Provincial XML Service User Guide

Organization:	Ontario Association of Community Care Access Centres (OACCAC)
Division:	IS/IT
Version:	1.14
Version Date:	10 June 2013
Prepared By:	Fiona Williamson

Revision Log

Version No.	Version Date	Summary of Change	Changed by/Input from
1.0	May 31, 2009	Initial version	Scott Atkins
1.1	June 1, 2009	Updates from feedback, move document to user guide focused.	Scott Atkins
1.2	June 25, 2009	Change security model to delegated Transport-with-Message-Credential Minor updates to WSDL regarding GetMessageList filter object types. GetMessageListFilter argument clarifications and changes Update example config file to show support for new security model.	Scott Atkins
1.3	July 2, 2009	Modified message requests to include user authentication credentials at the message level. Moved security model to Transport-Only to assist vendors in accessing the validation environment in a timely manner.	Scott Atkins
1.4	August 10, 2009	Remove now-obsolete credentials fields from SOAP message Move security model back to message-based security with username token	Scott Atkins

Version No.	Version Date	Summary of Change	Changed by/Input from
		secured via TLS and server-certificate	
1.5		Change security-context attribute for endpoint bindings. Added 'MessageDestination' Filter field for both request and response filters.	Scott Atkins
1.6	October 7, 2009	Added details for the new 'PostMessage' method	Manuel Ng
1.7	November 4, 2009	Updated the example of using 'PostMessage'	Manuel Ng
1.8	November 23, 2009	Updated the GetMessage Data Contracts Added an GetMessage example to mark 'Posted' messages 'Processed'	Manuel Ng
1.9	November 27, 2009	Added the valid message type strings in the GetMessageListFilter data contract for R1.2	Manuel Ng
1.10	December 17, 2009	Added message type strings in the GetMessageListFilter data contract for R1.3	Manuel Ng
1.11	March 29, 2010	Added message type strings in GetMessageListFilter data contract for R1.4 Added an GetMessage example for the multi-versioning feature	Manuel Ng
1.12	August 30,	Updated document content and format	Arthur Bydon

Technical Documentation	Version: 1.14
Provincial XML Service, User Guide	Date: 10 June 2013

Version No.	Version Date	Summary of Change	Changed by/Input from
	2010	to give clear examples and make document easier to use for providers and vendors.	Fiona Williamson
1.13	November 01, 2011	Added acceptable usage section (1.3).	Ion Moraru Fiona Williamson
1.14	March 21, 2012	Added Production and Certification service reference and endpoint addresses to document	Fiona Williamson
1.15	June 10, 2013	Removed reference to extranet. Removed endpoint references.	Fiona Williamson

Table of Contents

Provincial XML Service

User's Guide

1. Introduction

1.1 Purpose

This document presents a technical overview of the service contracts, data contracts, and usage patterns that client is expected to abide by while consuming the Provincial XML Service. Descriptions of messaging contracts, data contracts, and usage examples are provided.

1.2 Background

The Provincial XML Solution will extend the document exchange mechanism currently used between providers and vendors with a new format – XML. The solution will promote a Public Interface designed for **system-to-system** electronic data exchange with Providers and Vendors interested in integrating their systems with CHRIS/HPG platform.

Provincial XML will act as an integration layer on top of the Health Partner Gateway (HPG) to allow service-oriented access to HPG documents in a programmatic way. Additionally, Provincial XML will translate all HPG formats into a new standard XML format, called the Provincial XML Document (PXD) format. Each specific message type in the PXD domain will be backed by a specific XML Schema which will define the structure and rules of the allowable document content. Over time, these PXD schema files will be versioned to support new functionality or re-arrange information to the benefit of all parties involved in document exchange.

Provincial XML was built using Microsoft .Net technology, version 3.5. The underlying service structure was built with Windows Communication Foundation (WCF) .framework.

1.3 Usage Restrictions

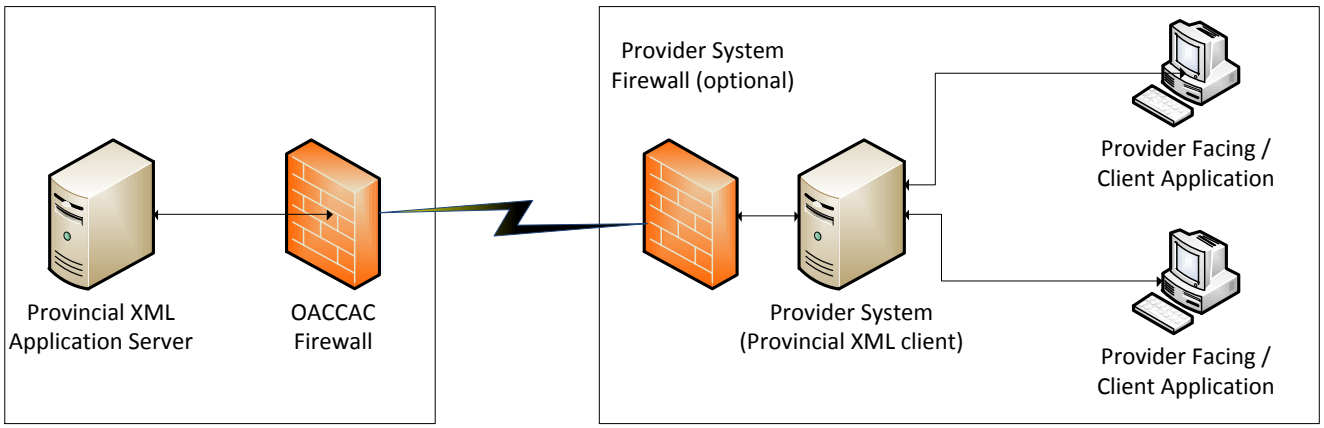
The Provincial XML solution is provided as a system to system interface in order to facilitate the data exchange between the HPG environment to a third party system which will store and present the data using its client application technology of choice.

