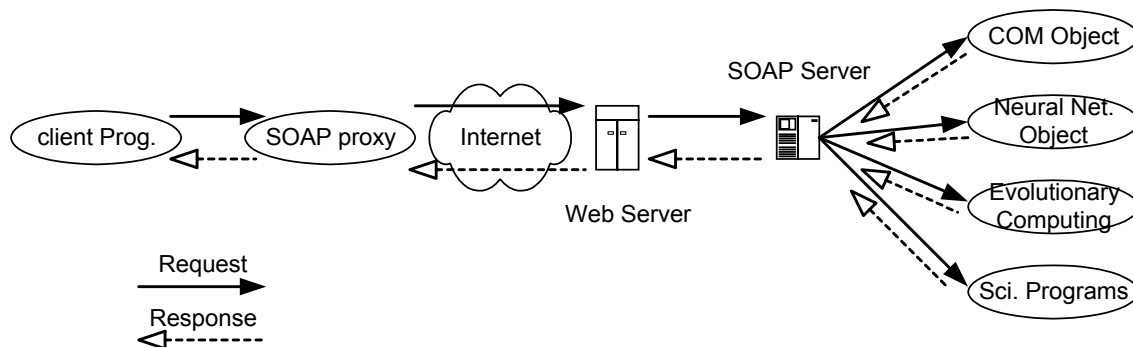


Lecture 1: Introduction to SOAP

SOAP: Simple Object Access Protocol

Let's put some network between the object and us, then create a thing called a "SOAP client". A SOAP client is called a client, not because it is used by some end-user human, but just because it creates a thing called a "SOAP request document". In fact, most SOAP clients are applications or servers. This SOAP request document is sent over the network, usually over HTTP, to a piece of code called a "SOAP server".



An example of a SOAP request document is shown in following:
SOAP Request Document.

```
POST /getWeather HTTP/1.1
Host: www.weather.com
Content-Length: 244
Content-Type: text/xml
SOAPAction: http://myweather.com/services#getWeather

<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getWeather xmlns="urn:schemas-architag-com:weather">
      <zipcode>80112</zipcode>
      <temp>F</temp>
      <wind>MPH</wind>
    </getWeather>
  </soap:Body>
</soap:Envelope>
```

Why use HTTP? There are several reasons. (1) The most important is that the HTTP infrastructure already exists. There are Web servers in every organization already, and HTTP has already proven to work well across networks. (2) The second reason is that it uses the standard HTTP port of any TCP connection, port 80. This port is wide open in most organizations, because organizations want their employees to be able to access resources on the Web.

This document is sent from the SOAP client to the SOAP server, where the SOAP server must parse it out and find the appropriate elements. Then, the SOAP server must tell the object, in its language, what to do. When the object speaks back, the SOAP server must translate that and return the results to the SOAP client.

However it gets the information from the object, it is the SOAP server's job, then, to encapsulate this information into a thing called a SOAP response document and send it back to the SOAP client. The following is a SOAP Response Document:

```
HTTP/ 1.1 200 OK
Content-Type: text/xml
Content-Length: 367

<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <forecast
      xmlns="http://weather.com/schemas/getweather.xsd">
      <day name="2002-04-05">
        <high>78</high>
        <low>44</low>
        <wind>5</wind>
      </day>
      <day name="2002-04-06">
        <high>78</high>
        <low>44</low>
        <wind>5</wind>
      </day>
    </forecast>
  </soap:Body>
</soap:Envelope>
```

Demo: mySOAP_0010.xml 及 mySOAP_0011.xml:
(為了方便作 validation, 將官方的 SOAP Schema 檔更名爲 SOAPSchemas.xsd)

Notice that the SOAP response document is also an HTTP document.

SOAP provides a simple methodology for getting some procedure call package over HTTP to a destination. However, we think that SOAP has two distinct personalities. Both of these personalities use the exact same specification and the same methodology for getting a payload from one location to another. However, you should understand these two personalities of SOAP, because how you handle the SOAP documents may vary widely. The two personalities are:

- SOAP for **RPC**
- SOAP for **Messaging**

Using SOAP for RPC, the payload inside of the SOAP document is usually short, and the response is immediate. This is called “synchronous”. The suggestion is that “I need a piece of information from you now, and I will wait until you give it to me.”

