Microsoft Health Connection Engine 2.1

Architecture and Design Guide

11/1/2006

Microsoft

Table of Contents

[Preamble: History and Terminology 1](#_Toc151809748)

[What is the Connected Health Framework (CHF)? 1](#_Toc151809749)

[What is the Canadian Connected Health Platform (CHP)? 1](#_Toc151809750)

[What is the Health Connection Engine (HCE)? 1](#_Toc151809751)

[Overview 2](#_Toc151809752)

[Health Connection Engine (HCE) Background 2](#_Toc151809753)

[The Strategic Challenge for Connected Health 2](#_Toc151809754)

[HCE Description 3](#_Toc151809755)

[Design Principles 3](#_Toc151809756)

[What HCE is not 4](#_Toc151809757)

[Health Connection Engine Service Blocks 4](#_Toc151809758)

[Connection Engine Message 8](#_Toc151809759)

[Message Structure 8](#_Toc151809760)

[Message Schema 10](#_Toc151809761)

[Message Management Services 12](#_Toc151809762)

[Routing Service 12](#_Toc151809763)

[Implementation Overview 13](#_Toc151809764)

[Message Flow 14](#_Toc151809765)

[BizTalk Configuration 17](#_Toc151809766)

[Adapters 18](#_Toc151809767)

[Purpose and Design Principles 18](#_Toc151809768)

[Overview 18](#_Toc151809769)

[Implementing an Adapter 19](#_Toc151809770)

[Adapter Base Class 19](#_Toc151809771)

[Adapter Classes 21](#_Toc151809772)

[Adapter Web Service 23](#_Toc151809773)

[Adapter Message Flow 24](#_Toc151809774)

[SystemAdapterBase – Instantiating an Adapter 24](#_Toc151809775)

[SendConnectionEngineMessage – Sending a Message 25](#_Toc151809776)

[ReceiveConnectionEngineMessage – Receiving a Message 25](#_Toc151809777)

[GetInteractiveSessionInformation -- Generating an Interactive Session URL 27](#_Toc151809778)

[Configuration Parameters 27](#_Toc151809779)

[Exception Handling 28](#_Toc151809780)

[Registers and Associated Services 29](#_Toc151809781)

[Overview 29](#_Toc151809782)

[HCE Registers 30](#_Toc151809783)

[Service Provider Register 31](#_Toc151809784)

[Invocation Register 38](#_Toc151809785)

[Purpose 38](#_Toc151809786)

[Design Principles 38](#_Toc151809787)

[Workflow 38](#_Toc151809788)

[Infrastructure Services 39](#_Toc151809789)

[Security Envelope 39](#_Toc151809790)

[Exception Management, Logging and Auditing Services 39](#_Toc151809791)

[Purpose 39](#_Toc151809792)

[Design Principles 39](#_Toc151809793)

[Change Notification Service 41](#_Toc151809794)

[Purpose 41](#_Toc151809795)

[Design Principles 41](#_Toc151809796)

[Additional Design Patterns 42](#_Toc151809797)

[Overview 42](#_Toc151809798)

[Health Domain Registers Overview 42](#_Toc151809799)

[Data Models 42](#_Toc151809800)

[Patient, Practitioner, and ServiceProvider 43](#_Toc151809801)

[Patient Register 43](#_Toc151809802)

[Practitioner Register 45](#_Toc151809803)

[Asynchronous Messaging, Store and Forward and Conversation Chaining 47](#_Toc151809804)

[Purpose 47](#_Toc151809805)

[Design Principles 47](#_Toc151809806)

[“Pull” Functionality 48](#_Toc151809807)

[Configuration 48](#_Toc151809808)

[Connection Engine Adapter 49](#_Toc151809809)

[Purpose 49](#_Toc151809810)

[Design Principles 49](#_Toc151809811)

[Connection Engine Message Translation Service 50](#_Toc151809812)

[Purpose 50](#_Toc151809813)

[Design Principles 50](#_Toc151809814)

[Appendix 1 – Connection Engine Message XML Schema 51](#_Toc151809815)

[Appendix 2 – Health Connection Engine Service Index 63](#_Toc151809816)

[Appendix 3 – Message Types 65](#_Toc151809817)

[Summary of support message types 65](#_Toc151809818)

[Appendix 4 – Routing Service BizTalk Implementation 69](#_Toc151809819)

[Appendix 5 – Adapter Base Class Diagram 76](#_Toc151809820)

[Appendix 6 – ProcessMessage source code 77](#_Toc151809821)

# Preamble: History and Terminology

## What is the Connected Health Framework (CHF)?

The **Microsoft Connected Health Framework** (**CHF**) ***Architecture and Design Blueprint*** provides generic and scenario-specific recommendations illustrating how to design, develop, deploy, and operate an architecturally sound application portfolio and interoperability infrastructure in a Healthcare environment. It offers deep technical guidance based on real-world experience that goes far beyond typical white papers.

As such it provides a conceptual framework for the Health Connection Engine (HCE) that addresses specific connectivity issues in particular domains. CHF thus provides potential guidance in areas not addressed by HCE.

## What is the Canadian Connected Health Platform (CHP)?

This is a planned version of HCE specifically focused on the needs of the Canadian Federal Government Infoway program, establishing inter alia the Health Information Access Layer (HIAL) which seeks to enable longitudinal patient records across a number of Healthcare Domains in each Province.

## What is the Health Connection Engine (HCE)?

HCE was originally conceived as a demonstration environment to show Microsoft clients how collaborative and connected health could be achieved. That was HCE 1.0, connecting the health applications of 6 vendors.

Subsequently, as HCE 2.0 it was enhanced to be production ready for implementation in a New Zealand regional community setting to connect applications, and in three specialist care facilities.

At the same time work has been undertaken to identify potential requirements for demonstration in Canada (in a HIAL context) and elsewhere. As particular Health Connectivity challenges are identified, potential design approaches have been considered, and where appropriate have been incorporated in this document.

This document and the accompanying code is thus a mix of:

* Production ready design and code that can be tailored to the user’s requirements
* Demonstration ready design and code that can provide a base for extension (mainly Section ).
* Design patterns (mainly in Section ) that can be considered for adoption when considering similar issues. In that regard, reference should also be made to the Connected Health Framework as that may provide different approaches for consideration. The Design Principles in Section also provide an important context for why HCE approaches particular design issues in the way it does.

As far as possible, sections of this document have been marked to clarify the maturity of the material in terms of production readiness.

# Overview

## Health Connection Engine (HCE) Background

The efficient transfer of health information between and amongst healthcare providers is one of the greatest challenges the health sector faces. Healthcare provider information systems vary considerably in terms of modernity and sophistication, from paper-based systems to highly sophisticated web-based integrated information management tools. Information needs vary from practice to practice.

HCE was developed as a consequence of a project to demonstrate how Microsoft’s Collaborative Health strategy could be brought to life. The project involved the integration of applications from 6 NZ Microsoft ISV’s, in such as way as to also provide the means for plug-and-play of applications from other Microsoft partners around the world. It followed the journey of a Type 2 Diabetic patient through primary, secondary and tertiary settings of care.

## The Strategic Challenge for Connected Health

One of the most significant IT challenges facing larger organizations today is determining how to address evolution of the application architecture.

This applies both to those that selected integrated enterprise applications in the expectation they would cover the full functionality required, and would be readily upgradeable over time, and those who have gone for integration of “best of breed”.

To their dismay, the purchasers of enterprise applications have found that upgrading the whole suite is such a major, costly and disruptive project, that they avoid doing so unless absolutely necessary. Consequently, best of breed and other point solutions start to appear to address urgent needs, and need to be integrated with the enterprise application. Meanwhile, those who purchased best of breed solutions initially have found the complexity of the application integration increasing. Whilst in most cases they have used integration middleware, rather than the hand crafted interfaces used historically, the mapping is still necessarily individual application focused, and with complex changes needed for changed applications.