In this context, the data exchanged through the Provincial XML solution has a transient nature is not guaranteed to be a persistent provider data store. Therefore the Provincial XML solution cannot be used as a definitive repository of provider data for third party systems and connected directly to user facing client applications. Various third party systems have to accumulate the provider data furnished through Provincial XML solution and present it to their client application from within the Provider system boundaries.

The Provincial XML solution will not support connecting user facing client applications directly to the Provincial XML solution as this will unnecessarily increase the bandwidth requirements for the provincial solution. Any third party system has to optimize the Provincial XML data exchange and only access the OACCAC Provincial XML application though a limited (finite) number of connections.

All messages that are retrieved successfully on the first attempt will have to be marked as PROCESSED by the Provincial XML Client application.

An **Acceptable usage** of Provincial XML data exchange is depicted below:

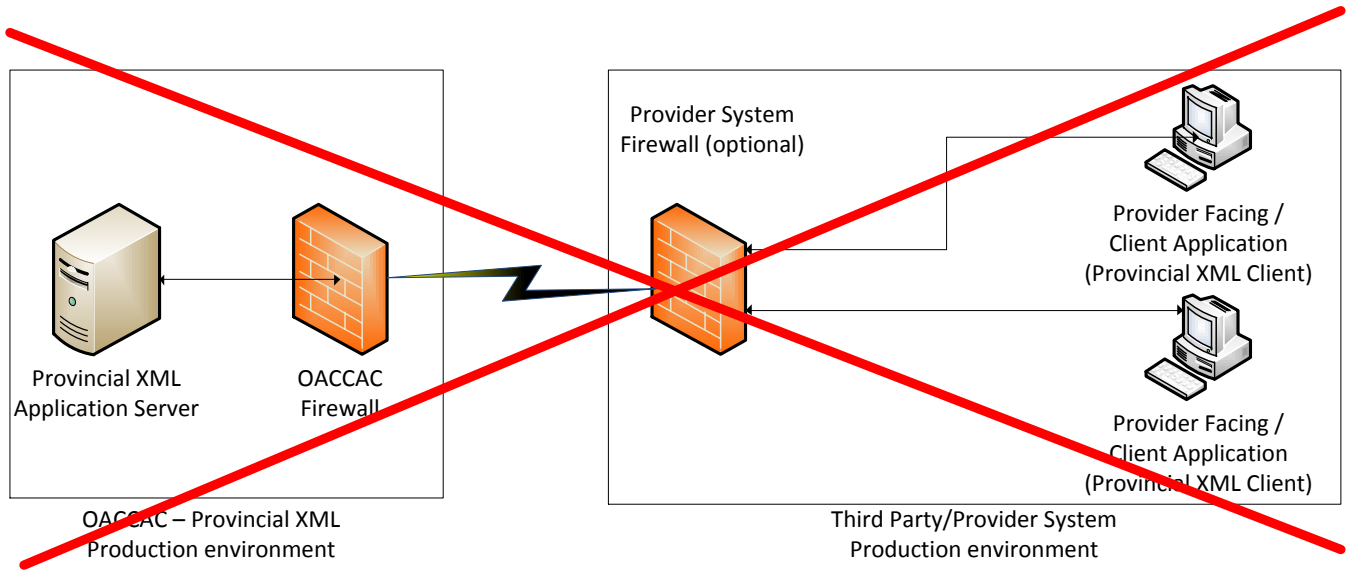


In this scenario the Provider XML client is implemented by a server application within the third party/provider system environment. The server application in this environment will channel all end-user/provider application requests to the Provincial XML server hosted in the OACCAC environment.

In this context, it is also assumed that the transactions that have been consumed already by the third party/provider system are already residing in the third party/provider system environment and will not be requested again from the Provincial XML server application hosted on the OACCAC environment.

This restriction is included to ensure that provider systems are not continuously relying on Provincial XML server application to act as a data store for various third party/provider client applications. The Provincial XML solution implements a transient system from a data persistency perspective and its clients cannot assume the transactional data will be available after the first successful delivery.

The picture below is shown as an **Unacceptable usage** of the Provincial XML solution:



In this context the client applications implement the Provincial XML client and are connected directly to the Provincial XML server application hosted within the OACCAC environment. The third party/provider system users are also using the Provincial XML server application as a repository of provider data; requesting it from the server as part of the workflow implemented in their systems.

This solution is an unacceptable usage of the Provincial XML solution since it increases the bandwidth required for the client system to execute Provincial XML transactions, it wrongly assumes the data will permanently be available in the Provincial XML for every single transaction even after the information was already delivered to the third party/provider system in a previous request. These two factors combined create an unnecessary stress on the Provider XML server application and limits its ability to serve other legitimate requests.

In conclusion the usage of the Provider XML solution is envisioned as a server to server XML transactional system and its usage should be limited to that.

2. Service Contracts

2.1 Overview

Three service contracts are exposed to end consumers, **Get Message List**, **Get Message**, and **Post Message**. All methods require Windows Impersonation to be enabled at the client and server, and all methods are compliant with the WS-Security specification published by OASIS. (See the security overview section for more details about security and user impersonation).