Using SOAP for messaging usually requires more handshaking, because the results might not come back right away. In this way, SOAP for messaging is usually “asynchronous.” The suggestion here is, “Here is a purchase order. Let me know if you received it, and tell me in the next couple days if you can deliver the goods.”

SOAP RPC Convention

When RPC calls are serialized in a SOAP message, the name of the element must match the name of the method, as do the parameters.

Consider the method signature:

```
//Return the current stock price, given the company symbol  
double GetStockQuote ( [in] string sSymbol );
```

If our method namespace is <http://www.wroxstock.com/>, then the serialized method call that requests the stock quote using symbol OU812 would look like this:

```
<q:GetStockQuote xmlns:q= http://www.wroxotck.com/ >
  <q:sSymbol xsi:type="xsl:string">OU812</q:sSymbol>
</q:GetStockQuote>
```

Method 名稱

該 Method 所需的變數 名稱

該變數的數值

The method name matches the element, as does the parameter. While the parameter names match the child elements, only inbound parameters appear in the serialized call.

The Return

As we mentioned earlier, the RPC convention uses a request-response model. Just as the call is represented in the request SOAP message, the results of the call are returned in the response SOAP message.

The name of the method response struct can be anything, but it is a convention to append the word "Response" to the name of the method call struct. If the method returns a value, the name is irrelevant, but it must be the first child of the method struct.

(不管其 tag name，將第一個 child node 的值視為回傳值 return value)

Return values cannot be identified by name, only position, but the name should not conflict with the parameters of the method.

```
// Reverse the string, s, and return the new string.
string ReverseString ( [in] string s );
```

```
// Reverse the string, s, and return the new string.
void Reverse ( [in] sting s, [out] string sRev );
```

```
// Reverse the string, s, passed in by reference.
void ReverseString ( [in, out] string s );
```

The first version:

```
<ReverseStringResponse xmlns:x= http://www.wrox.com/ >
  <x:ret xsi:type="xsd:string">THOR</x:ret>
</x:ReverseString>
```

(1) method 名稱

(2) 可自訂 element name，因為原 subrouitne 並不需要 parameter 回傳，但因為它會 return，∴必須要求第一個 child element 的值就是其 return value，但是

element name 名字並不重要。

The second version:

```
<ReverseStringResponse xmlns:x= http://www.wrox.com/ >  
  <x:sRet xsi:type= "xsd:string">THOR</x:sRet>  
</x:ReverseString>
```

(1) method 名稱

(2) 必須取成 **sRet**，因為它是函數中傳參數(首成 pointer, or reference)

The third version:

```
<ReverseStringResponse xmlns:x= http://www.wrox.com/ >  
  <x:s xsi:type= "xsd:string">THOR</x:s>  
</x:ReverseString>
```

(1) method 名稱

(2) 必須取成 **s**，因為它是函數中傳參數(首成 pointer, or reference)

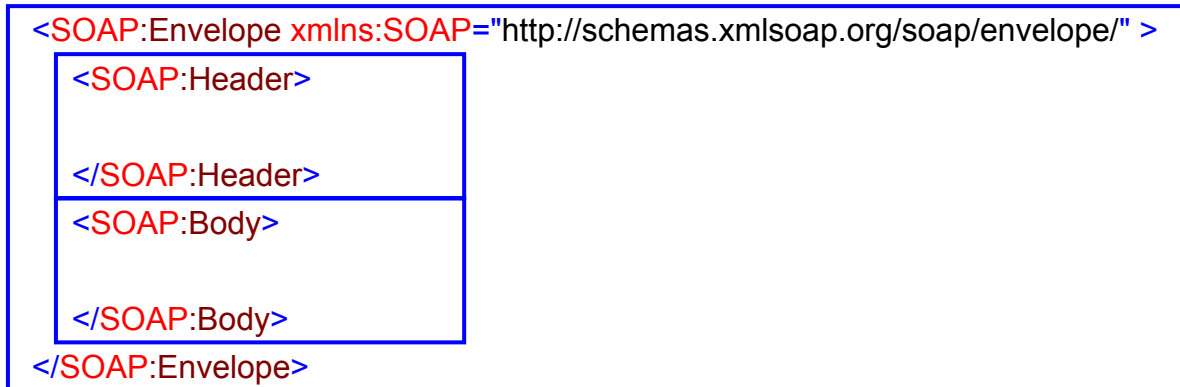
利用一些 tools 將已開發完成的 functions 轉成 request SOAP message/
response SOAP message。

