

## Email Open Interfaces – Developing compliant web services

1. Introduction
2. Format
  - 2.1 WSDL Format
  - 2.2 Method Format
  - 2.3 Flattening .NET Web Services WSDLs
3. Administration
4. Timeouts and capacity
5. Log messages
6. Sample application
7. Debugging with SoapUI

### 1. Introduction

You can develop custom web services that the Email Manager can call when an email is processed. Custom web services can perform tasks such as manipulating the originating email and modifying the rule routing options. The Email Open Interfaces package in the Email Manager dynamically creates a Web service client based on the WSDL of your web service.

AACC Email Open Interfaces have been designed to work with any web services that follow the specifications outlined in this document. Although this document provides some specific information for developing compliant web services in .NET, this feature is not restricted to using .NET applications and has also been tested against Java EE and Cache web services.

The web services must follow a set format to make this possible and this format, as well as other considerations, is outlined below.

### 2.1 WSDL Format

The web service must produce a single flat file WSDL; the use of imports is not supported. Using imports essentially causes the WSDL to be split into a number of separate files, the main WSDL and a number of nested ones, instead of a single file.

The .NET Framework 4.5 provides a single WSDL option of the box, however if you are using an older version of .NET then it will be necessary to carry out some additional steps to flatten the WSDL. Section 2.3 details these additional steps.

You must use basic HTTP binding. The following example from a .NET Web.Config shows the correct HTTP binding configuration:

```
<endpoint address="basic" binding="basicHttpBinding" bindingConfiguration=""  
contract="EmailManagerService.IEmailFilterService">
```

```
<identity>
  <dns value="localhost" />
</identity>
</endpoint>
```

## 2.2 Method Formatting

The methods on the web service must follow a set format. Non-compliant methods may exist on the web services but they will not be considered for use by Email Open Interfaces.

- Methods must accept as an input parameter an array of strings (String[])
- Methods must return an array of strings as the return value (String[])

Example of a **valid** method signature:

```
public string[] GetTrackingNo(string[] inputParameters)
```

Example of an **invalid** method signature:

```
public int GetTrackingNo(string Customer) – This will not work.
```

Methods must always use string arrays, even if the method accepts a single string or returns a single string from the web service.

## 2.3 Flattening .NET Web Services WSDLs

By default .NET projects generate a WSDL with nested internal WSDLs. When using .NET 4.5, you have the option of choosing a single WSDL or using one with imports. You must choose the ?singleWsdL option. The use of a WSDL with import statements is not supported in Email Open Interfaces.

### EmailOpenInterfacesService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://localhost:49932/EmailOpenInterfacesService.svc?wsdl
```

You can also access the service description as a single file:

```
http://localhost:49932/EmailOpenInterfacesService.svc?singleWsdL
```

#### 2.3.1 Single WSDL option

The single WSDL link will return a single WSDL file, containing all the schemas in it:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy" xmlns:wsa10="
xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsx="http://sche
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" xmlns:wsdp="http://schemas.
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.0"
targetNamespace="http://tempuri.org/">
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="http://tempuri.org/">...</xs:schema>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://schemas.microsoft.com/2003/10/Serialization/" attributeFormDefault="q
targetNamespace="http://schemas.microsoft.com/2003/10/Serialization/">...</xs:schema>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://schemas.microsoft.com/2003/10/Serialization/Arrays" elementFormDefaul
targetNamespace="http://schemas.microsoft.com/2003/10/Serialization/Arrays">...</xs:schema>
  </wsdl:types>
  <wsdl:message name="IEmailOpenInterfacesService_PrioritiseBasedOnFromAddress_InputMessage">...</wsdl:message>
  <wsdl:message name="IEmailOpenInterfacesService_PrioritiseBasedOnFromAddress_OutputMessage">...</wsdl:message>
  <wsdl:message name="IEmailOpenInterfacesService_DeleteCreditCardNumber_InputMessage">...</wsdl:message>
  <wsdl:message name="IEmailOpenInterfacesService_DeleteCreditCardNumber_OutputMessage">...</wsdl:message>
  <wsdl:message name="IEmailOpenInterfacesService_GetTrackingId_InputMessage">...</wsdl:message>
  <wsdl:message name="IEmailOpenInterfacesService_GetTrackingId_OutputMessage">...</wsdl:message>
  <wsdl:message name="IEmailOpenInterfacesService_Translate_InputMessage">...</wsdl:message>
  <wsdl:message name="IEmailOpenInterfacesService_Translate_OutputMessage">...</wsdl:message>
  <wsdl:portType name="IEmailOpenInterfacesService">...</wsdl:portType>
  <wsdl:binding name="BasicHttpBinding_IEmailOpenInterfacesService" type="tns:IEmailOpenInterfacesService">...</wsdl:binding>
  <wsdl:service name="EmailOpenInterfacesService">...</wsdl:service>
</wsdl:definitions>
```