Get Messages List and Get Message

The usage pattern is to first obtain a list of pending messages with the **Get Message List** method, and to then retrieve those messages by constructing **Get Message** requests for a sub-list of the messages Ids returned by **Get Message List**.

The Ids returned by **Get Message List** correspond to the 'Track IDs' which exist within HPG today. The messages returned by **Get Message** will be Provincial XML representations of existing HPG documents. For release 1.5, the following document types are supported:

- Equipment and Supply Order
- Service Referral
- Service Frequency Update
- Billing Reconciliation Report for Equipment and Supply
- Billing Reconciliation Report for Purchase Service
- Service Offer
- Provider Report Response

Post Message

The usage pattern is to construct **Post Message** request with an input document of Provincial XML Document (PXD) format. The response of the method will contain a 'Track ID' for tracking purpose. For release 1.5, the following document types are supported:

- Billing Invoice for Equipment and Supply

- Billing Invoice for Purchase Service
- Service Offer Response
- Provider Report

Subsequent releases will support other document types. The schema files for PXD documents are provided by the OACCAC.

Below is an overview of these two methods, include WSDL snippets describing the service methods. The bulk of the remaining document goes into details about request arguments, object definitions, and usage examples. These methods will initially be exposed as Web Services over the SOAP protocol.

2.2 Service Reference

For the Certification Environment:

Service reference: http://provxml-cert.apps.ccac-ont.ca/ProvincialXmlService.svc

For the Production Environment:

Service reference: http://provxml.ccac-healthpartners.ca/ProvincialXmlService.svc

2.3 Get Message List

The Get Message List method will return a list of message IDs (HPG Track IDs) corresponding to the request arguments. If no arguments are supplied, the HPG default of all pending documents for the last calendar week will be applied.

Method Signature Pseudo-code
GetMessageListResponseItemWrapper GetMessage(GetMessageListRequestItemWrapper)

WSDL message input element

```
<wsdl:message name="IProvincialXmlService_GetMessageList_InputMessage">
   <wsdl:part name="parameters" element="tns:GetMessageList"/>
</wsdl:message>
```

WSDL message output element

```
<wsdl:message name="IProvincialXmlService_GetMessageList_OutputMessage">
   <wsdl:part name="parameters" element="tns:GetMessageListResponse"/>
</wsdl:message>
```

WSDL operation element

```
<wsdl:operation name="GetMessageList">
```

<wsdl:input wsaw:Action="http://tempuri.org/IGetMessageListService/GetMessageList" message="tns:IProvincialXmlService_GetMessageList_InputMessage"/>
<wsdl:output wsaw:Action="http://tempuri.org/IGetMessageListService/GetMessageListResponse" message="tns:IProvincialXmlService_GetMessageList_OutputMessage"/>
</wsdl:operation>

Description

The GetMessageList Service Contract method defines the interface for participants to retreive lists of documents IDs according to the passed critera. A response will contain a list of Ids and reference names, one item per document. GetMessageList will return an empty list in the event that no documents were found for the given criteria arguments.

An empty criteria set will default to the last calendar week's worth of documents.

Arguments

1. **GetMessageListRequestItemWrapper** : The GetMessageListRequestWrapper is the top container argument for the GetMessageList method. It is a container of individual GetMessageListRequestItems, each of which can produce a non-empty return set. The fields for GetMessageListRequestItem and its Wrapper are described below in the Data Contracts section.

Response

1. **GetMessageListResponseItemWrapper** : The GetMessageListResponseWrapper is the top container which encapsualtes all response items for a given request. This top container also contains response status information (i.e., Success or Fail). The fields for GetMessgeListResponseItemWrapper are described below in the Data Contracts section.

2.4 Get Message

The Get Message method will return a Provincial XML document payload corresponding to a translation of an existing HPG document. The actual document types returned are content-driven. For example, a document stored as Chris-XML CCME-Equipment Supply Order will return an equivalent Provincial Xml Equipment Supply Order document.

The returned document versions are dependent upon client preference and server configuration and ability – supported versions will be agreed upon out-of-band during scheduled meetings or correspondence.

Method Signature Pseudo-Code
GetMessageResponseItemWrapper GetMessage(GetMessageRequestItemWrapper)

WSDL message input element
```
<wsdl:message name="IProvincialXmlService_GetMessage_InputMessage">
   <wsdl:part name="parameters" element="tns:GetMessage"/>
</wsdl:message>
```

WSDL message output element
```
 <wsdl:message name="IProvincialXmlService_GetMessage_OutputMessage">
   <wsdl:part name="parameters" element="tns:GetMessageResponse"/>
 </wsdl:message>
```

WSDL operation element
```
<wsdl:operation name="GetMessage">
      <wsdl:input wsaw:Action="http://tempuri.org/IGetMessageService/GetMessage"
message="tns:IProvincialXmlService_GetMessage_InputMessage"/>
      <wsdl:output wsaw:Action="http://tempuri.org/IGetMessageService/GetMessageResponse"
message="tns:IProvincialXmlService_GetMessage_OutputMessage"/>
 </wsdl:operation>
```

Description
The GetMessage Service Contract method defines the interface for participants to retreive one or more document bodies which correspond to the PXD schemas. Each response item will contia a single document with its body, routing information, and primary identifying fields. Once a document has been retrieved, it will be marked as 'processed' within HPG according to the current business rules (i.e., if a document action from the HPG interface would cause a document to be marked as processed, an equivlent acvtion through Provincial XML will also cause the message to be processed)

GetMessage will return an empty result set in the event that no documents were found for the given criteria arguments.