Demo: xmlServer01/ xmlClient01 中的:

WebServerProject02.exe/ myBrowserProject01.exe

Lecture 2: Structure of a SOAP Message

一、Structure of a SOAP Message



二、SOAP 使用範例。

For example, you could use the Header to process an order.

1. Original client request:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/" >  
  <SOAP:Header>  
    <t:fillInID xmlns:t="http://www.scottseely.com/customer"  
      SOAP:mustUnderstand="1"  
      SOAP:actor="http://schemas.xmlsoap.org/soap/actor/next">  
      <t:bodyID>cust</t:bodyID>  
    </t:fillInID>  
  
    <u:placeOrder xmlns:u="http://www.scottseely.com/po"  
      SOAP:mustUnderstand="1"  
      SOAP:actor="http://www.scottseely.com/placeOrder">  
      <u:bodyID>lineItems</u:bodyID>  
    </u:placeOrder>  
  
    <v:shipOrder xmlns:v="http://www.scottseely.com/po"  
      SOAP:mustUnderstand="1"
```

```
        SOAP:actor="http://www.scottseely.com/shipOrder">
        <v:bodyID>shipper</v:bodyID>
    </v:shipOrder>

</SOAP:Header>

<SOAP:Body>
    <customer ID="cust">
        <name ID="custName" first="Joe" Last="Smith" />
        <address id="custAddress">
            <street>123 Main st.</street>
            <city>Milwaukee</city>
            <state>WI</state>
            <zip>53219</zip>
        </address>
    </customer>

    <order ID="lineItems">
        <item>bat</item>
        <item>glove</item>
        <item>baseball</item>
    </order>

    <ship ID="shipper">
        <shipBy>UPS Ground</shipBy>
        <shipTo>
            <name href="#custName" />
            <address href="#custAddress" />
        </shipTo>
    </ship>

</SOAP:Body>

</SOAP:Envelope>
```

2. Message gets passed to the order system
(<http://www.scottseely.com/placeOrder>)

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/" >
```

```
<SOAP:Header>
```

```
<u:placeOrder xmlns:u="http://www.scottseely.com/po"
  SOAP:mustUnderstand="1"
  SOAP:actor=" http://schemas.xmlsoap.org/soap/actor/next ">
  <u:bodyID>linelItems</u:bodyID>
</u:placeOrder>
```

```
<v:shipOrder xmlns:v="http://www.scottseely.com/po"
  SOAP:mustUnderstand="1"
  SOAP:actor="http://www.scottseely.com/shipOrder">
  <v:bodyID>shipper</v:bodyID>
</v:shipOrder>
```

```
</SOAP:Header>
```

```
<SOAP:Body>
```

```
<customerID ID="10233" />
```

```
<customer ID="cust">
```

```
<name ID="custName" first="Joe" Last="Smith" />
```

```
<address id="custAddress">
```

```
<street>123 Main st.</street>
```

```
<city>Milwaukee</city>
```

```
<state>WI</state>
```

```
<zip>53219</zip>
```

```
</address>
```

```
</customer>
```

```
<order ID="linelItems">
```

```
<item>bat</item>
```

```
<item>glove</item>
```

```
<item>baseball</item>
```

```
</order>
```



```
<ship ID="shipper">  
  <shipBy>UPS Ground</shipBy>  
  <shipTo>  
    <name href="#custName" />  
    <address href="#custAddress" />  
  </shipTo>  
</ship>  
  
</SOAP:Body>  
  
</SOAP:Envelope>
```