### 2.3.2 Generated single file WSDL

If you are using older versions of .NET then this option is not available. There is however a number of methods to 'flatten' a WCF WSDL into a single file, including using a custom WCF service host (<http://blogs.msdn.com/b/dotnetinterop/archive/2008/09/23/flatten-your-wsdl-with-this-custom-servicehost-for-wcf.aspx>) or using WCFExtras (<http://wcfextras.codeplex.com/>).

WCFExtras is a collection of useful WCF extensions and you can easily add this to your project and use it to create a single WSDL.

Follow the following steps to do so:

1. Get the recommended download from the WCFExtras web site.
2. Add a reference to WCFExtras.dll to your project. This DLL will also need to be deployed with the web service.
3. Update the configuration file to reference the DLL:

Within the `<system.serviceModel>` `</system.serviceModel>` tags add a reference to the DLL:

```
<extensions>
  <behaviorExtensions>
    <add name="wsdlExtensions" type="WCFExtras.Wsdl.WsdlExtensionsConfig,
      WCFExtras, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
  </behaviorExtensions>
</extensions>
```

4. Update the configuration file to set the singlefile flag to true:  
This is done by adding a new endpoint behavior within the `<behaviors>` `</behaviors>` tag:

```
<endpointBehaviors>
  <behavior name="WSDL.SingleFileWsdlEndpointBehavior" address="basic">
```

```

        <wsdlExtensions singleFile="True"/>
    </behavior>
</endpointBehaviors>

```

- Update the configuration file to add a reference to this behavior to the endpoint:

```

<endpoint address="basic" binding="basicHttpBinding" bindingConfiguration=""
behaviorConfiguration="WSDL.SingleFileWsdEndpointBehavior"
contract="EmailManagerService.IEmailFilterService">
    <identity>
        <dns value="localhost" />
    </identity>
</endpoint>

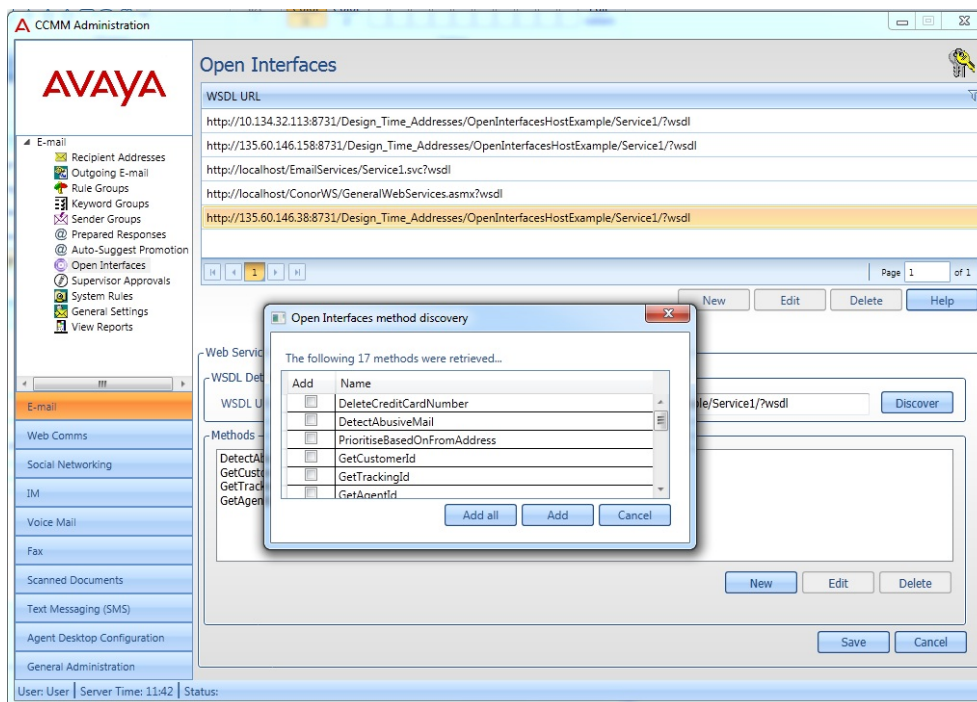
```

Note: the address for the basic HTTP binding and the behavior should match, both are set to basic in this example.