An empty criteria set will default to the last calendar week's worth of documents.

Arguments

1. **GetMessageRequestItemWrapper** : The GetMessageRequestWrapper is the top container argument for the GetMessage method. It is a container of individual GetMessageRequestItems, each of which has a 1-to-1 correspondance to a document to return. The fields for GetMessageRequestItem and its Wrapper are described below in the Data Contracts section

Response

1. **GetMessageResponseItemWrapper** : The GetMessageResponseWrapper is the top container which encapsualtes all response items for a given request. This top container also contains response status information (i.e., Success or Fail). The fields for GetMessgeResponseItemWrapper are described below in the Data Contracts section.

2.4.1 Get Message – changing POSTED to PROCESSED

It is mandatory that whether a message are retrieved via XML or manually through the HPG GUI that the status of specific message types changes from POSTED to PROCESSED and a processed date is populated in the HPG GUI. The message types that must change from POSTED to PROCESSED are:

- Equipment and Supply Order
- Service Referral
- Service Frequency Update
- Billing Reconciliation Report for Equipment and Supply
- Billing Reconciliation Report for Purchase Service
- Provider Report Response

Note: Service Offer messages appear as POSTED in HPG, the status does not change PROCESSED.

2.5 Post Message

The Post Message method will accept an XML document of Provincial XML Document (PXD) format as an input, translate it to a corresponding Chris-XML document type, and store the translated content to a backend database for future processing. The actual document types stored are content-driven. For example, an input document of Provincial XML Equipment Supply Billing Invoice type will be stored as a Chris-XML CCM-Equipment Supply Billing Invoice document.

Method Signature Pseudo-Code

PostMessageResponseItemWrapper PostMessage(PostMessageRequestItemWrapper)

WSDL message input element

```
<wsdl:message name="IProvincialXmlService_PostMessage_InputMessage">
   <wsdl:part name="parameters" element="tns:PostMessage"/>
</wsdl:message>
```

WSDL message output element

```
<wsdl:message name="IProvincialXmlService_PostMessage_OutputMessage">
   <wsdl:part name="parameters" element="tns:PostMessageResponse"/>
</wsdl:message>
```

WSDL operation element

```
<wsdl:operation name="PostMessage">
      <wsdl:input wsaw:Action="http://tempuri.org/IPostMessageService/PostMessage"
message="tns:IProvincialXmlService_PostMessage_InputMessage"/>
      <wsdl:output wsaw:Action="http://tempuri.org/IPostMessageService/PostMessageResponse"
message="tns:IProvincialXmlService_PostMessage_OutputMessage"/>
</wsdl:operation>
```

Description

The PostMessage Service Contract method defines the interface for participants to send one or more document bodies which correspond to the PXD schemas. Each response item contains an ID for

tracking purpose. Once a document is sent and stored to a backend database, it will be marked as 'Posted' within HPG.

Arguments

1. **PostMessageRequestItemWrapper** : The PostMessageRequestWrapper is the top container argument for the PostMessage method. It is a container of individual PostMessageRequestItems, each of which has a 1-to-1 correspondance to a document to post. The fields for PostMessageRequestItem and its Wrapper are described below in the Data Contracts section

Response

1. **PostMessageResponseItemWrapper** : The PostMessageResponseWrapper is the top container which encapsualtes all response items for a given request. This top container also contains response status information (i.e., Success or Fail). The fields for PostMessgeResponseItemWrapper are described below in the Data Contracts section.

When posting messages the status must be POSTED. There should be no processed date.

3. Data Contracts

3.1 Overview

There are three pairs of messaging data contracts; a request-response pair for each of **GetMessageList**, **GetMessage**, and **PostMessage**. Additionally, there is a data contract for response errors.

The following sections provide more details on how the requests can be constructed. Since all methods require authentication, section describes how to construct a WCF client using C#.

3.2 Construct a new Provincial XML Service Client

This snippet shows how to construct a new client, and to set the security options correctly. All examples that follow will make use of this 'method' whenever the 'GetNewClient()' called.

```
private static ProvincialXmlServiceClient GetNewClient()
{
    ProvincialXmlServiceClient toReturn = new ProvincialXmlServiceClient();
    toReturn.ClientCredentials.UserName.UserName = "{UserName supplied by OACCAC}";
    toReturn.ClientCredentials.UserName.Password = "{Password supplied by OACCAC}";
    return toReturn;

}
```

3.3 Get Message List Data Contracts

3.3.1 Get Message List Request Item Wrapper

Description
The Get Message List Request Item Wrapper is the top-level container which encapsualtes all information involved in a GetMessageRequest message.

Data Contract pseudo-code

```
GetMessageRequestItemWrapper
{
```

```
    public enum RequestVersions { OnePointZero = 1 }

    public List<GetMessageListRequestItem> RequestItems;

    public RequestVersions ContractVersion;

}
```

Data Constants
RequestVersions – An enumerated type bounding the valid values for the contract version.

Data Members
RequestItems - A list of individual GetMessageListRequestItems. The list must be non-empty to be a valid request.

ContractVersion – This is an enumerated field that depicts the version of this data contract. Future releases may refine the data contract to include new fields or restrictions to support enhanced functionality. For release 1.0 only the 'OnePointZero' enumeration is valid.

3.3.2 Get Message List Request Item

Description
The GetMessageListRequestItem object encapsulates all information necessary to query Provincial XML for a list of pending (i.e., unprocessed) HPG documents according to the given criteria. It is permissible to send multile GetMessageListRequest Items per GetMessageListRequestItemWrapper top-level request, but the system is not obligated to filter the results such that each returned list set contains only unique documents, since the document identifiers themsvels may be in different domains, both Ids and reference names for example.