The **mustUnderstand** Attribute

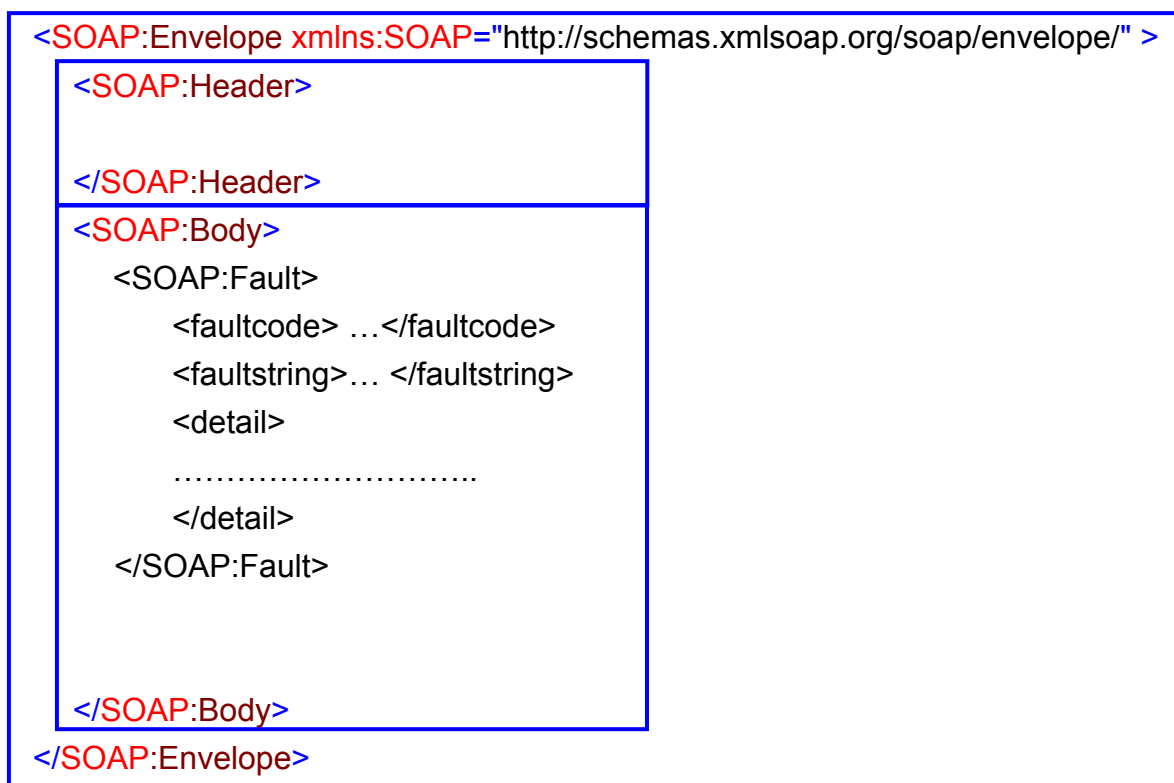
The `mustUnderstand` attribute specifies whether it is absolutely necessary for the SOAP server to process the message in the header. A value of "1" indicates that the header entry is mandatory, whereas a value of "0" indicates that the header entry is optional. The `<t:fillInID ...>` header entry, we specified a value of "1" for `mustUnderstand`. This means that the **SOAP server must process the header entry**. If the SOAP server doesn't understand this header entry, it must reject the entire SOAP message- it is not allowed to process the entries in the SOAP body.