### 3. Administration

The administration of this feature is done in the CCMM Administration application. For more information about the Email Open Interfaces Administration see *Avaya Aura® Contact Center Server Administration* (44400-610).

You can use the Discover button on this page to find the valid methods on the web service:



#### 3.1 Method discovery

#### 4. Timeouts and capacity

The Email Manager includes a client side timeout mechanism. When the Email Manager calls a web service it will wait a maximum of one minute for a response. If it does not receive a response in that time then it will abort the call. If AACC is configured to call an open interface web service for a high percentage of the emails being processed and if these calls are taking a relatively long time to return then this will have the effect of slowing down the processing of emails on the CCMM server. This reduces the capacity of Email Manager to process emails from the server. This should be taken into account for all customer sites but particularly for ones running at a medium to high traffic level.

#### 5. Log messages

The following are the list of relevant log messages that will be logged to the Email Manager log file.

Log level	Event ID	Log message
DEBUG	14186	Create open interfaces result set
INFO	14187	No parameters were returned from the web service
INFO	14188	Size mismatch in the number of parameters returned from the web service, expected %1 but received %2
INFO	14189	Null list returned from the web service
DEBUG	14190	Parse WSDL
INFO	14191	Could not get WSDL definition
DEBUG	14192	Get WSDL definition for %1
INFO	14193	Exception getting WSDL definition: %1
INFO	14194	Exception parsing WSDL envelope header: %1
DEBUG	14195	Method parsed: %1
INFO	14196	Method could not be parsed: %1
INFO	14197	All methods were parsed
INFO	14198	Not all methods were parsed
DEBUG	14199	Method '%1' could not be matching to any binding operation
DEBUG	14200	Exception parsing method '%1': %2
DEBUG	14201	Open Interfaces Engine: %1
DEBUG	14202	Load data into the open interface engine
DEBUG	14203	Number of web services retrieved from database: %1
DEBUG	14204	Load of open interface engine complete, result: %1
DEBUG	14205	Reload open interface engine
DEBUG	14206	Call method: %1 (ID: %2)
DEBUG	14207	This web service method has already been parsed successfully
DEBUG	14208	This web service has not been parsed, attempt to do so now
DEBUG	14209	Result of attempted parsing: %1
INFO	14210	Web service could not be parsed so it cannot be called at this time
INFO	14211	No web service found for this ID: %1
DEBUG	14212	Unknown input parameter type, %1

DEBUG	14213	Web service invoker: create service
DEBUG	14214	Web service invoker: create dispatch
DEBUG	14215	Web service invoker: create request
DEBUG	14216	Web service invoker: add input parameters to request
DEBUG	14217	Web service invoker: process reply
INFO	14218	Web service invoker: exception processing reply
DEBUG	14219	Web service invoker: call method %1
DEBUG	14220	Call open interface engine with method ID: %1
DEBUG	14221	Result Set: %1
INFO	14222	Method not found, cannot call open interfaces
DEBUG	14223	Map open interfaces result set to mail
DEBUG	14224	Map output parameter %1 to mail
INFO	14225	Unknown output parameter type: %1
DEBUG	14226	Null value returned from web service, make no change
INFO	14227	Open Interfaces result set is empty
INFO	14228	Exception reading open interfaces last modified time: %1
DEBUG	14229	Reload of open interfaces web services needed
INFO	14285	Exception reading the open interfaces timeout properties form the mailservice properties file
DEBUG	14286	Open interfaces timeout value: %1

## 6. Sample application

The sample application has been developed with Visual Studio 2012 and uses .NET Framework 4.5. It is a WCF project which may be hosted in IIS. It contains 4 sample methods. These contain shell implementations and are intended to be used for illustrative purposes only.

Project Name: EmailOpenInterfacesSampleWebService

URL of the WSDL: <http://localhost:49932/EmailOpenInterfacesService.svc?singleWsdI>

Sample methods:

### 1. PrioritiseBasedOnFromAddress

If an email has a from address containing '@avaya.com' then set the priority level to 1 and append text to the subject to indicate that the email is of a high priority. If the from address does not contain '@avaya.com' then set the priority to 5 and append text to the subject to indicate that the email is of a lower priority.