Data Contract pseudo-code

```
GetMessageListRequestItem
{

    public GetMessageListFilter RequestFilter;

 }
```

Data Members
RequestFilter – An object which encapsulates criteria to return a particular set of pending documents

related in some way i.e., by date range, partial reference name, message type, etc. The GetMessageListFilter data contract id detailed in the filter section below.

3.3.3 Get Message List Examples

3.3.4 Get Message List – Using No Filters

Get a list of messages, using no filters, and iterate through the returned list

```
using (ProvincialXmlServiceClient cli = GetNewClient())
{
    GetMessageListRequestItemWrapper req = new GetMessageListRequestItemWrapper();
    req.RequestItems = new GetMessageListRequestItem[1];
    req.RequestItems[0] = new GetMessageListRequestItem();
    GetMessageListResponseItemWrapper resp = cli.GetMessageList(req);

    foreach ( GetMessageListResponseItem item in resp.ResponseItems)
        Console.Out.Write( "[MessageID] " + item.ResponseData +
                           ", [Message Name] " + item.ReferenceName +
                           ", [DestinationTeamId] " + item.ResponseFilter.MessageDestination);

}
```

3.3.5 Get Message List – Using Message Name Filter

The following example gets a list of messages using a message name filter. Message name filters will match with wildcards on the server, so the following example will find all messages with the word "**Order**" in the name, i.e. 'EquipmentSupply**Order**_1456'

It is important to note that message name is based on the HPG document description and is not the same as message type. (see HPG screen shot below for Document Description) Some message types have multiple message names.

```
using (ProvincialXmlServiceClient cli = GetNewClient())
{
    GetMessageListFilter f = new GetMessageListFilter();
```

```
f.ReferenceName  = "Order";
GetMessageListRequestItemWrapper req = new GetMessageListRequestItemWrapper();
req.RequestItems = new GetMessageListRequestItem[1];
req.RequestItems[0] = new GetMessageListRequestItem();
req.RequestItems[0].RequestFilter = f;
GetMessageListResponseItemWrapper resp = cli.GetMessageList(req);

foreach ( GetMessageListResponseItem item in resp.ResponseItems)
        Console.Out.Write( "[MessageID] " + item.ResponseData +
                        ", [Message Name] " + item.ReferenceName +
                         ", [DestinationTeamId] " + item.ResponseFilter.MessageDestination);

}
```

3.3.6 Get Message List – Specific Message Types Filter

There are a number of message type, you may want to reteive only specific message types.

Retrieving specific message types is optional.

The following example shows how the client can request a list of Service Offer and Service Referral messages.

```
using (ProvincialXmlServiceClient cli = GetNewClient())
```

```
{
    GetMessageListFilter f1 = new GetMessageListFilter();
    f1.MessageType = "ServiceReferral";
    GetMessageListFilter f2 = new GetMessageListFilter();
    f2.MessageType = "ServiceOffer";

    GetMessageListRequestItemWrapper req = new GetMessageListRequestItemWrapper();
     req.RequestItems = new GetMessageListRequestItem[2];
     req.RequestItems[0] = new GetMessageListRequestItem();
    req.RequestItems[0].RequestFilter = f1;
     req.RequestItems[1] = new GetMessageListRequestItem();
    req.RequestItems[1].RequestFilter = f2;
     GetMessageListResponseItemWrapper resp = cli.GetMessageList(req);

    foreach ( GetMessageListResponseItem item in resp.ResponseItems)
            Console.Out.Write( "[MessageID] " + item.ResponseData +
                        ", [Message Name] " + item.ReferenceName +
                        ", [DestinationTeamId] " + item.ResponseFilter.MessageDestination);

}
```

3.3.7 Get Message List – Using Date Filters

The following example gets a list of messages using a date filter. All messages between March 15 and April 1, 2009, will be queried.

```
using (ProvincialXmlServiceClient cli = GetNewClient())

{
    GetMessageListFilter f = new GetMessageListFilter();
    f.EndDate = DateTime.Parse("2009-04-01");
     f.StartDate = DateTime.Parse("2009-03-15");
    GetMessageListRequestItemWrapper req = new GetMessageListRequestItemWrapper();
     req.RequestItems = new GetMessageListRequestItem[1];
     req.RequestItems[0] = new GetMessageListRequestItem();
```

```
        req.RequestItems[0].RequestFilter = f;
        GetMessageListResponseItemWrapper resp = cli.GetMessageList(req);

        foreach ( GetMessageListResponseItem item in resp.ResponseItems)
                Console.Out.Write( "[MessageID] " + item.ResponseData +
                                ", [Message Name] " + item.ReferenceName +
                                ", [DestinationTeamId] " + item.ResponseFilter.MessageDestination);
}
```

3.3.8 Get Message List Response Item

Description
The GetMessageListResponseItem object encapsulates a single returned document instance. The information returned is analagous to to the document header, in that a GetMessageListResponseItem contains all necessary information to uniquely identify a document instance, and a subsequent GetMessageRequestItem can be constructed to fetch the document contents translated into PXD schema format.

Data Contract pseudo-code

```
GetMessageListResponseItem
{
    public readonly String ResponseData;
    public readonly String ReferenceName;
    public readonly GetMessageListFilter ResponseFilter;
}
```

Data Members
ResponseData – A GUID string which uniquely represents the returned document in Provincial XML and HPG. This is equivalent to the Track Id in HPG.