For these reasons, a number of the major enterprise application vendors have recognized they need to adopt a component approach to their applications, allowing the connectivity to work in such a way that organizations can upgrade individual components, rather than the whole suite. Their approach to this has generally been to adopt a service orientated architecture based on web services. In parallel with that, application integration architects have been considering similar approaches to reduce the integration complexity.

With the range of clinical support systems in the Health sector, the integration challenges are magnified, despite the positioning of some major vendors as “the” answer.

Although service orientated architectures have their own complexity, they are based on standards. The major opportunity for Health Application Integration is that health informatics is substantially standards based. The goal therefore is to develop a standards based set of web services that together with an integration orchestration allow applications to collaborate in an ecosystem based solely on the nature of the events being described, without having to be aware of the nature of the applications using those services.

The HCE offers the opportunity to start developing a “next generation” approach to application connection, allowing existing systems to participate in the ecosystem (whether through existing middleware tools or HCE) and over time allowing more flexible connection of both existing and new applications.

## HCE Description

HCE is a standards-based set of web services enabling health point of service applications to connect with other applications to support clinical collaborations delivering more efficient and knowledge based healthcare.

## Design Principles

The following are key overarching design principles for the HCE

* A Service Oriented Architecture approach has applied
* Consequently, the objective is for connected systems to be “Plug and Play” – provided they can supply or use data in schema compliant form through adapters.
* The adapters used internally are reference implementations of the structure required for connected system adapters.
* Messages represent clinical events not data items within individual point of service systems (known as service providers within an HCE solution)
* Translating messages at the edge of the solution - Semantic / data translation of messages should be where it is most easily handled – whether that is in the point of service system, or closer to the edge of the HCE within the adapter
* EHR information should as far as possible be federated, with pull-based messaging to assemble information where it is needed, when it is needed
* All messaging is synchronous, with those connected systems requiring asynchronous messaging being handled through a store and forward service provided by the adapter
* Service blocks should be self contained (in accordance with SOA principles) providing flexibility for physical deployment
* Connected systems (service providers) should not have to know the details of systems receiving or supplying data i.e. they should not have to map that data to the requirements of the other system, but rather abstract it to be consistent with standards based XML schemas appropriate for the particular clinical (and administrative) events being supported.
* Where an interactive session is needed (such as use of decision support tools within a clinical workflow) this will be undertaken by the originating connected system (source service provider) invoking the decision support system, not via the use of HCE messaging.
* Unless decided otherwise in a particular implementation, the clinical payloads should not be visible to the HCE i.e. they are encrypted / decrypted by the adapters and thus only visible within each service provider.
* HCE should allow implementing organizations to leverage existing, legacy applications and infrastructure investment. The use of adapters and the HCE provides translation from legacy applications to a Service Oriented Architecture based solution
* The overall design of the HCE should support connection of HCE with existing messaging infrastructures – e.g. HCE to HealthLink in the New Zealand context
* The introduction of a HCE-based solution should minimize disruption to existing clinical workflow. Where ever possible, information distributed via the HCE should be presented in a user’s existing application, without introducing yet another application for users to access information available within an HCE-based solution
* Specialized knowledge and logic within each point of service or connected system should be leveraged wherever possible – e.g. ordering of laboratory tests should be completed using the interface provided by Laboratory Information Systems (LIS) directly instead of replicating functionality within a practice management system
* The HCE should leverage the Microsoft technology stack throughout the solution from server products (e.g. BizTalk, SQL Server, and Active Directory) through to code (Patterns & Practices Application Blocks and Enterprise Library). This approach maximizes use of existing components, minimizes custom coding, allows solution to evolve in-line with Microsoft product roadmaps and reduces the technical risk by reusing widely used components

## What HCE is not

HCE is not a

* Clinical Data Repository (CDR) – although it could optionally support consolidation of clinical information into such a CDR (whether some form of consolidated information store or a partial information set to support, for example, “out of hours” emergency care when source systems may not be available for recent treatment history, current meds and allergies)
* Clinical Portal – this is assumed to be provided by an appropriate point of service system (even if that was only a viewing portal)
* Point of Service system

# Health Connection Engine Service Blocks

The functionality within the Health Connection Engine (HCE) is provided by a series of self contained, loosely coupled service blocks.

The components within each service block expose and consume Web Services. The service blocks provided by the HCE are illustrated in Figure 1.

Figure - HCE service blocks

The following table provides a high level description of each service block provided by the HCE:

|  |  |
| --- | --- |
| Service Block | Description |
| Message Management Services | A series of services associated with the processing of routing requests from Service Providers, Registers and Administration Services participating in a solution enabled by the HCE.  Services provided by this block include routing, logging and monitoring of Connection Engine messages. |
| Health Connection Engine (HCE) Registers | A series of discrete registers which store data needed to support the HCE configuration.  Primarily this service block consists of the Service Provider Register which holds Service provider, Pool, Message Type and Schema information. Each register provides access to its data store by accepting Connection Engine Messages routed to the register by the Message Management Services.  This block also contains other registers, such as the Invocation Register and Schema Repository which are required to ensure Service Provider communication within an HCE enabled solution.  The interface exposed by each register takes the form of a standard Adapter which accepts; processes and returns Connection Engine Messages. |
| Health Domain Registers | A series of discrete registers that contain Health Domain specific information, that typically in production implementations will be either supplied by third party / in existence systems or will need to be extended to meet the requirements of a particular implementation.  Health Domain Registers will include   * Patient Register – providing an authoritative source of Patient Identifier and basic demographic information within a HCE solution * Practitioner Register - providing an authoritative source of Practitioner Identifier and basic demographic information within a HCE solution * Consent Register – providing the ability to place role-based privacy constraints over the information available within a HCE solution * Event Register – providing an authoritative index of clinical event information which is available within the context of a HCE solution   Each register provides access to its data store by accepting Connection Engine Messages routed to the register by the Message Management Services.  The interface exposed by each register takes the form of a standard Adapter which accepts; processes and returns Connection Engine Messages.  Note: For version 2.1 of the HCE, a reference implementation of the Patient Register has been implemented. This reference implementation is intended to provide a sample application architecture and code implementation which can be used to construct production implementations required for a particular HCE solution. |
| Health Connection Engine (HCE) Administration Services | A set of data administration services which provide the ability to maintain data stored within each HCE Services Register.  The administration service components serve as a kind of “super adapter”, which translates requests from the HCE Administration Portal into Connection Engine Message routing requests. Each service component provides the business logic to complete this translation as well as the functionality associated with validation of the maintenance operations from both a content and security perspective. |
| Health Domain Administration Services | A set of data administration services which provide the ability to maintain data stored within each Health Domain Services Register.  The administration service components serve as a kind of “super adapter”, which translates requests from the Health Domain Administration Portal into Connection Engine Message routing requests. Each service component provides the business logic to complete this translation as well as the functionality associated with validation of the maintenance operations from both a content and security perspective.  Note: For version 2.1 of the HCE, limited reference implementations have been provided. It is expected that definition of the administration services required to support a production implementation will be based on requirements within each production environment. |
| Health Connection Engine (HCE) Administration Portal | A reference implementation of a browser-based user interface which provides user access to the web service interfaces exposed by the Health Service Directory Administration Services.  This portal, in association with the Health Domain Services Administration Services, provides the ability for administrators of the HCE to maintain the data held within the Health Domain Services Registers. |
| Health Domain Administration Portal | A reference implementation of a browser-based user interface which provides user access to the web service interfaces exposed by the HCE Services Administration Services.  This portal, in association with the Health Domain Services Administration Services, provides the ability for administrators of the HCE to maintain the data held within the HCE Services Registers.  Note: For version 2.1 of the HCE, a Health Domain Administration Portal has not been implemented. It is expected that definition of the administration portal required to support a production implementation will be based on the same architecture and design of the HCE Administration Portal and will be tailored based on the requirements within each production environment. |
| Infrastructure Services | Includes Security Envelope, Exception Management Logging and Auditing Services, and Change Management Services.  Security ensures that all Connection Engine Messages interaction between the HCE Services, Health Domain Services, Service Providers and Message Management Services are completed by identified and authorized entities.  This security is based on positive identification and authorization of Adapters, either those exposed within the HCE (by the HCE Services or Health Domain Services) or by the Connected System within a particular Service Providers.  Any exceptions that are raised during the processing of Connection Messages between systems and services via the HCE Routing Service, are handled and logged by the Adapters of those various systems and services  The Change Notification Service is part of the functionality provided by the Service Provider Register. The main goal of this service is to guarantee that changes within the Register that would affect the operation of an Adapter are notified to all affected Adapters within a HCE-enabled solution. This allows the Adapters to invalidate all affected cache data, forcing a reload during the next operation*.* |

Within the Service Provider Register, each Service Provider is configured within a Pool. Pools provide the ability to assign a Service Provider into one or more logical groups within an HCE solution.