*Input parameters:*

- From Address
- Subject

*Output parameters:*

- Subject
- Priority

## **2. DeleteCreditCardNumber**

Search the body of an email for a credit card number and if found then scrub them and return the modified email.

Credit card formats that will be detected are:

- 1234-1234-1234-1234
- 1234123412341234
- 1234 1234 1234 1234

*Input parameters:*

- Plain text body of email
- HTML body of email

*Output parameters:*

- Plain text body of email
- HTML body of email

## **3. GetTrackingId**

Searches the subject for a tracking ID in the format AVAYA1234 and if found returns a custom field. This custom field can then be used to launch a screen pop.

*Input parameters:*

- Subject

*Output parameters:*

- Custom field name
- Custom field value

## **4. Translate**

This method performs a translation of the supplied email by making calls to Microsoft Translator. You need to sign up to the Microsoft Translator API to receive a valid client ID and client secret code. These then need to be added to the Translator class for this method to work.

*Input parameters:*

- Plain text body of email
- HTML body of email
- Subject

*Output parameters:*

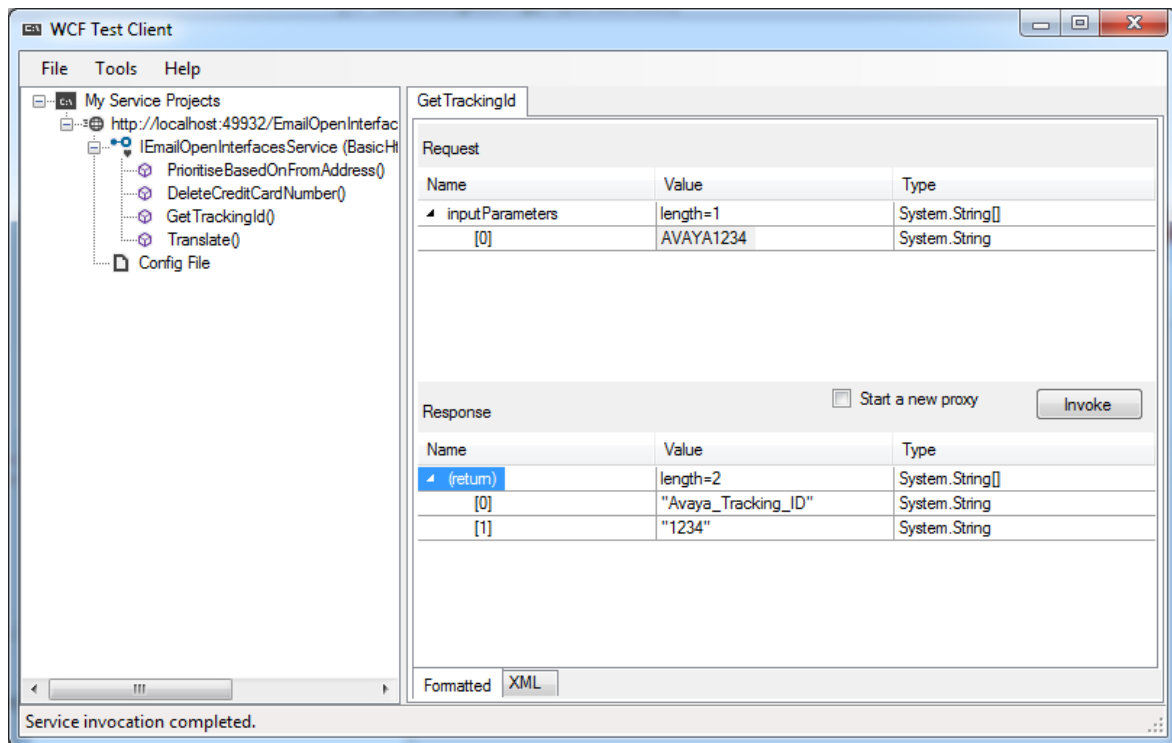
- Plain text body of email
- HTML body of email
- Subject

## 7. Debugging with SoapUI

SoapUI is a free and open source solution for testing web services. You can use it to help test and debug the web services you develop for Email Open Interfaces. There is other similar software available which may also be used.

If the web service you have developed in not working in the way you expected it to then the following steps can be taken to test it.

