Creation Date: March 26, 2010

 Last Updated:

 Revision: 1.0

# Revision History

| **Date** | **Author** | **Description of Change** |
| --- | --- | --- |
| March 26, 2010 | Brad Hodge - PASI Data Architect | Initial Draft |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Audiences

**Involvement Types: R -** review ; **A -** approve ; **C -** consume; **I -** Informational

|  |  |  |  |
| --- | --- | --- | --- |
| **Audience**  | **Involvement Type** | **Performed Involvement On** |  **Notes** |
| PASI Core | R | April 8, 2010 | Initial review |
| PASI Prep | R |  |  |
| Ministry Client | R |  |  |
|  |  |  |  |
|  |  |  |  |

# table of contents

[1. Revision History 1](#_Toc258394597)

[2. Audiences 2](#_Toc258394598)

[3. table of contents 3](#_Toc258394599)

[4. Introduction 4](#_Toc258394600)

[4.1 Purpose 4](#_Toc258394601)

[4.2 Scope 4](#_Toc258394602)

[5. Indicating Requirement Levels 5](#_Toc258394603)

[6. Versioning Overview 6](#_Toc258394604)

[6.1 Service Contracts 6](#_Toc258394605)

[6.2 Data Contracts 7](#_Toc258394606)

[7. PASI Versioning Guidelines 9](#_Toc258394607)

[7.1 General 9](#_Toc258394608)

[7.2 Service Contracts 9](#_Toc258394609)

[7.3 Data Contracts 11](#_Toc258394610)

[7.4 Service Level Agreements 12](#_Toc258394611)

# Introduction

This document provides an overview of the versioning strategy used by PASI Core to publish its Service and Data contracts.

### Purpose

As indicated above, the main goal of the document is to effectively communicate the versioning strategy used by PASI Core to publish its Service and Data contracts to its internal and external stakeholders. These stakeholders, including technical staff within the PASI Core, PASI Prep, and Ministry Client project teams as well as external consumers of the PASI Core services will be able to review the strategy and evaluate its adequacy from the perspective of their individual areas of subject matter expertise.

The document achieves its purpose by providing an overview of the relevant versioning considerations for both Service and Data contracts, focusing on the most significant elements within each. In addition, it describes the significant decisions made within each area of consideration and the impacts of those decisions.

### Scope

The information contained within this document is intended to illustrate the practices that PASI will follow when versioning the PASI Core services. While providing relevant overviews, the document is not meant to provide a complete and in depth technical description of the underlying Windows Communication Foundation (WCF)[[1]](#footnote-1) infrastructure used to implement the functionality.

# Indicating Requirement Levels

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in this section.

**“MUST** This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the standard.

**MUST NOT** This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the standard.

**SHOULD** This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

**SHOULD NOT** This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

**MAY** This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular application requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, MUST be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides).”

# Versioning Overview

This section provides an overview of the different versioning considerations for service and data contracts.

### Service Contracts

WCF contracts are version tolerant by default, meaning that WCF services allow for non-required data to be missing while also ignoring superfluous data for service and data contracts. This default version tolerance necessitates careful consideration when versioning contracts. Service contract versioning is generally defined using the following three categories:

* Data Contract changes
* Service Implementation changes
* Service Address and Binding changes

**Data Contract Changes**

Covered in the next section, data contract changes directly affect the versioning of the service contracts that expose them. In practical terms, when strict schema validation is not required a service contract does not need to be versioned if a non-breaking change is made to a related data contract. This allows the data contracts to evolve separately from the service contract and maintains compatibility with clients using earlier versions of the data contracts.

However, if a breaking change is introduced in a related data contract, a new service and service contract must be introduced if backward compatibility is to be maintained for clients of older versions.

**Service Implementation Changes**

Implementation changes are associated with the addition, subtraction, and/or changing of existing service operations and their signatures.

Adding new service operations to an existing service is considered a non-breaking change as existing clients need not consume the new operations.