Service Providers must exist within the same pool in order to successfully send or receive a Connection Engine Message to one another. The Routing Service validates that the source and destination Service Providers are in the same pool during the routing of a Connection Engine Message.

The use of Pools to separate Service Providers into groups allows a single physical instance of the HCE to support several logical implementations, with Service Providers being placed into the one or more Pools which represent the logical connection network within which they can communicate.

HCE provides the ability for Pools to be applied based on different factors (such as security, commercial relationships and/or organization boundaries) as the business rules and business process used to determine and configure Pools are applied outside the core functionality provided by the Service Provider Register.

In addition to the service blocks provided within the HCE, the HCE platform defines the way in which Connected Systems installed within a Service Provider should communicate with the HCE. Communication between the HCE and Service Providers is facilitated by the implementation of an Adapter which is developed in a tightly coupled fashion with each Connected System. Each Adapter exposes a standard web services interface which means all Connected Systems present a uniform interface to the HCE.

Figure - Service Provider and Health Connection Engine communication

# Connection Engine Message

All messages passed between Adapters within the HCE platform take the form of a Connection Engine Message.

The Connection Engine Message provides the common, XML-based document structure used for all messages where routing is coordinated by the Message Management Services. Connection Engine Messages are produced by and consumed by Adapters implemented by each Service Provider and by the internal services provided by the Registers and Administration Services.

## Message Structure

shows the components of the Connection Engine Message and the following table provides an overview of the structure and purpose of each message component.



Figure - Connection Engine Message structure

|  |  |
| --- | --- |
| Message Component | Description |
| Header | Each message contains a standard header. Items within the header identify parameters such as the   * + Unique Identifier allowing separate messages to be related   + Service Provider sending the message   + Intended recipient Service Provider for the message   + Type of the message   + Message Status code and description   The elements within the header are not encrypted and are available for interrogation and modification by the Message Management Services and Adapters during the routing process. |
| Body / Payload | The body of each message contains the payload associated with the message type. The payload conforms to the schema defined for the each message type held within the Service Provider Register.  The payload of each message is encrypted by the source Service Provider and can only be decrypted by the destination Service Provider. The body contains this encrypted payload along with the details needed to decrypt the payload, such as the type of encryption used. |

## Message Schema

Each Connection Engine Messages conforms to the XML Schema displayed in Appendix 1 – Connection Engine Message XML Schema. Further information about the definition and use of types within the schema can be found in .

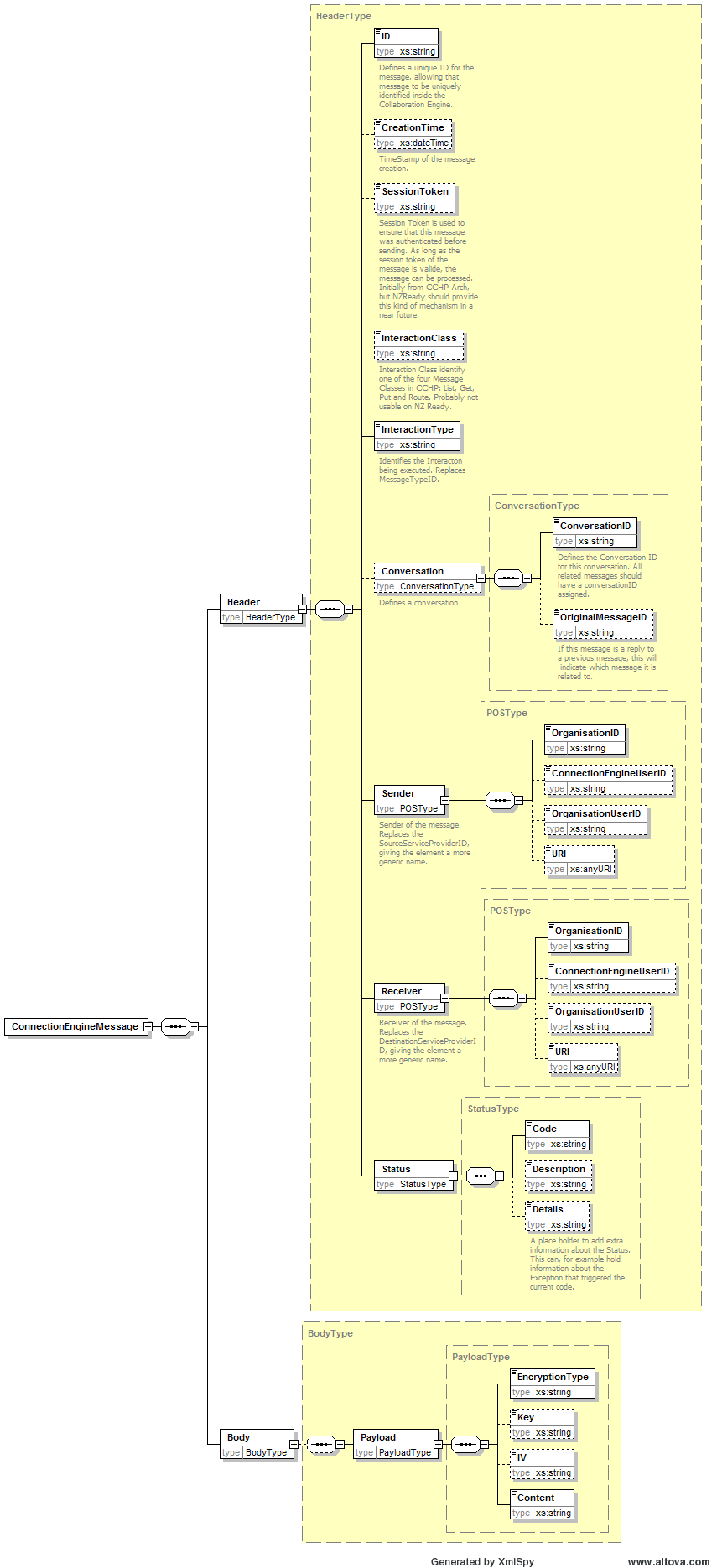


Figure - Connection Engine Message XML Schema

# Message Management Services

The Message Management Services provide a series of services associated with the processing of routing requests from Service Providers, HCE Services and Health Domain Services participating in a solution enabled by the HCE.

Services provided by this block include routing, logging and monitoring of all Connection Engine messages.