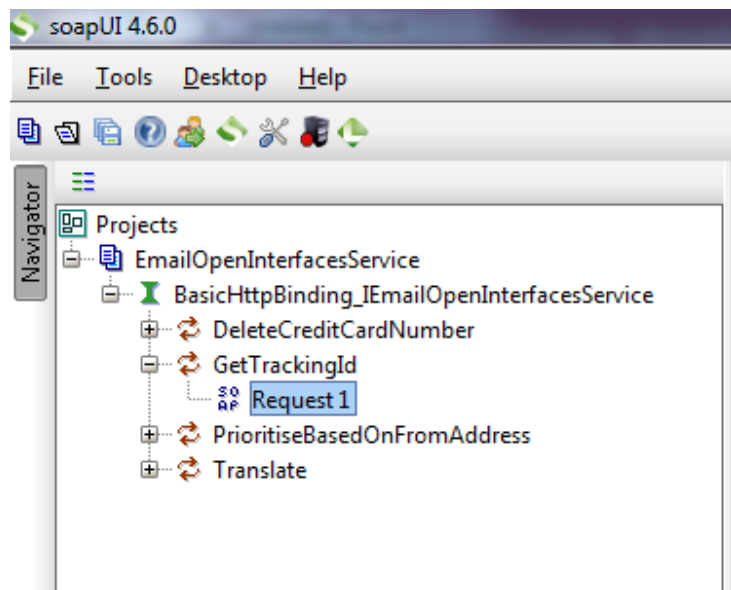
1. Test the web service using the WCF Test Client. This only applies if you are using a .NET application.



### 7.1 Test the web service using the WCF Test Client



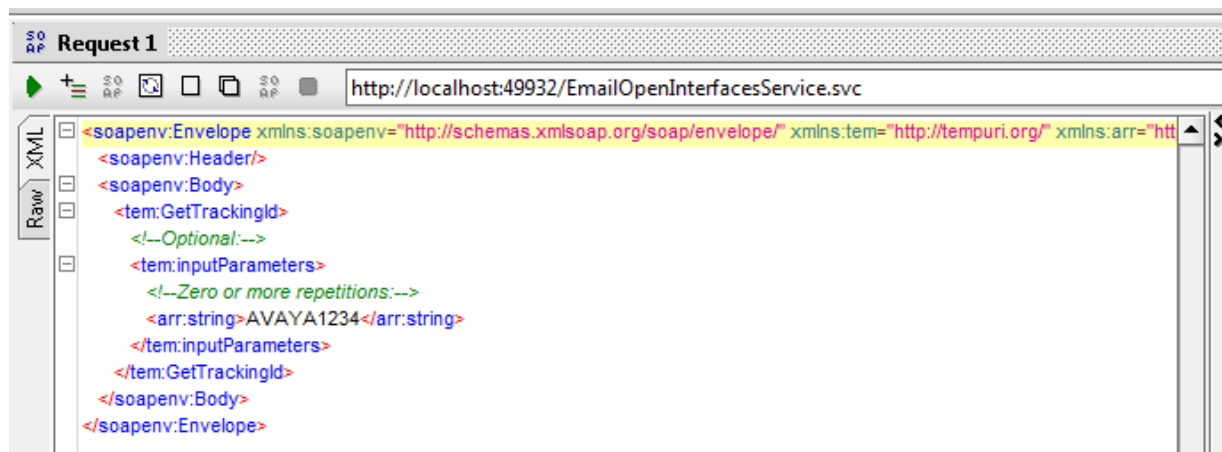
2. Add the web service to a new project in SoapUI.



7.2 Adding the web service to SoapUI

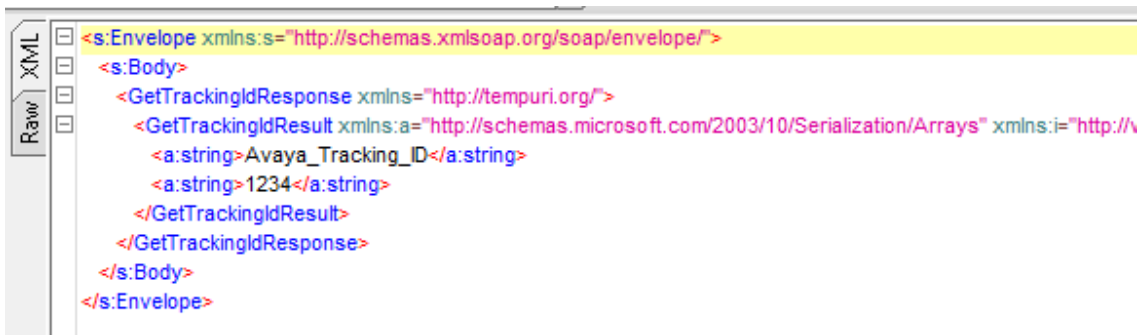
3. Test the web service using SoapUI.