Removing service operations from an existing service contract is a breaking change as existing clients do expect the operations to be available. To introduce such change while maintaining backward compatibility, a new service contract must be exposed on a new endpoint.

Changing existing service operation signatures is also considered a breaking change; whether the change is to the operation’s parameters or return type. Much like removing service operations, to introduce a change to an operation’s signature and maintain backward compatibility with existing clients, a new service contract must be exposed on a new endpoint.

**Service Address and Binding Changes**

Changes to the service endpoint and bindings are always breaking changes unless a Universal Discovery Description and Integration (UDDI) registry is used by a client to dynamically discover the new endpoint address or bindings for the service.

### Data Contracts

Data contract schemas produced by WCF make no explicit provisions for versioning besides the fact that data elements are marked as optional by default. Thus, changes to data contracts can be breaking or non-breaking, depending upon the nature of the change and whether strict schema validation is required. If strict schema validation is required, data contracts should be considered immutable; meaning that if versioning is required a new data contract should be created with a different name and namespace.

*Original version*

[DataContract(Name = "StudentLocator", Namespace = "http://education.alberta.ca/PASI/2009")]

 public class StudentLocator

 {

 [DataMember(Name = "StateProvinceId", IsRequired = false, Order = 2)]

 public string StateProvinceId { get; set; }

*New version – Order and IsRequired properties have changed*

[DataContract(Name = "StudentLocator", Namespace = "http://education.alberta.ca/PASI/2010")]

 public class StudentLocator

 {

 [DataMember(Name = "StateProvinceId", IsRequired = true, Order = 1)]

 public string StateProvinceId { get; set; }

In practice, strict schema compliance is rarely required and numerous changes can be made to a data contract without affecting is compatibility. However, there are some changes that will always break compatibility and they are listed below:

* Changes to the *Name* or *Namespace* of a data contract.

[DataContract(Name = "StudentLocator", Namespace = "http://education.alberta.ca/PASI/2009")]

public class StudentLocator

* Changes to the order of the data memebers of a data contract when using the *Order* property of the DataMemberAttribute.

[DataMember(Order = 1)]

public string FirstName { get; set; }

* Changing the Name of a data member either using the *Name* property of the DataMemberAttribute, or the name of the data member field when not using the Name property.

[DataMember(Name = "FirstName")]

public string FirstName { get; set; }

* Changing the type of a data contract or a data member, such as changing a data member to a string from an integer.

The following changes are also possible and may or may not be considered breaking changes if strict schema validation is not required.

**Adding or Removing Data Members**

If data members are not marked as required elements on a data contract, they can be added or subtracted from the data contract without introducing a breaking change.

When a data contract with extra data elements is deserialized into a type missing those elements, the extra elements are ignored. Conversely, when a data contract with missing data elements is deserialized into a type with those extra elements, the extra elements are set to their default value (usually NULL or 0).

**Required Data Members**

Required data members on a data contract are marked by setting the *IsRequired* property of the DataMemberAttribute to *true*.

[DataMember(IsRequired = true)]

public string FirstName { get; set; }

If required members are missing, an exception is thrown during serialization. The addition of a required data member is a breaking change, meaning that the latest version can be sent to a service endpoint expecting the previous type but not the other way around. Conversely, removing a member that was required in previous versions of the contract is also a breaking change for an endpoint that accepts the previous version of the contract.

**Default Values**

It is possible to enable default values for a field by setting the *EmitDefaultValue* property on the DataMemberAttribute to *true*.

[DataMember(EmitDefaultValue = true)]

public string FirstName { get; set; }

 If *EmitDefaultValue* is *false*, the data member will not be emitted if it is set to its default value for that specific data type (usually NULL or 0). There are two main considerations that affect version compatibility:

* A data contract with members that are marked as required cannot receive default data from a different version of the contract in which the members have *EmitDefaultValue* set to *false*.
* A required data member that has *EmitDefaultValue* set to *false*cannot be used to serialize its default value, but can receive such a value on deserialization.