Figure - Message Management Services

The following table provides a high level description of each service block provided within the Message Management Services:

|  |  |
| --- | --- |
| Service Block | Description |
| Routing Service | Provides routing of Connection Engine Messages from the Adapter implemented by a source Service Provider to the Adapter implemented by a destination Service Provider.  Validation of each routing request is completed to ensure that source Service Provider is allowed to send the Connection Engine Message (defined by the Message Type) to the destination Service Provider. |
| Monitoring Service | Provides logging of all Connection Engine Messages submitted to the Message Management Services.  All elements within each Connection Engine Message header are logged and functionality is provided to view logged information for monitoring and auditing purposes. |

## Routing Service

| Purpose | Provides routing of Connection Engine Messages from the Adapter implemented by a source Service Provider to the Adapter implemented by a destination Service Provider.  Validation of each routing request is completed to ensure that source Service Provider is allowed to send the Connection Engine Message (defined by the Message Type) to the destination Service Provider. |
| --- | --- |
| Design Principles | This service has been designed and implemented to conform to the following principles:   * All interactions with the service must be loosely coupled with other services within the Health Connection Engine * Connection Engine Messages are passed in, modified within orchestrations and returned by the service * Components within the service should be implemented in a modular manner, allowing the functionality provided by the service to be extended with the minimal impact on other components, both within the service and within other service provided by the Health Connection Engine * All interaction should be assumed to be synchronous, end-to-end from source Service Provider to destination Service Provider * The Routing Service should only rely on access to the header information within the Connection Engine Message. The payload should be considered “opaque” to the service as it is encrypted with the destination Service Provider’s public key and can only be decrypted using the destination Service Provider’s private key |

### Implementation Overview

The Routing Service consists of 3 BizTalk Orchestrations and a Web Service interface, generated by BizTalk, which provide the ability:

* for the Adapter associated with the source Service Provider to submit a Connection Engine Message routing request and to receive feedback about the status of that request
* for the Adapter associated with the destination Service provider to receive a validated Connection Engine Message via its Adapter web service interface

The components within this service are represented in Figure 6 and described briefly in the following table:



Figure - Routing Service components

|  |  |  |
| --- | --- | --- |
| Service Component | Description | Implementation details |
| Web Service | Provides the web service exposed by the Routing Service which is called by a source Service Provider to submit a Connection Engine Message routing request | Implemented as an ASP.Net 2.0 Web Service.  Site name: ConnectionEngine  Web Service: ConnectionEngineAdapter.asmx  Web Method: ReceiveConnectionEngineMessage |
| Receive Message | Receives the Connection Engine Message from the Web Service and co-ordinates the processing of the routing request. Also provides feedback to the source Service Provider about the status of the routing request | Implemented as an Orchestration within BizTalk 2006  Orchestration name: ReceiveMessage |
| Validate Message | Called during the routing process to validate that the routing instructions within the Connection Engine Message header are valid, based on the Message Type, source Service Provider and destination Service Provider. | Implemented as an Orchestration within BizTalk 2006  Orchestration name: ValidateMessage |
| Process Message | Performs the duty of forwarding the Connection Engine Message to the destination Service Provider (if routing validation succeeds) | Implemented as an Orchestration within BizTalk 2006  Orchestration name: ProcessMessage |

### Message Flow

The following steps are completed whenever a Service Provider (the source Service Provider) wishes to send a Connection Engine Message to another Service Provider (the destination Service Provider). This message flow is illustrated in Figure 7.



Figure - Connection Engine Message routing request message flow

1. The Adapter in use at the source Service Provider submits a routing request to the Routing Service (see section 5 for further details on the process completed by an Adapter to submit a routing request). This request results in a Connection Engine Message being submitted to the ReceiveConnectionMessage web method exposed by the Routing Service Web Service component
2. The Routing Web Service in turn passes the Connection Engine Message to the Receive Port (port name: prtConnectionEngineInbound) of the Receive Message orchestration.
3. The Receive Message orchestration calls the Validate Message orchestration. This call passes the Connection Engine Message as an input parameter (parameter called: MessageToValidate) to the Validate Message orchestration
4. The Validate Message orchestration dynamically determines the location of the validation web service exposed by the Service Provider Register based on a configuration setting (appSetting name: ServiceProviderRegister.Validation.WebServiceURL) held in the BizTalk configuration file (filename: BTSNTSvc.exe.config) and constructs the message expected by the Service Provider Validation web service
5. The Validate Message orchestration calls the Service Provider Validation web service, by passing the Connection Engine Message to the Send Port (port name: prtConnectionEngineValidation).
6. The Service Provider Register validates the routing request based on parameters within the header of the Connection Engine Message. The validation consists of the following steps:
   1. The routing request is valid if the source and destination Service Providers are in the same Pool (as defined by rows within the ServiceProviderPool table within the Service Provider Register database
   2. The routing request is valid if the Message Type of the Connection Engine Message can be sent from the source Service Provider type to the destination Service Provider type (as defined by rows within the PoolServiceProviderType table within the Service Provider Register database
7. If the routing request is valid, then
   1. The Status Code and Description contained within the Connection Engine Message is set to the code associated with successful validation (e.g. the Header.Status.Code element is set to “0” and the Header.Status.Description element is set to “Ok”)
   2. The Service Provider Register also updates the location of the destination Service Provider Adapter web service contained within the Connection Engine Message (element: Header.Receiver.URI), based on the ID of the destination Service Provider (element: Header.Receiver.OrganisationID). This ensures that the Connection Engine Message will be routed to the correct destination Service Provider and means that the source Service Provider does not need to know the physical address of the destination Service Provider Adapter web service.
8. Once the result of the validation has been determined, the Service Provider Validation web service returns the Connection Engine Message with an updated status code and description to the Validate Message orchestration, via the Send Port (port name: prtConnectionEngineValidate) of the Validate Message orchestration.
9. The Validate Message orchestration in turn returns this Connection Engine Message as an output parameter (parameter called: ValidatedMessage) to the Receive Message orchestration.
10. The Receive Message orchestration determines if the validation was successful, based on the Status Code) element name: Header.Status.Code) contained within the Connection Engine Message.
11. If the validation was successful (Header.Status.Code = 0), then
    1. The Receive Message orchestration assigns a new Conversation ID (element name: Header.Conversation.ConversationID) to the Connection Engine Message if the Connection Engine Message is the first message in a conversation (e.g. if the Connection Engine Message didn’t previously contain a Conversation ID)
    2. The Receive Message orchestration then calls the Process Message orchestration. This call passes the Connection Engine Message as an input parameter (parameter called: MessageToProcess) to the Process Message orchestration
    3. The Process Message orchestration dynamically determines the location of the Adapter web service exposed by the destination Service Provider based on value of the Header.Receiver.URI element within the Connection Engine Message and constructs the message expected by the destination Service Provider Adapter web service
    4. The Process Message orchestration passes the Connection Engine Message to the Send Port (port name: prtConnectionEngineProcess) which results in a call to the ReceiveConnectionEngine web method exposed by the destination Service Provider Adapter web service
    5. The destination Service Provider processes the message (see section for further details of the processing completed by the destination Service Provider)
    6. Once the Connection Engine Message has been processed by the destination Service Provider, a response message (also a Connection Engine Message) is passed back to the Process Message orchestration, via the Send Port (port name: prtConnectionEngineProcess). Success or failure of the processing completed by the destination Service Provider is contained within the Header.Status.Code, Header.Status.Description and Header.Status.Details elements of the returned Connection Engine Message
    7. The Process Message orchestration in turn returns this Connection Engine Message as an output parameter (parameter called: ProcessResult) to the Receive Message orchestration.
12. If the validation was not successful (Status Code does not equal 0), then:
    1. The Receive Message orchestration constructs the Connection Engine Message to return to the source Service Provider. This message contains the Header.Status.Code, Header.Status.Description and Header.Status.Details elements returned by the Validate Message orchestration.
13. The Receive Message orchestration returns the Connection Message Engine message to the source Service Provider, via the Receive Port (port name: prtConnectionEngineInbound) and the Routing Service Web Service component
14. The source Service Provider determines the success or failure of the routing request, based on the status information contained within the returned Connection Engine Message (see section for further details of the processing completed by the source Service Provider).

