>
(20)     </operation>
(21)     <operation name="CreateReservation_SubmitFault">
(22)         <input
message="tns:CreateReservation_SubmitFaultInputMessage"/>
(23)         <output message="tns:EmptyMessage"/>
(24)     </operation>
(25) </portType>
```

Simple HTNG Framework 2.0 Asynchronous Sample

A sample application is contained in the file *HTNG_Framework_2.0_Simple_Async_Sample.zip* in the *HTNG_Framework_2.0_Samples.zip* archive which can be downloaded from the same location as this specification.

The sample illustrates an implementation of the HTNG Framework 2.0 asynchronous pattern in WSE 3.0. In this sample, the notion of a "provider" and a "consumer" is used. The provider represents the "service" that is providing some business functionality, in this particular sample the business functionality consists of a dummy method called "CreateReservation". The consumer represents the client application that is calling the provider to create a reservation.

The asynchronous pattern in the HTNG Framework 2.0 specifies an asynchronous operation as two separate message exchanges:

1. The consumer makes a regular synchronous http Web Service call (where the reply is received on the same connection as the request was sent on) to initiate the asynchronous request. An empty reply is sent back to indicate successful receipt of the request. The provider will kick off the asynchronous process.
2. The provider will make a regular synchronous http Web Service call (where the reply is received on the same connection as the request was sent on) to the consumer when the asynchronous process is completed to deliver the successful result or the unsuccessful fault of the asynchronous process.

The web methods that the provider and the consumer implements are described in separate portTypes in the WSDL to clearly separate the difference in the two roles that the consumer and provider plays in the message exchange.

This sample would have been fairly straightforward if WSE3 had out-of-the-box provided a way to accept http requests to a non-ASP.NET hosted process (by simply adding an "http" URI to the WSE "SoapReceivers" in the same way that it is possible to add a "soap.tcp" SoapReceiver, for example). While this is supported in WCF/Indigo, WSE3 does not support it, so this sample also contains a small Class Library "HttpSysTransport" originally written by Aaron Skonnard (with some minor enhancements) to make this possible in WSE3.

Appendix 2 - SOAP Fault handling

It is recommended to use the standard SOAP fault model as described in the SOAP 1.1 and 1.2. This specification divides faults into two categories:

1. Faults that are known ahead of time, which are typically business-specific faults and are specific to each Web Service. These faults are well defined in the WSDL and XML Schema of the Web Service.
2. Faults that are not known ahead of time, which are typically implementation specific faults, e.g. communication faults, etc.

Declarative Approach

Faults that are known ahead of time and are business-specific should be declared in the WSDL to let the consumer of the Web Service know what types of business-related faults can be expected by calling a specific method. Each fault must also have a corresponding XML Schema element declared that uniquely describes the fault. Consider the following WSDL definition of a Web Service Method where there are two faults declared in the WSDL :

```
(01) <operation name="CreateReservation">
(02)     <input message="tns:CreateReservationInputMessage"/>
(03)     <output message="tns:CreateReservationOutputMessage"/>
(04)     <fault name="NoAvailability"
message="tns:NoAvailabilityFaultMessage">
(05)         <documentation>Thrown if there is no longer any
availability;</documentation>
(06)     </fault>
(07)     <fault name="InvalidData" message="tns:InvalidDataFaultMessage">
(08)         <documentation>Thrown if one or more fields in the data are
not filled in or did not validate.
                            The contents of the fault will contain more
information</documentation>
(09)     </fault>
(10) </operation>
```

The message definition for the NoAvailabiltyFaultMessage and InvalidDataFaultMessage look like this:

```
(01) <message name="NoAvailabilityFaultMessage">
(02)     <part name="parameters" element="tns:NoAvailabilityFault"/>
(03) </message>
(04) <message name="InvalidDataFaultMessage">
(05)     <part name="parameters" element="tns:InvalidDataFault"/>
(06) </message>
```

And the XML Schema elements NoAvailabilityFault and InvalidDataFault look like this:

```
(01) <xs:element name="NoAvailabilityFault">
(02)     <xs:complexType>
(03)         <xs:sequence>
(04)             <xs:element name="FirstAvailableDateAndTime"
type="xs:dateTime"/>
```

14

```
(05)            </xs:sequence>
(06)        </xs:complexType>
(07) </xs:element>
(08) <xs:element name="InvalidDataFault">
(09)        <xs:complexType>
(10)            <xs:sequence>
(11)                <xs:element name="Field" maxOccurs="unbounded">
(12)                    <xs:complexType>
(13)                        <xs:sequence>
(14)                            <xs:element name="Reason" type="xs:string"/>
(15)                        </xs:sequence>
(16)                        <xs:attribute name="name" type="xs:string"
use="required"/>
(17)                    </xs:complexType>
(18)                </xs:element>
(19)            </xs:sequence>
(20)        </xs:complexType>
(21) </xs:element>
```

A SOAP 1.2 envelope containing the SOAP fault for InvalidData would look like this:

```
(01) <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
                    xmlns:t="http://new.webservice.namespace"
                    xmlns:xml="http://www.w3.org/XML/1998/namespace">
        … WS-Addressing, Security, etc headers omitted …
(02)    <env:Body>
(03)        <env:Fault>
(04)            <env:Code>
(05)                <env:Value>env:Sender</env:Value>
(06)                <env:Subcode>
(07)                    <env:Value>t:InvalidData</env:Value>
(08)                </env:Subcode>
(09)            </env:Code>
(10)            <env:Reason>
(11)                <env:Text xml:lang="en">Some of the fields were
invalid</env:Text>
(12)            </env:Reason>
(13)            <env:Detail>
(14)                <t:InvalidDataFault>
(15)                    <t:Field name="FirstName">
(16)                        <t:Reason>A first name must set</t:Reason>
(17)                    </t:Field>
(18)                    <t:Field name="DOB">
(19)                        <t:Reason>The date of birth cannot be in the
future</t:Reason>
(20)                    </t:Field>
(21)                </t:InvalidDataFault>
(22)            </env:Detail>
(23)        </env:Fault>
(24)    </env:Body>
(25) </env:Envelope>
```

And the SOAP 1.2 envelope for the NoAvailability fault would look like this:

```
(01) <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
                    xmlns:t="http://new.webservice.namespace"
                    xmlns:xml="http://www.w3.org/XML/1998/namespace">
        … WS-Addressing, Security, etc headers omitted …
(02)    <env:Body>
(03)        <env:Fault>
```

```
(04)                 <env:Code>
(05)                     <env:Value>env:Sender</env:Value>
(06)                     <env:Subcode>
(07)                         <env:Value>t:NoAvailability</env:Value>
(08)                     </env:Subcode>
(09)                 </env:Code>
(10)                 <env:Reason>
(11)                     <env:Text xml:lang="en">There was no availability left
to complete the reservation.</env:Text>
(12)                 </env:Reason>
(13)                 <env:Detail>
(14)                     <t:NoAvailabilityFault>
(15)                         <t:FirstAvailableDateAndTime>2006-12-
17T09:30:00Z</t:FirstAvailableDateAndTime>
(16)                     </t:NoAvailabilityFault>
(17)                 </env:Detail>
(18)             </env:Fault>
(19)         </env:Body>
(20) </env:Envelope>
```

Please note that the example above represents the SOAP message carrying a fault as it might look in a synchronous session, or in an asynchronous session if the fault were to occur in the context of the call.
Fault(s) that occur in the asynchronous process initiated by an asynchronous call would be submitted to the consumer via the _SubmitFault operation, therefore the SOAP message might look close to the following:

```
(01) <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
                    xmlns:t="http://new.webservice.namespace"
                    xmlns:xml="http://www.w3.org/XML/1998/namespace">
        … WS-Addressing, Security, RelatesToCorrelationID etc. headers
omitted …
(02)    <env:Body>
(03)        <m:XYZ_SubmitFault>
(04)          <env:Fault>
(05)            <env:Code>
(06)              <env:Value>env:Sender</env:Value>
(07)                <env:Subcode>
(08)                  <env:Value>t:NoAvailability</env:Value>
(09)                </env:Subcode>
(10)            </env:Code>
(11)          <env:Reason>
(12)            <env:Text xml:lang="en">There was no availability left to
complete the reservation.</env:Text>
(13)          </env:Reason>
(14)          <env:Detail>
(15)            <t:NoAvailabilityFault>
(16)              <t:FirstAvailableDateAndTime>2006-12-
17T09:30:00Z</t:FirstAvailableDateAndTime>
(17)            </t:NoAvailabilityFault>
(18)          </env:Detail>
(19)          </env:Fault>
(20)        </m:XYZ_SubmitFault>
(21)    </env:Body>
(22) </env:Envelope>
```

Appendix 3 - SOAP Exception Handling

A sample application has been created that provides an example of SOAP exception handling.

This example is contained in the file
HTNG_Framework_2.0_SOAP_Fault_Handling_with_WSE_Sample.zip in the
HTNG_Framework_2.0_Samples.zip archive which can be downloaded from the same location as this specification.

The package contains examples and utilities for how to more easily use structured SOAP exceptions in the Microsoft WSE environment. Microsoft WSE does not fully implement structured SOAP exceptions (those exceptions that are identified by the contents of the <Detail> element in the SOAP fault element).

The method described in the example is aimed to be similar to what is seen in other web service toolkits , e.g. AXIS, and Microsoft's upcoming communication framework "WCF" (previously known as "Indigo"). Once WCF is released, porting structured SOAP exception handling using this method will be straightforward. In order to make this work, this method consists of two parts:

1. A utility to generate the exception classes that correspond to a WSDL <fault> element that refers to a message with an XML Schema element describing the content. This utility is provided in this package as a console application with full source called "GenerateFaultWrappersFromWsdl." The exception classes that are created all inherit from SoapException, so it is easy and straightforward to use them on the service side, simply "throw" the strongly typed exception and you are done. A little bit more work is required on the client side to translate the SoapException to a strongly typed exception, for this purpose the GenerateFaultWrappersFromWsdl utility also generates a class to map a SoapException to a strongly typed exception.

2. In order to make the translation of the exception on the client side to happen seamlessly to the caller of the web service proxy, the proxy class that gets generated by the Wsdl/WseWsdl3 utilities needs to be extended. With .NET2 this can be done in a non-invasive way since the proxy web service class is generated with the "partial" attribute. Please see the MathServiceExtension.cs source file in "Contract.Client".

NOTE: One alternative to this could also be to implement a custom Policy Assertion in WSE3 and throw the strongly typed exception from there. Unfortunately, WSE3 will wrap all non-SoapException exceptions thrown from a policy assertion, which would not let the caller use the plain try {} catch {} pattern.

This will allow service implementers use the following style of code :

```
throw new MyStronglyTypedException(…);
```
And client implementers use the following style of code :
```
try
{
    Mywebserviceproxy.SomeCall(…);
}
catch(MyStronglyTypedException myException)
{
    // The content of the exception is in the myException.TypedDetail
property
}
```

Appendix 4 - Math(A+B) sample project

A sample application is provided that implements a simple A+B application. This application should be used as a minimum primer to become familiar with Framework 2.0.

This example is contained in the file *HTNG_Framework_2.0_Math_A+B_Sample.zip* in the *HTNG_Framework_2.0_Samples.zip* archive which can be downloaded from the same location as this specification.

This example contains

- Custom UserTokenManager that demonstrates and discusses how to implement proprietary authentication.
- Custom Policy implemented in code to require SOAP action header and UsernameToken
- Web service implementation of add, subtract, multiply, divide.
- Console application client that demonstrates how to call the service with a username token, etc.

Appendix 5 - Payment Posting Example

A sample application is provided that implements the payment posting message. This message has been most recently used by the Single Guest Itinerary workgroup.

The sample is contained in the file *HTNG_Framework_2.0_Payment_Posting_Sample.zip* in the *HTNG_Framework_2.0_Samples.zip* archive which can be downloaded from the same location as this specification.

The example implements a Web Service at http://htng.org/PWSWG/2006/04/SingleGuestItinerary#PostPayment and was developed using a small part of Activity.wsdl designed by Guest Itinerary Workgroup. The WSDL that was used can be found in the Schema subfolder. Please note, no custom Fault conditions were defined in the WSDL.

The following pre-requisites are required on the system in order to build and run the sample:

- MS IIS
- MS .NET Framework v2.0
- MS Visual Studio 2005
- MS Web Service Enhancements v3.0

Appendix 6 - Additional Examples

Additional examples have been provided by the Workgroup to assist in people's understanding of Framework 2.0.

1. HTNG 2.0 OTA Ping Service

   This example is contained in the file *HTNG_Framework_2.0_OTA_Ping_Sample.zip* in the *HTNG_Framework_2.0_Samples.zip* archive which can be downloaded from the same location as this specification.

   The zip archive contains:

   • a WSDL which describes the service and should allow the generation of the client code
   • an example for the request and response including the HTTP header To use the service, use:

   username = trust
   password = xyz

   The service is programmed in Java and uses AXIS 1.4.

   For addressing, http://schemas.xmlsoap.org/ws/2008/08 was used.
2. WS-Security example

   This example is contained in the file *HTNG_Framework_2.0_WS_Security_Sample.xml* in the *HTNG_Framework_2.0_Samples.zip* archive which can be downloaded from the same location as this specification.

This is a sample Synchronous message that uses the UsernameToken element as per WS-Security 1.0. WS-Addressing headers are also included. Please note that the message is only using WS-Securityfor authentication, therefore no in-message signature or encryption is performed. The password is provided in clear text, and implies that the communication is secured at the transport level.