**Enumerations**

Adding or removing enumerations is considered a breaking change. Likewise, changing the name of an enumeration is also a breaking change unless the *Value* as published in the schema is maintained by using the EnumMemberAttribute.

public enum NameType : byte

 {

 [EnumAttribute()] Unknown,

 [EnumAttribute(Value="Legal", DatabaseValue=12010)] Legal,

 [EnumAttribute(Value="Alias", DatabaseValue=12020)] Alias

 } **a**

# PASI Versioning Guidelines

This section describes the guidelines by which PASI will implement service and data contract versioning.

### General

* Acknowledging that PASI clients must be considered autonomous of the PASI services, maintaining backwards compatibility with existing clients is a requirement when non-breaking changes are introduced. This means that PASI does not support strict schema validation.
* Where breaking changes cannot be avoided, PASI will expose the new service capabilities via a new endpoint if it is feasible for older clients to continue using the previous version. These services can be exposed concurrently through different endpoints until such time that the previous version(s) can be retired.

### Service Contracts

* New service operations that are not critical to existing business functionality will be appended to existing services. Clients reliant upon older versions of the contract will be unaffected while the new functionality will be available for newer clients.
* PASI will publish its services under explicit Namespaces in order to avoid the usage of the default <http://tempuri.org> namespace.

 [ServiceBehavior(Namespace:="http://education.alberta.ca/PASI/")]

Public Class PASIService…

* If there are changes to existing service operation signatures, or if existing operations have been removed, PASI will publish a new version of the service via a new endpoint. The new service will be available concurrently with the previous versions until such time that the previous version can be retired. New services will utilize updated Namespaces to avoid confusion with previous versions.

[ServiceBehavior(Namespace:="http://education.alberta.ca/PASI/2010")]

Public Class PASIService…

* PASI will not be utilizing a UDDI registry at this time, thus any changes to the endpoint addresses or bindings of the PASI services will constitute a breaking change and will need to be communicated to all clients.



Figure 7‑1 Service Contract Decision Tree

### Data Contracts

* PASI will set the *Name* property of the DataContractAttribute and DataMemberAttribute. This will insulate clients from changes to the names of the underlying types and members of the contract.

[DataMember(Name = "LastName")]

public string FirstName { get; set; }

maintains compatibility as it changes to:

[DataMember(Name = "LastName")]

public string SurName { get; set; }

* PASI will not support Forward-Compatible data contracts[[2]](#footnote-2)
* The *Order* property of the DataMemberAttribute will not be used in most cases as changes to this explicit ordering is considered a breaking change. The *Order* will only be set on data contracts where it provides clarity to the usage of the data contract that cannot otherwise be conveyed through indicating if the member is required.

[DataContract(Name = "StudentLocator", Namespace = "http://education.alberta.ca/PASI/2009")]

 public class StudentLocator

 {

 [DataMember(Name = "StateProvinceId", IsRequired = false, Order = 2)]

 public string StateProvinceId { get; set; }

 [DataMember(Name = "Name", IsRequired = true, Order = 0)]

 public NameCriteria Name { get; set; }

 [DataMember(Name = "Age", IsRequired = true, Order = 1)]

 public AgeCriteria Age { get; set; }

* If a new, non-business critical data member is added to a data contract, the *IsRequired* property of the DataMemberAttribute will be set to *false*.
* If a new business critical data member is added to a data contract, the *IsRequired* property of the DataMemberAttribute will be set to *true* and a new version of the service will be published.
* Default values will not be enabled on data members and each data member will omit the *EmitDefaultValue* property from the DataMemberAttribute. This will have the same effect as setting the *EmitDefaultValue* property to *false.*
* Enumerations will not be used in data contracts as changes to enumerations constitute a breaking change to the service.



Figure 7‑2. Data Contract Decision Tree

1. Windows Communication Foundation - http://msdn.microsoft.com/en-us/netframework/aa663324.aspx [↑](#footnote-ref-1)
2. Forward-Compatible Data Contracts - http://msdn.microsoft.com/en-us/library/ms731083.aspx [↑](#footnote-ref-2)