### BizTalk Configuration

The following entries are required within the <appSettings> section of the BizTalk Server Configuration file (BTSNTSvc.exe.config) in order to support the Routing Service:

|  |  |
| --- | --- |
| Configuration Entries | Description |
| SystemAdapter.AuthenticationMethod | Identifies the type of authentication used by the destination Adapters. |
| ConnectionEngine.GeneralException.Code | Defines the message status code for unhandled errors that occurred within the Routing Service boundaries. |
| ConnectionEngine.GeneralException.Description | Defines the message status description for unhandled errors that occurred within the Routing Service boundaries. |
| ConnectionEngine.SystemAdapterException.Code | Define message status code for connection errors with the destination Adapter |
| ConnectionEngine.SystemAdapterException.Description | Define message status description for connection errors with the destination Adapter |
| ConnectionEngine.ClientCertificateThumbPrint | Defines the thumbprint of the client certificate that uniquely identifies the Routing Service for connection with destination Adapters through a secure channel. |
| ServiceProviderRegister.Validation.WebServiceURL | Defines the URL for the Service Provider Register’s Validation Web Service, used by the Routing Service. |

The following section provides an example of the <appSettings> section within the BizTalk Server configuration file:

<appSettings>

<add key="SystemAdapter.AuthenticationMethod" value="NTLM" />

<add key="ConnectionEngine.GeneralException.Code" value="99"/>

<add key="ConnectionEngine.GeneralException.Description"

value="An Unhandled exception occurred"/>

<add key="ConnectionEngine.SystemAdapterException.Code" value="18"/>

<add key="ConnectionEngine.SystemAdapterException.Description"

value="Error connecting to destination adapter"/>

<add key="ConnectionEngine.ClientCertificateThumbPrint" value="e8 33 06 57 9f 36 bc e6 6f fc 30 4d b1 eb 3d 5b 84 e5 74 88"/>

<add key="ServiceProviderRegister.Validation.WebServiceURL" value=<http://hce21dev.simpl.co.nz/ServiceProviderRegisterWS/ConnectionEngineValidation.asmx>”/>

</appSettings>

# Adapters

## Purpose and Design Principles

The Adapters provide a bridge between a Connected System and the Health Connection Engine, providing message translation, encryption and submission services.

This service has been designed and implemented to conform to the following principles:

* Create a framework that provides the minimum amount of implementation effort possible in order to have a new Adapter up and running.
* Isolate activities that require tight coupling with the associated Connected System from the common activities which must be performed by all Adapters. This allows the addition of new message types to the Service Provider without any major refactoring of the common activities.

## Overview

Adapters provide a bridge between Connected Systems and the Health Connection Engine by providing two basic services:

* Generating Health Connection Engine Messages based on data sent from the associated Connected system
* Extracting data from Health Connection Engine messages to allow process by the associated Connected System.

In order to achieve this goal, they perform a series of activities as displayed in Figure 8.



Figure - Adapter Activity Flow

The activities in Figure 8 can be divided into two main categories:

* Activities which are particular to each Adapter. Those activities are tightly coupled to the Connected System, and should be implemented for each new Adapter.
* Common activities to all Adapters (validation, encryption/decryption, message submission/reception). Those activities can be implemented within a base framework and be used by the implementation of each Adapter.

Activities that are particular to each Adapter can be implemented in a way that doesn’t impact the existing functionality of an Adapter. Additional activities are usually message type based, meaning that the addition of new message types to the pool of messages a Service Provider can accept/generate can be implemented without any major refactoring within the Adapter code.

## Implementing an Adapter

### Adapter Base Class

The Adapter Base class defines a foundation class for the creation of Adapters, implementing all the common activities, as well as implementing the activity flows for both sending and receiving messages.

The class also defines virtual or abstract methods that implement the interface with the associated Connected System and the acquisition of reference data. One of these methods, ProcessMessage, defines a pattern for a scalable interaction with Connected Systems through the use of reflection and delegates, allowing new message types to be dynamically added to the pool of messages processed by the Adapter without the need to recompile the Adapter. The diagram in Figure 9 represents the class structure of the foundation class, called Microsoft.ConnectionEngine.Adapters.SystemAdapterBase[[1]](#footnote-2):

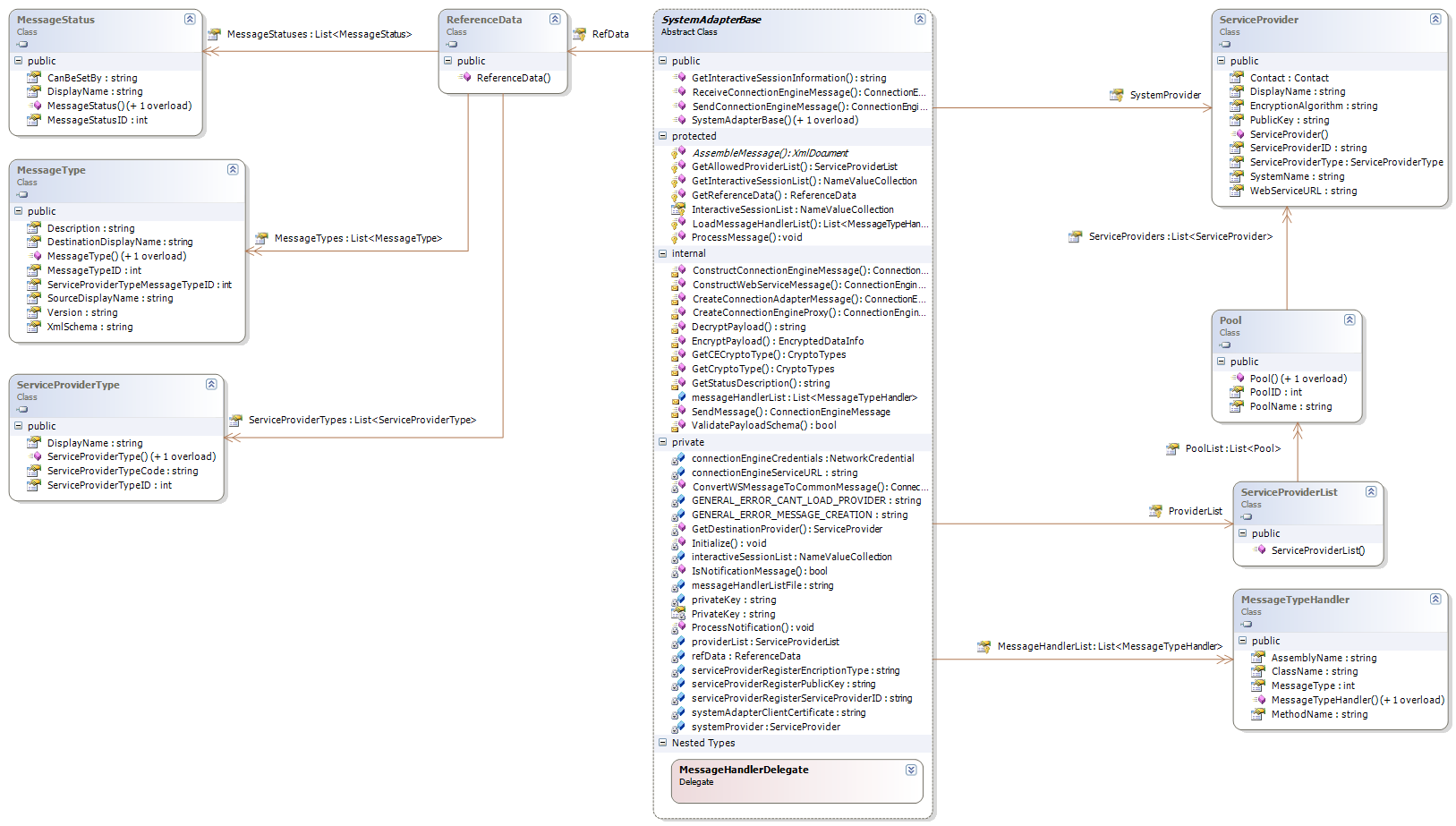


Figure - Adapter Base Class Structure

The following table provides a description of each entity within the Adapter Base class.