The application will auto-generate the SOAP envelope that the web service is expecting. You can then add appropriate values before calling the web service. Note the input array of strings.



7.3 Using SOAP envelope generated by SoapUI

The web service should respond with a similar SOAP envelope.



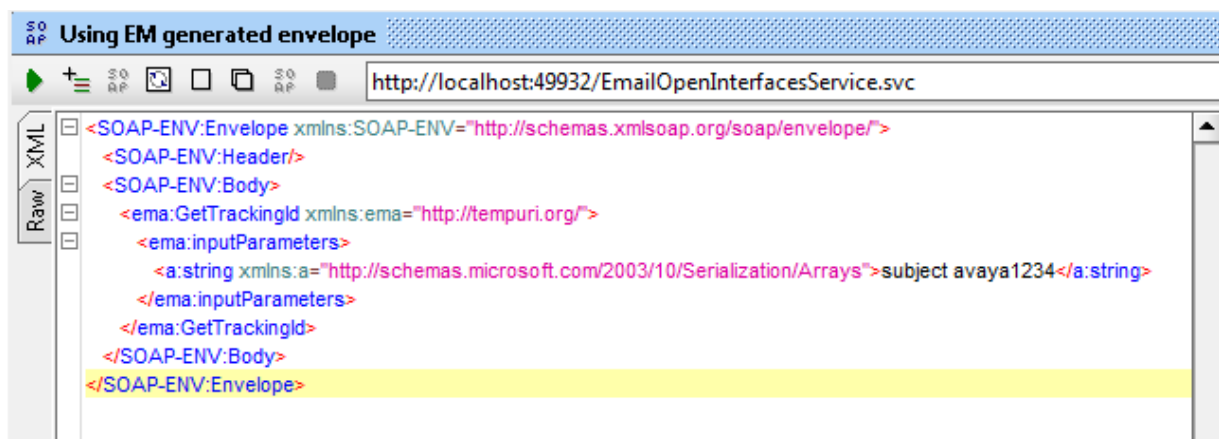
```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <GetTrackingIdResponse xmlns="http://tempuri.org/">
      <GetTrackingIdResult xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <a:string>Avaya_Tracking_ID</a:string>
        <a:string>1234</a:string>
      </GetTrackingIdResult>
    </GetTrackingIdResponse>
  </s:Body>
</s:Envelope>
```

7.4 Response from the web service using SOAP envelope generated by SoapUI

At the point if the web service is responding as expected using SoapUI but the call from Email Manager is not working as expected then take the SOAP envelope that the Email Manager has generated, paste it into a new request in SoapUI and see if you get a valid response. Email Manager logs the SOAP envelope it generates to its log file, CCMM\_EmailManager.log.

Email Manager log message (event ID 24055):

```
2013-10-18 10:57:57.485 +0100 EmailManager:MH 5284:1 24055 Debug None
      SOAP Envelope: <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/><SOAP-ENV:Body><ema:GetTrackingId xmlns:ema="http://tempuri.org/"><ema:inputParameters><a:string xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays">subject avaya1234</a:string></ema:inputParameters></ema:GetTrackingId></SOAP-ENV:Body></SOAP-ENV:Envelope>
```



```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ema:GetTrackingId xmlns:ema="http://tempuri.org/">
      <ema:inputParameters>
        <a:string xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays">subject avaya1234</a:string>
      </ema:inputParameters>
    </ema:GetTrackingId>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

7.5 Using SOAP envelope generated by the Email Manager

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <GetTrackingIdResponse xmlns="http://tempuri.org/">
      <GetTrackingIdResult xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays" xmlns:i="http://www...">
        <a:string>Avaya_Tracking_ID</a:string>
        <a:string>1234</a:string>
      </GetTrackingIdResult>
    </GetTrackingIdResponse>
  </s:Body>
</s:Envelope>
```

7.6 Response from web service using SOAP envelope generated by the Email Manager