The **actor** Attribute

In some cases a SOAP message may pass through a number of applications on a number of computers before it gets to its final destination. You might send a SOAP message to computer A, which might then send that message on to computer B. Computer A would be called a **SOAP intermediary**. In these cases you can specify that some SOAP headers must be performed by a specific intermediary, by using the **actor** attribute. **The value of the attribute is a URI, which uniquely identifies each intermediary**. Or, if you want the header entry to be executed **by the next intermediary in the line**, regardless of what computer it's on, you can use the special URI defined in the SOAP specification, <http://schemas.xmlsoap.org/soap/actor/next> .

Lecture 3: Structure of a SOAP Fault Message

一、Structure of a SOAP Fault Message



二、SOAP 使用範例。

For example, you could use the Header to process an order.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >  
  <soap:Body>  
    <soap:Fault>  
      <faultcode>soap:server</faultcode>  
      <faultstring>Insufficient funds</faultstring>  
      <detail>  
        <x:TransferError xmlns:x="http://www.Acer.com.tw">  
          <sourceAccount> 22-3415</sourceAccount>  
          <transferAmount>100.0</transferAmount>  
        </x:TransferError>  
      </detail>  
    </soap:Fault>  
  </soap:Body>  
</soap:Envelope>
```

```
</x:TransferError>  
</detail>  
</soap:Fault>  
</soap:Body>  
</soap:Envelope>
```

Lecture 4: SOAP Encoding- Simple Data Types

The SOAP Specification defines a single set of encoding rules that are referred to as **SOAP encoding**. SOAP encoding is based on XML Schemas and as such it closely models many of the standard types and constructs that developers would be familiar with. The value of the encodingStyle attribute for SOAP encoding is <http://schemas.xmlsoap.org/soap/encoding/>, which points to the XML Schema that defines the encoding rules.

The SOAP Encoding consists of two parts: **Simple Data Types** and **Compound Data Types**:

In SOAP encoding, simple types are always represented as single elements in the body. SOAP encoding exposes all the simple types that are built into the XML Schemas Specification. If we are using a simple type with SOAP encoding, then it must come from XML Schemas, or be derived from a type that does.

一、Primitive Simple Data Type

if we assume the **xsd** prefix is associated with the XML Schemas URI, and the **soapENC** prefix is associated with the SOAP encoding URI, then both of these payload values work with strings.

`<param1 xsi:type="xsd:string">OU21</param1>` refers to XML Schemas:

`<param2 xsi:type="soapENC:string">OU22</param2>` refers to SOAP encoding:

demo: mySOAP_0050.xml

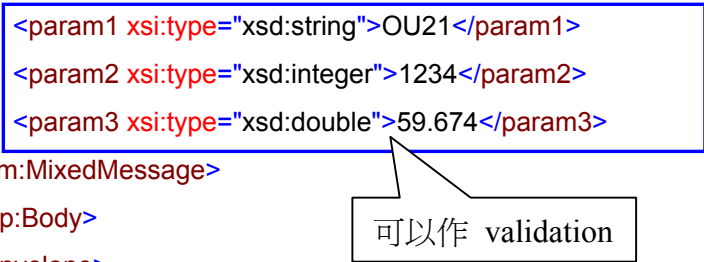
xsi:type attribute 的註冊：

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <soap:Body>
    <m:MixedMessage xml:m="http://www.ntou.edu.tw/" >
      <param1 >OU21</param1>
      <param2 >1234</param2>
      <param3>59.674</param3>
    </m:MixedMessage>
```

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

That message meets all the requirements of SOAP, but many implementations would not be able to process it because they would not be able to map the values in the payload to types in the target language. (即無法對 payload 中的資料作 validation)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"  
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" >  
  <soap:Body>  
    <m:MixedMessage xml:m="http://www.ntou.edu.tw/" >  
      <param1 xsi:type="xsd:string">OU21</param1>  
      <param2 xsi:type="xsd:integer">1234</param2>  
      <param3 xsi:type="xsd:double">59.674</param3>  
    </m:MixedMessage>  
  </soap:Body>  
</soap:Envelope>
```



Now all the data in the payload is identified by type, and it becomes much easier for a SOAP implementation to process.

(如何作 validation ?? 對 XMLSPY 而言，上述兩種方式都是一樣 valid !!)

二、Enumerations。

Here is an example of an enumeration that defines a set of geographical regions.