|  |  |
| --- | --- |
| Class | Description |
| SystemAdapterBase | Defines the foundation class for Adapters. |
| MessageTypeHandler | Message Type Handler is a data object class used in conjunction with Message Handler delegate to define a pattern for generically handling the processing associated with each message type. |
| ServiceProvider | Service provider is a data object class that represents a Service Provider entity. |
| Pool | Pool is a data object class that represents a logical grouping of Service Providers. |
| ServiceProviderList | Service Provider list is a data object class that represents a collection of pools, which in turn represent collections of Service Providers. |
| ReferenceData | Reference Data is data object class that groups collections of HCE related metadata, including message statuses, message types and Service Provider types. |
| MessageStatus | Message Status is a data object class that represents a status indicating success or reason for failure within a Health Connection Engine Message. |
| MessageType | Message Type is a data object class that represents information related to message type, including the message type id, description and xml schema location. This schema location is used during the validation process. |
| ServiceProviderType | Service Provider Type is a data object class that represents a logical grouping of Service Providers that have the same functionality, including message type that each type of Service Provider should be able to send and receive. |

### Adapter Classes

Adapter classes should be implemented for each Connected System in use within a Service Provider, as each implementation of an Adapter should manage the block of activities that is particular for each adapter, leveraging from the common activities already implemented in Adapter Base. This implies that each Adapter should implement the AssembleMessage abstract class, and the ProcessMessage method, if the Adapter needs to handle incoming message processing in a different way than the default implementation of ProcessMessage provided within the Adapter Base class.

In order to leverage from the common functionality already existing in Adapter base, the new Adapter classes, should be inherited from the Adapter Base class and then implement or overload any method as required.

The class diagram in Figure 10 presents an example of an Adapter class (SystemAdaptersSample) that is inherited from Adapter Base.

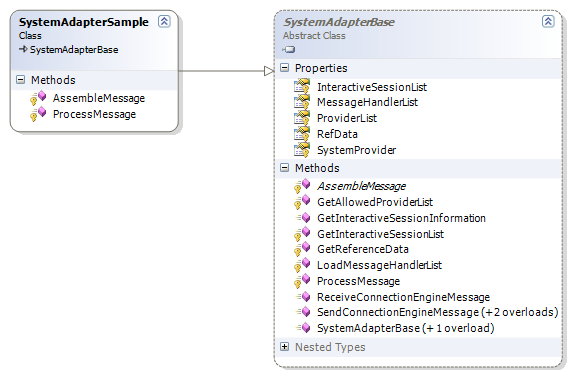


Figure – Example Adapter class inherited from the Adapter Base class

As shown in the class diagram in Figure 10, the Adapter Base class exposes a series of methods that can be accessed externally or within the inherited Adapter. There are basically two types of accessible methods: Public Methods and Protected Methods.

#### Public Methods

Public methods are used by either the Connected System or the Health Connection Engine to interact with the Adapter. The public methods are:

|  |  |
| --- | --- |
| Method | Description |
| SystemAdapterBase | The SystemAdapterBase constructor |
| SendConnectionEngineMessage | Sends a Connection Engine Message to another Service Provider by submitting a routing request to the Message Management Services within the HCE. |
| ReceiveConnectionEngineMessage | Receives a Connection Engine Message from another Service Provider, as routed by the Message Management Services within the HCE. |
| GetInteractiveSessionInformation | Gets the URL for an HTTP Interactive Session with a Destination Service Provider, based on a messagetype and a list of arguments. |

Section 5.4 presents the message flow for each of these public methods.

#### Protected Methods

Protected methods are methods exposed only to the Adapter class being implemented. These are either abstract methods that should be implemented, like Assemble Message, or virtual methods that can be overloaded if the ISV or SI implementing an Adapter wants more control over certain aspects of the Adapter.

|  |  |  |
| --- | --- | --- |
| Method | Inheritance Modifier | Description |
| AssembleMessage | Abstract | Creates a new Connection Engine Message compatible Payload based on data supplied by a Connected System and a Message Type. |
| GetAllowedProviderList | Virtual | Load a list of providers this Adapter is able to contact.  The implementation provided by the Adapter Base retrieves the list of providers through a message exchange with the Service Provider Register, but this approach can be changed by overloading this method and providing a different implementation. |
| GetReferenceData | Virtual | Gathers the common data needed by the Adapter to perform its activities.  The implementation provided by the Adapter Base retrieves the list of providers through a message exchange with the Service Provider Register, but this approach can be changed by overloading this method and providing a different implementation. |
| LoadMessageHandlerList | Virtual | Loads a list of Adapter Message Handlers, based on a xml document defined within the Adapter’s configuration file |
| ProcessMessage | Virtual | Process a Connection Engine Message compatible Payload based on its Message Type. |
| GetInteractiveSessionList | Virtual | Gets a list of Interactive Session URI the Adapter is allowed to generate. |

## Adapter Web Service

The Adapter Web Service provides a loosely coupled communication interface between the Adapter and both the Connected System and HCE.

From the Connected System point of view, it guarantees that the Adapter can be easily updated or replaced, as long as the contract interface remains the same.

From the HCE point of view it provides an interface that allows the Routing Service orchestrations to dynamically point to an Adapter by just setting up the correct web service URL. It also guarantees that each Adapter can have only one end point where HCE internal services, like the Change Notification, need to access in order to publish or update information.

As part of the Adapter contract with the Connected System and HCE, the following rules should be observed when implementing Adapter Web Services:

* This web service should have [**http://Microsoft.ConnectionEngine.Services**](http://Microsoft.CollaborationEngine.Services) as its XML Namespace;
* Each web service implementation must expose a set of web methods that proxy their public methods:
  + SendConnectionEngineMessage
  + ReceiveConnectionEngineMessage
  + GetInteractiveSessionURL

The following snippet shows the source code of a typical Adapter Web Service:

using System;

using System.Web;

using System.Collections;

using System.Web.Services;

using System.Web.Services.Protocols;

using Microsoft.ConnectionEngine.Adapters.Samples;

using Microsoft.ConnectionEngine.Common;

[WebService(Namespace = "http://Microsoft.ConnectionEngine.Services")]

public class ConnectionEngineAdapter : System.Web.Services.WebService

{

static private ServiceProviderSampleSystemAdapter systemAdapter;

public ConnectionEngineAdapter()

{

if (systemAdapter == null)

systemAdapter = new ServiceProviderSampleSystemAdapter();

}

[WebMethod]

public ConnectionEngineMessage ReceiveConnectionEngineMessage(ConnectionEngineMessage connectionEngineMessageRequest)

{

ConnectionEngineMessage result = systemAdapter.ReceiveConnectionEngineMessage(connectionEngineMessageRequest);

return result;

}

[WebMethod]

public ConnectionEngineMessage SendConnectionEngineMessage(object objmessage, int messagetype, Guid conversationid, string destinationid)

{

ConnectionEngineMessage result = systemAdapter.SendConnectionEngineMessage(objmessage, messagetype, conversationid, destinationid);

return result;

}

[WebMethod]

public string GetInteractiveSessionInformation(string serviceproviderid, int messagetypeid, System.Collections.Generic.List<System.Collections.Generic.KeyValuePair<string,string>> args)

{

string result = systemAdapter.GetInteractiveSessionInformation(serviceproviderid, messagetypeid, args);

return result;

}

}

## Adapter Message Flow

The message flow within the Adapter is centered on the activities implemented by its public methods. There are exchanges of messages with HCE during the Adapter Instantiation, through the communication with Service Provider Register, and later on during the sending or receiving of Connection Engine Messages routed by the Message Management Services.