ReferenceName – A user-supplied name which represents this document, not guaranteed to be unique. Usually descriptive, i.e., 'EquiplentSupplyOrder1234'

ResponseFilter – The Response Filter.

i) **ResponseFilter.MessageDestination** – The Destination's TeamID.

3.4 Get Message Data Contracts

3.4.1 Get Message Request Item Wrapper

Description
The Get Message Request Item Wrapper is the top-level container which encapsualtes all information involved in a GetMessageRequest message.

Data Contract pseudo-code

```
GetMessageRequestItemWrapper
{
    public enum RequestVersions { OnePointZero = 1 }
    public List<GetMessageRequestItem> RequestItems;
    public RequestVersions ContractVersion;
}
```

Data Constants
RequestVersions – An enumerated type bounding the valid values for the contract version.

Data Members
RequestItems - A list of individual GetMessageRequestItems. The list must be non-empty to be a valid request.

ContractVersion – The Contract version is an enumerated field that depicts the version of this data contract. Future releases may refine the data contract to include new fields or restrictions to support enhanced functionality. For release 1.0 only the 'OnePointZero' enumeration is valid.

3.4.2 Get Message Request Item

Description
The Get Message Request Item represents a request for a single document by Id.

Data Contract pseudo-code

```
GetMessageRequestItem
 {
      public String RequestData;
      public String RequestSenderType;
      public String RequestSenderId;
      public String ExpectedResponseVersion;
}
```

Data Members
RequestData - The request data represents the unique ID for the specific entity to operate upon. For HPG messages, it corresponds to the Message TrackID.

RequestSenderType – Always set to 'Team' to correlate to **RequestSenderId** (i.e. TeamId)

RequestSenderId - The request sender ID represents the TeamID for the requestor retrieving the specific message. According to the business rule, this field must be non-empty and valid in order to mark the "Posted" message "Processed"

ExpectedResponseVersion – The response version represents the version of the payload of a message the requestor expects to retrieve. Below is the message type currently supported in Release 1.4 with multi-versioning capability:

a) ServiceReferral

3.4.3 Get Message Examples

The Get Message method will return the actual payload of a message, in Provincial XML document format.

3.4.4 Get Message – For a Specific Message ID

The following example will get a single message with an explicit Message ID, equivalent to the GUID with value '06c8a5d9-49e0-4dd6-8f3a-548d53e7c21c'.

This implementation only retrieves the message, without changing the status from "Posted" to "Processed". Because the status is not changed to Procesed this example should be used for testing only.

```
using (ProvincialXmlServiceClient cli = GetNewClient())
{
        GetMessageRequestItemWrapper req = new GetMessageRequestItemWrapper();
        req.RequestItems = new GetMessageRequestItem[1];
        req.RequestItems[0] = new GetMessageRequestItem();
        req.RequestItems[0].RequestData = "06c8a5d9-49e0-4dd6-8f3a-548d53e7c21c";
        req.RequestItems[0].RequestSenderType = "Team";
        GetMessageResponseItemWrapper resp = cli.GetMessage (req);

        foreach( GetMessageResponseItem item in resp.ResponseItems)
            Console.Out.Write( item.ResponseData );
}
```

3.4.5 Get Message – Change Status to PROCESSED

The following example will get a single message with an explicit Message ID, as well as marking it "Processed" with the input of TeamID.

Changing the status of a message from POSTED to PROCESSED is mandatory for implementation in the production environment.

```
using (ProvincialXmlServiceClient cli = GetNewClient())
{
        GetMessageRequestItemWrapper req = new GetMessageRequestItemWrapper();
        req.RequestItems = new GetMessageRequestItem[1];
        req.RequestItems[0] = new GetMessageRequestItem();
        req.RequestItems[0].RequestData = "06c8a5d9-49e0-4dd6-8f3a-548d53e7c21c";
```

```
        req.RequestItems[0].RequestSenderType = "Team";
        req.RequestItems[0].RequestSenderId = "{Requestor's TeamID (eg. 6092)}";
        GetMessageResponseItemWrapper resp = cli.GetMessage (req);

        foreach( GetMessageResponseItem item in resp.ResponseItems)
            Console.Out.Write( item.ResponseData );
    }
```

3.4.6 Get Message – By Specific Version

The message schemas will change over time. To give providers/vendors opportunity to make necessary changes in their systems multiple versions of the same message type will be supported. It is important to note that if the message version is not specified you will retreive the oldest version of the message. For example: if there are versions 1.1; 1.2 and 1.3 of a schema and you do not specifiy a version you will retreive 1.1 by default. If on the other hand you are not concerned about the new message versions being released and you would like to retrieve the latest (most recent) message versions you can specify a high decimal value, such as 99.0 or Decimal.MaxValue to automatically request them.

Retrieving specific message types is not mandatory but it is highly suggested.

The following example will get a single message with an explicit Message ID, and the contents of the message can be different based on the **ExpectedResponseVersion** specified.

```
using (ProvincialXmlServiceClient cli = GetNewClient())
{
        GetMessageRequestItemWrapper req = new GetMessageRequestItemWrapper();
        req.RequestItems = new GetMessageRequestItem[1];
        req.RequestItems[0] = new GetMessageRequestItem();
        req.RequestItems[0].RequestData = "06c8a5d9-49e0-4dd6-8f3a-548d53e7c21c";
        req.RequestItems[0].RequestSenderType = "Team";
        req.RequestItems[0].ExpectedResponseVersion = "{Payload Version (eg. 2.15)}"
        GetMessageResponseItemWrapper resp = cli.GetMessage (req);

        foreach( GetMessageResponseItem item in resp.ResponseItems)
            Console.Out.Write( item.ResponseData );
    }
```

If requesting more then one message you can specify multiple GetMessageRequestItem elements.