```
<simpleType name="Region" base="xsd:string">  
  <enumeration value="North" />  
  <enumeration value="East" />  
  <enumeration value="South" />  
  <enumeration value="West" />  
</simpleType>
```

if this enumeration appeared in a referenced schema, we could then use this type in a SOAP message.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">  
  <soap:Body>  
    <m:GetSalesTotals xml:m="http://www.ntou.edu.tw/" >  
      <m:reg xsi:type="m:Region" >East</m:reg>  
    </m:GetSalesTotals>  
  </soap:Body>  
</soap:Envelope>
```

三、 Binary Data

As part of the simple types it supports, SOAP and XML Schemas provide a type for representing binary data. One approach for working with binary data is to use the *base64* type. We can represent binary data, such as an image file, as an array of bytes in the message. The *base64* type converts binary data to text using the *base64*-encoding algorithm of XML Schemas.

Lecture 5: SOAP Encoding- Compound Data Types

SOAP encoding handles two compound types: *structs* (records), and *arrays*.

一、Structs Type

Consider C++ struct definition of a super-hero:

Struct SuperHero

```
{  
    string sCodeName;  
    string sFirstName;  
    string sLastName;  
    int nAge;  
};
```

```
SuperHero hero = {"Hulk", "Bruce", "Banner", 32 }
```

If we serialize the variable “hero” into a SOAP message payload using SOAP encoding, it would look like this:

```
<hero xsi:type="x:SuperHero">  
    <sCodeName xsi:type="xsd:string">Hulk</sCodeName>  
    <sFirstName xsi:type="xsd:string">Bruce</sFirstName>  
    <sLastName xsi:type="xsd:string">Banner</sLastName>  
    <nAge xsi:type="xsd:integer">32</nAge>  
</hero>
```

As can be seen in this example, the *xsi:type* attribute is used on compound data types as well as simple types. In this case, the type is *x:SuperHero*, and the *x* namespace would point to a schema that represents out *SuperHero* struct.

二、Arrays Type

In SOAP encoding, arrays are considered a special type. This type is indicated by their *xsi:type* attribute, which is *SOAP-ENC:Array*. As with all SOAP encoding, the namespace associated with the Array type is <http://schema.xmlsoap.org/soap/encoding>. Elements with this *xsi:type* are declared as SOAP encoding arrays.

The type of the array members is declared using another attribute, `SOAP-ENC:arrayType`. This attribute indicates the type and size of the array.

```
<numbers xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:integer[5]">
  <item>10</item>
  <item>20</item>
  <item>30</item>
  <item>40</item>
  <item>50</item>
</numbers>
```

or

```
<names xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[4]">
  <e>John</e>
  <e>Tom</e>
  <e>Allen</e>
  <e>Harry</e>
</names>
```

使用 `SOAP-ENC:ur-type[]` 作為 universal type :

```
<mix xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="SOAP-ENC:ur-type[4]">
  <e xsi:type="xsd:string">John</e>
  <e xsi:type="xsd:integer">7</e>
  <e xsi:type="xsd:string">Allen</e>
  <e xsi:type="xsd:data">1999</e>
</mix>
```

使用 `SOAP-ENC:offset` 作為 陣列資料的起點 :

```
<names xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[4]"
      SOAP-ENC:offset="[2]" >
  <e>Allen</e>
  <e>Harry</e>
</names>
```

只傳 A[2] 及 A[3]，並沒有傳 A[0] 及 A[1] 過來！

使用 SOAP-ENC:position 傳送部分陣列資料 (sparse Matrix) :

```
<names xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[100]" >  
  <e SOAP-ENC:position="[11]">Allen</e>  
  <e SOAP-ENC:position="[45]">Harry</e>  
</names>
```

只傳 A[11] 及 A[45]，其他資料並沒有傳過來！

Q: 如何對一份 SOAP 資料作 data validation ?