### SystemAdapterBase – Instantiating an Adapter

The following steps are completed when an Adapter is being instantiated:

1. Populate information about Service Provider based on configuration parameters within the Adapter configuration file.
2. Populate Private Key Information based on configuration parameters within the Adapter configuration file.
3. Get Allowed Provider List Information from the Service Provider Register, through the use of the GetAllowedProviderList protected method.
4. Get Reference Data Information from the Service Provider Register, through the use of the GetReferenceDate protected method.

#### Reference Data Caching:

The cache of data needed by the Adapter is populated within the Adapter Constructor. The data stored during this step includes:

* **ProviderList** – A list of providers this Adapter is allowed to contact;
* **RefData** – A collection of metadata used to help the Adapter to perform its activities. The collection includes Message Status List, and Message Type List

The only other piece of data that is cached within the Adapter is the **MessageHandlerList**, that represents the list of methods within the Adapter that should handle each one of the message types the Service Provider should be able to process. MessageHandlerList is loaded the first time the property is used.

Each one of these data collections is stored within the Adapter and loaded on instantiation and after a notification message has been received from the HCE. The HCE notification message clears down the required cache of data to facilitate the reload of reference data.

### SendConnectionEngineMessage – Sending a Message

The following steps are completed when a Connection Engine Message is submitted by an Adapter to the HCE:

1. Translate the native content provided by the associated Connected System into a Connection Engine Message Payload that conforms to a specific message type, using the AssembleMessage method.
2. Validate the generated payload by completing schema validation based on Message Type, using the ValidatePayloadSchema internal method.
3. Encrypt the generated Payload based on the public key of the destination Service Provider, using the EncryptPayload internal method.
4. Submit the routing request by calling ReceiveConnectionEngine web method exposed by HCE Routing Service.
5. Receive the result message back from the HCE Routing Service. This result message is a full Connection Engine Message
6. Assess if the result message is an acknowledgement or a new message
   1. If the result is a new message, decrypt the payload and assign it back to Result Message. This is performed by the DecryptPayload internal method.
   2. If acknowledgement, assume payload hasn’t changed
7. Return Result Message, which is a Connection Engine Message with a decrypted payload. This result message also holds the message status which communicates an error condition to the associated Connected System

### ReceiveConnectionEngineMessage – Receiving a Message

The following steps are completed when a Connection Engine Message is received by an Adapter (e.g. when a Connection Engine Message is sent to the Adapter from the HCE Routing Service):

1. Receive Connection Engine Message from HCE in a call to ReceiveConnectionEngineMessage, which is exposed by the Adapter as the web method ReceiveConnectionEngineMessage in the web service ConnectionEngineAdapter.asmx
2. Decrypt the payload using the Service Provider’s own private key. This is performed by the DecryptPayload internal method.
3. Validate Payload against the schema for message type, using the ValidatePayloadSchema internal method.
4. Access if the received Connection Engine Message is a notification message
   1. If it is a notification message, empty the Reference Data Cache. This is done by the ProcessNotificaton method, which determines the changed element of the payload and clears down the required cache of data. ProcessNotification message returns a new payload and the related message type
   2. Otherwise, hand over the validated payload to the associated Connected System for processing. This is done by the ProcessMessage method, which uses delegates and reflection to identify the right method to be called based on the message type of the message received. ProcessMessage returns a new payload and the related message type
5. Assess if the processed message is an acknowledgement or new message
6. If the processed message is a new message
   1. Validate the processed message payload
   2. Encrypt Payload using public key of source service provider
   3. Assign the encrypted payload to processed message
7. If the processed message is an acknowledgement, just go to step 8
8. Return the processed message to the HCE Routing Service. The HCE in turn passes the message back to the source Service Provider as the result message

#### ProcessMessage and MessageHandlerDelegate

In order to have the Adapter loosely coupled from the Connected System when receiving messages, the default implementation of ProcessMessage method uses reflection and delegation to dynamically communicate with the Connected System’s Business Processes. To achieve this goal, the following steps must be executed:

1. The Connected System must create a set of static proxy methods that implement the MessageHandlerDelegate Signature. Ideally, each message type processed by this Connected System will have its own associated proxy method.
2. The MessageHandlerList.xml file should be updated to store the following information for each one of the message types processed:
   1. MessageType – the id for the message type being handled
   2. AssemblyName – the assembly that must be loaded in order to get the proxy method.
   3. ClassName – the class within the assembly where the proxy method should be found
   4. MethodName – the proxy method name to be dynamically executed in order to process the Message Type.

If the ISV or SI implementing an Adapter wants more control over how the messages are processed or how the Message Handlers are loaded, both ProcessMessage and LoadMessageHandlerList methods are virtual and can be easily overloaded.

Example of the MessageHandler section within the MessageHandler.xml file:

<MessageTypeHandler>

<MessageType>1000</MessageType>

<AssemblyName>Microsoft.ConnectionEngine.Registers.Core.

ServiceProvider.Facade.dll

</AssemblyName>

<ClassName>Microsoft.ConnectionEngine.Registers.Core.

ServiceProvider.Facade.ServiceProviderManager

</ClassName>

<MethodName>GetAllowedProviders</MethodName>

</MessageTypeHandler>

The source code for the ProcessMethod is provided in section 0 and could be used to define a similar approach for the AssembleMessage abstract method which is used to assemble message payloads prior to them being sent to the HCE.

### GetInteractiveSessionInformation -- Generating an Interactive Session URL

The following steps are completed to create an interactive session compatible message, when the GetInteractiveSessionInformation is invoked:

1. Validate the received parameters by generating an XML document
   1. Service Provider Identification – unique identifier for a Service Provider
   2. Message Type Identification - the message type used to generate the arguments of the interactive session
   3. Arguments List – a collection of key value pairs needed to generate the interactive session

If validation is successful

1. Get the Destination Provider information based upon the Service Provider Identification
2. Assemble a new URL based on the Service Provider Identification, Message Type Identification and Arguments List
3. Return the assembled URL

If validation is not successful

1. Raise a Creating Iteractive Session Payload exception

## Configuration Parameters

The following is a list of configuration parameters used during the initialization of an Adapter and during the Sending and Receiving Messages activities. The configuration parameters add a layer of maintainability, allowing environmental variables to change without the need to recompile the Adapter code.

| ****Configuration Entries**** | ****Description**** |
| --- | --- |
| ConnectionEngine.WebServiceURL | The URI of the web service exposed by the HCE Routing Service. |
| ConnectionEngine.UserName | Credentials used to connect to the Routing Service within the HCE (optional – these credentials can substitute Client certification as authentication method in deployments that don’t require SSL tunnelling). |
| ConnectionEngine.Password |
| ServiceProviderRegister.ServiceProviderID | The unique identifier (Service Provider ID) of the Service Provider Register. |
| ServiceProviderRegister.PublicKey | The public key used to encrypt payloads sent to the Service Provider Register during initialization. |
| ServiceProviderRegister.EncryptionType | The encryption type used by the Service Provider Register. |
| SystemAdapter.ServiceProviderID | Unique Identifier of the Service Provider associated within this Adapter |
| SystemAdapter.WebServiceURL | A URI identifying the Adapter’s Connection Engine Adapter Web Service |
| SystemAdapter.EncryptionType | The Encryption Type used by this Adapter. This value should match the value recorded for the Service Provider within the Service Provider Register. |
| SystemAdapter.PublicKey | The public key used by this Service Provider. |
| SystemAdapter.PrivateKey | The private key used by this Service Provider. |
| SystemAdapter.MessageHandlerList | The location of MessageHanderList Xml file. |
| SystemAdapter.ClientCertificateName | This Adapter’s Client Certificate. Required for systems that use SSL as the authentication method. |

Example Adapter settings:

<appSettings>

     <add key="ConnectionEngine.WebServiceURL"

value="http://hcebiztalk.simpl.co.nz/CollaborationEngine/

CollaborationEngineAdapter.asmx"/>

     <add key="ConnectionEngine.UserName" value="CS1"/>

     <add key="ConnectionEngine.Password" value="password1"/>

     <add key="ServiceProviderRegister.ServiceProviderID" value="101"/>

     <add key="ServiceProviderRegister.PublicKey" value="C:\HSD Keys\PublicKey.xml"/>

     <add key="ServiceProviderRegister.EncryptionType" value="TripleDES"/>

     <add key="SystemAdapter.ServiceProviderID" value="107"/>

     <add key="SystemAdapter.EncryptionType" value="TripleDES"/>

     <add key="SystemAdapter.PublicKey" value="C:\HSD Keys\PublicKey.xml"/>

     <add key="SystemAdapter.PrivateKey" value="C:\HSD Keys\PrivateKey.xml"/>

     <add key="SystemAdapter.ClientCertificateName" value=""/>

</appSettings>

## Exception Handling

For all Message Flows, a failure in any of the steps generates a SystemAdapterException that will translate into a Message Status.