3.4.7 Get Message Response Item

Description
The GetMessageResponseItem encapsulates all of the fields representing an entire document, including the XML payload.

Data Contract pseudo-code

```
GetMessageResponseItem
 {
      public readonly String ResponseData;
      public readonly String SenderId;
      public readonly String SenderType;
      public readonly String DestinationId;
      public readonly String DestinationType;
      public readonly String ReferenceId;
      public readonly String ReferenceName;
}
```

Data Members
ResponseData - The Response Data field contains the entire XML document, encoded in UTF-8.

SenderId- The Sender's ID - i.e., the Team ID. The empty string if not applicable or error.

SenderType – The Sender's Type - i.e., 'Team'. The empty string if not applicable or error

DestinationId – The Destination's ID - i.e., the Team ID. The empty string if not applicable or error.

DestinationType – The Destination's Type - i.e., 'Team'. The empty string if not applicable or error.

ReferenceId – The Referernce Id is a unique (to the applicable domain) identifier for this item. In the case of messages this is equivlent to the HPG TrackID. The empty string if not applicable or error.

ReferenceName – Descriptive text for this item. For XML messages, this is quivlent to the HPG Messagevname. The emptry string if not applicable or error.

3.5 Post Message Data Contracts

3.5.1 Post Message Request Item Wrapper

Description
The Post Message Request Item Wrapper is the top-level container which encapsualtes all information involved in a PostMessageRequest message.

Data Contract pseudo-code

```
PostMessageRequestItemWrapper
{
    public enum RequestVersions { OnePointZero = 1 }
    public List<PostMessageRequestItem> RequestItems;
    public RequestVersions ContractVersion;
}
```

Data Constants
RequestVersions – An enumerated type bounding the valid values for the contract version.

Data Members
RequestItems - A list of individual PostMessageRequestItems. The list must be non-empty to be a valid request.

ContractVersion – The Contract version is an enumerated field that depicts the version of this data contract. Future releases may refine the data contract to include new fields or restrictions to support enhanced functionality.

3.5.2 Post Message Examples

The Post Message method will accept an XML document of Provincial XML Document (PXD) format as an input, translate it to a corresponding Chris-XML document type, and post the translated content to HPG backend database.

The following example will post an XML message with *<Message Reference Name>,* and will return the unique HPG Track ID to the sender.

```
using (ProvincialXmlServiceClient cli = GetNewClient())
{
    PostMessageRequestItemWrapper req = new PostMessageRequestItemWrapper();
    req.RequestItems = new PostMessageRequestItem[1];
    req.RequestItems[0] = new PostMessageRequestItem();
    req.RequestItems[0].RequestData = "{Provincial XML Message to Post}";
    req.RequestItems[0].RequestReferenceName = "Billing Invoice 12345 - ES";
    PostMessageResponseItemWrapper resp = cli.PostMessage (req);

    Foreach( PostMessageResponseItem item in resp.ResponseItems)
        Console.Out.Write( item.ReferenceId );
}
```

3.5.3 Post Message Request Item

Description
The Post Message Request Item represents a request for posting a single document into HPG database.

Data Contract pseudo-code

```
PostMessageRequestItem
{
    public String RequestData;
    public String RequestReferenceName;
}
```

Data Members
RequestData - The Request Data field contains the entire XML document to post.

RequestReferenceName – this is equivalent to the HPG Message Name. The emptry string if not applicable or error.

3.5.4 **Post Message Response Item**

Description
The PostMessageResponseItem encapsulates fields uniquely representing the posted document, including HPG Track ID.

Data Contract pseudo-code

```
PostMessageResponseItem
{
    public readonly String ReferenceId;
}
```

Data Members
ReferenceId – the Track Id in HPG, which uniquely represents the posted document in Provincial XML and HPG

3.6 **Filter Data Contracts**

Filters represent modifications or limits to request/response items. Initially, they are intended to allow for an encapsulation of message querying criteria. In future releases, they will be extended to allow for partial acceptance of messages and reporting on excluded transaction Ids.

3.6.1 **GetMessageListFilter**

Description
The GetMessageListFilter encapsultes a number of fields which can be used to specify the kind of messages a consumer will wish to download and process. If multiple criteria are specified, they are combined to produce a result set in which all criteria must be satisfied. Messages which partially match the crtieria will be excluded.

Data Contract pseudo-code

```
GetMessageListFilter
{
```

```
    public DateTime StartDate;
    public DateTime EndDate;
    public MessageStatusEnum Message Status;
    public String MessageType;
    public String MessageDestination;
    public String MessageSource;
    public String MessageAuthor;
    public String ReferenceName;
}
```

Data Members

StartDate – Only messages with a timestamp after the start date will be returned. The exact comparison is greater-than-or-equal-to.

EndDate – Only messages with a timestamp before the end date will be returned. The exact comparison is greater-than-or-equal-to.

MessageStatus – Only messages with the particular status will be returned. This is equivlent to the HPG delivery status. An Enumerated type of 'Posted' {0} or 'Processed' {1} is provided, the numeric equivlents correspond to the ones in HPG currently

MessageType – Only messages of the particular types will be returned. This is equivlent to the HPG message type, and must be equivlent to the type string. Below are the valid type strings currently supported in Release 1.4:

b) EquipmentSupplyOrder

c) ServiceReferral

d) ServiceFrequencyUpdate

e) BillingRAEquipmentSupply

f) BillingRAPurchaseService

g) ServiceOffer

MessageDestination – Only messages with the TeamID matching messageDestination will be returned. For response filters, equal to the TeamID of the destination. Not compatible with any other filter being set as a *request* filter.

MessageSource – The Message's source. This value should be equivlent to the HPG 'Message Source String'.Id.

MessageAuthor – The ID of the user which authored the message.

ReferenceName – A String which will be partially matched (i.e., it will implicitly have wildcards on each end) against message reference names in the user's inbox. No spaces or none alpha-numeric characters are accepted.

3.7 Error Response Data Contracts

All wrappers and message items contain response error fields. In the event of most application errors, these objects will be populated accordingly. Errors will contain the following information

Data Contract pseudo-code

```
ErrorResponse
{
    public enum ErrorType { … Described below… }
    public String ErrorDescription
    public String ErrorID
}
```

Data Members

Error Type: An enumeration, detailed as follows:

'Request Type Invalid' – if a service contract was invoked with an incorrect request object

'Request Parameters Invalid' – If insufficient parameter or filter information was contained in the request to construct a meaningful response. For example, if a get message request is received without a filter or reference Id.

'Invalid DEWS User' – If the Windows user impersonation cannot find a valid HPG/DEWS account. Correspondence with OACCAC support is recommended if this error is encountered.

'Filter Parameter Error' – If a filter contains invalid information. i.e., if the start date is after the end date for a date range criteria. =

'Invalid HPG Xml' – If an HPG document cannot be translated into Provincial XML Format.

'General' – Other kind of error. The error description will contain details

Error Description: will always be populated with detailed information regarding the error.

Error ID: In addition, a GUID will be returned with each error – this GUID, called the 'Error ID' is also persisted on the server side and can be used to correlate client errors with server activity. The Error ID should be provided every time communication occurs with the Provincial XML support team.

In the event of more general system-level errors, the service contracts have implemented a WCF Fault Contract interface to return an Application Exception. No unhandled exceptions will be thrown by the Provincial XML service.

4. Security Overview

4.1 Overview

Provincial XML utilizes the WS-Security standard to provide signing and encryption envelopes around SOAP messages. Authentication will be handled by Windows Active Directory (AD) authentication and impersonation.

Authentication: Users are authenticated against an Active Directory (AD) domain which has been set up specifically to handle authentication and authorization for the Provincial XML solution. Users will be identified through a TSL secured security context as specified in the WS-Security message-based security specification (OASIS). The client code examples in section 5 demonstrate how to set the username credentials on the client before a message is constructed and transported.

Confidentiality and Integrity: The Provincial XML Solution will use message-based encryption and digital signature algorithms to ensure that confidentiality and integrity is achieved. The message-level security conforms to the WS-Security specification, the exact type of which is 'Message' No key storage or key generation requirements are placed upon the client, they are negotiated at transport time. The 'Basic-256' algorithm suite is currently used (AES-256, SHA1, RAS 1024)

5. Client Configuration

5.1 Client Configuration Overview

A default configuration file will be provided with the Provincial Xml Service Client distribution. The configuration is compliant with WCF client/service configuration files, and can be edited with the standard WCF service configuration editor provided by Microsoft.

Below is an *example* configuration for production settings. The OACCAC may advise that the below values will change before production or testing phases. Fields in <green> are in-line comments on particular configuration items which are common to all service-oriented-architecture (SOA) clients, and are intended to help configure any type of client, not just Microsoft or WCF ones. Fields in <red> indicate server URLs or Service Principal Names (SPNs) for the endpoint Provincial XML Service - the ones below are examples only, and are not intended to represent any UAT or production environment.

NB: Meta-Data is now dynamically available for the service, using a web browser navigate to the endpoint URL and instructions will be provided to programmatically obtain the WSDLs and XSD files for the service.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <system.serviceModel>
  <bindings>
   <wsHttpBinding>
    <binding name="ProvincialXmlWSBinding" maxReceivedMessageSize="1048576">

     <security mode="Message" >
      <message clientCredentialType="UserName" establishSecurityContext="false"/>
     </security>
    </binding>
   </wsHttpBinding>
  </bindings>
```

```
  <client>
   <endpoint address="https://example.ccac-ont.ca/ProvincialXmlProxy/ProvincialXmlService.svc"
         binding="wsHttpBinding" bindingConfiguration="ProvincialXmlWSBinding"
         behaviorConfiguration="ProvincialXmlClientBehavior"
         contract="ProvincialXml.Client.IProvincialXmlService" name="ProvincialXmlClientBinding">
    <identity>
     <The DNS settings will only need to be adjusted in the case that the endpoints DNS doest not
EXACTLY equal the common name (CN) in the server certificate.  In prodiction environments this element
should be removed. >
      <dns value="*.provxmltest.example.ccac-ont.ca"/>
    </identity>
   </endpoint>
  </client>
  <behaviors>
   <endpointBehaviors>
    <behavior name="ProvincialXmlClientBehavior">
     <clientCredentials>
      <In order for Windows to propreley authenticate the client's Kerberos tokens, delegation must be
explicitly allowed at the client end, and NTLM authentication must be turned off>
       <windows allowedImpersonationLevel = "Delegation"  allowNtlm = "False" />
     </clientCredentials>
    </behavior>
   </endpointBehaviors>
  </behaviors>
 </system.serviceModel>
</configuration>
```