Each of the values in the following table represents a SystemAdapterExceptionCause enumeration value. These values map to MessageStatus in the Service Provider Register metadata and represent the final status of a message. Any status value different from 0 (zero) indicates a failure:

|  |  |  |
| --- | --- | --- |
| Exception | Code | Description |
| SourceAndDestinationNotInTheSamePool | 3 | Raised when a Service Provider is not within the list of Allowed Providers based on the provider ID |
| PayloadFormatNotValid | 5 | Raised when the format of the message payload is invalid according to its message type |
| PayloadContentNotValid | 6 | Raised when an error occurred processing the payload |
| PayloadDecryption | 7 | Raised when a problem occurred during decryption of the payload |
| PayloadEncryption | 8 | Raised when a problem occurred during encryption of the payload |
| Hl7PlayloadCreation | 9 | Raised when an error occurred when creating an iteractive session Payload |
| CreatingConnectionEngineMessage | 11 | Raised when a problem occurred creating a Connection Engine message |
| Unknown | 99 | Raised when an unknown error occurred |

# Registers and Associated Services

## Overview

****HCE Registers, with the associated administration services and portal to exercise those services, provide the means to manage the core HCE services. The Health Domain Registers and administration services conceptually provide the same functions for areas such as patient, practitioner, and consents. Potential approaches to those are set out elsewhere in this document.

Figure : Registers and associated services

## HCE Registers

Each HCE Register is a separate self-contained Connected System that interacts with other Connected Systems by means of messages managed by the HCE Routing Service.

The HCE regards each Register as a Service Provider, and deals with it in the same way as it deals with all other Service Providers including external clinical Point of Service systems.

Registers exhibit the following characteristics that are not typical of Service Providers in general:

1. Each instance of the HCE expects to be associated with exactly one instance of each type of Register.
2. The HCE, and Registers associated with it, are mutually interdependent, in that they all expect each Register to fulfill a standard service contract defined for its Register type. Except for this requirement, different implementations of HCE may be associated with different implementations of each Register.
3. Each Register publishes identifiers for each instance of each entity that it persists (service provider, patient, practitioner, etc). Each identifier may be relied on to unambiguously reference the entity instance for all time, although, over time, multiple identifiers may be published for the same entity instance.
4. Registers always respond synchronously to incoming messages via a single web service interface.
5. Registers do not send messages to external Service Providers although, in preparing a response to an incoming message, they may send a message to another internal Register (associated with the same instance of HCE), whereupon they will expect a synchronous response.

The HCE Registers are reference implementations of Registers in general. Their designs all follow a standard software design pattern that includes the following component layers:

**Adapter**

This exposes a single web service that accepts standard HCE messages containing a header and an encrypted payload.

The Adapter is configured to support a range of message types by a configuration file that defines the message types, and for each message type specifies the method exposed by the Façade layer (see below) to handle that message type.

The façade methods all accept and return XmlDocument objects that contain a standard set of data including the encrypted message payload.

This architecture allows the Adapter code to be highly standardized, but to offer full flexibility in supporting an extensible set of message types.

**Façade**

As described above, this interfaces with the Adapter using XmlDocument objects and methods tailored to each message that can be accepted from the HCE.

The primary task of the façade is to invoke the appropriate set of business layer functions to process each message. The façade exchanges data with the business layer functions using instances of business objects that are defined in the Common layer (see below). Data conversion between XmlDocuments and business objects is carried out within the façade.

**Business**

All of the business logic that defines how messages are interpreted and handled is embodied in the business layer. This logic operates on instances of business objects that are exchanged with the façade layer (above) and instances of typed datasets that are exchanged with the data access layer (below).

**Data Access**

The Data Access layer is responsible for all database access at the request of the business layer. It reads from and writes to the database using native SQL Server data access methods, filling or reading from typed datasets that are defined in the Common layer.

**Common**

The Common layer defines all business object classes and all typed data sets that are used to exchange data among the Data Access, Business and Façade layers. Some common static utility functions are also contained in the Common layer.

## Service Provider Register

This has 2 web services – standard Adapter Web Service (used to process requests from other Service Providers for information contained within the register) and Validation Web Service (used exclusively by the Validate Message component within the Routing Service to validate the parameters of each routing request)



|  |  |  |
| --- | --- | --- |
| Web Service | ASMX file | Web Method |
| Adapter Web Service | ConnectionEngineAdapter.asmx | Name: ReceiveConnectionMessage  Input Parameters: ConnectionEngineMessage  Output Parameters:  ConnectionEngineMessage |
| Validation Web Service | ConnectionEngineValidation.asmx | Name: ValidateConnectionEngine  Input Parameters: ConnectionEngineMessage  Output Parameters:  ConnectionEngineMessage |

#### Supported Message Types

Pattern to Message Types supported:

* Information Requests from Adapters
* Maintenance requests from HCE Administration Services
  + Entities
    - Message Type Manager
    - Pool Manager
    - Pool Service Provider Type Manager
    - Service Provider Manager
    - Service Provider Pool Manager
    - Service Provider Type Manager
    - Service Provider Type Message Type Manager
  + For each Entity
    - Search Request, Search Response
    - Select Request, Select Response
    - Lock Request. Lock Result
    - Unlock, Unlock All
    - Insert Request, Insert Response
    - Update Request, Update Response
    - Delete

General messages send from Service Providers:

|  |  |
| --- | --- |
| Message Type ID | Message Type Name |
| 1000 | Service Provider Search Request |
| 1001 | Service Provider Search Result |
| 1002 | Health Service Directory Reference Data Request |
| 1003 | Health Service Directory Reference Data Result |
| 1004 | Healthcare User Identifier Request |
| 1005 | Healthcare User Identifier Result |
| 1201 | Invocation Registry Item |
| 1200 | Invocation Registry Item Request |

Message types associated with authentication of users. Used by the HCE Administration Service:

|  |  |
| --- | --- |
| Message Type ID | Message Type Name |
| 2000 | Validate User Request |
| 2001 | Validate User Response |

Message types associated with maintenance of Service Provider. Used by HCE Administration Service:

|  |  |
| --- | --- |
| Message Type ID | Message Type Name |
| 1014 | Service Provider Search Request |
| 1015 | Service Provider Search Response |
| 1070 | ServiceProvider Select Request |
| 1071 | ServiceProvider Select Result |
| 1008 | Service Provider Lock |
| 1013 | Service Provider Check- Out Response |
| 1007 | Service Provider UnLock |
| 1040 | ServiceProvider Unlock All |
| 1010 | Service Provider Insert |
| 1011 | Service Provider Insert Response |
| 1009 | Service Provider Update |
| 1012 | Service Provider Update Result |
| 1006 | Service Provider Delete |

Message types associated with maintenance of Pools. Used by HCE Administration Service:

|  |  |
| --- | --- |
| Message Type ID | Message Type Code |
| 1024 | Pool Search Request |
| 1025 | Pool Search Response |
| 1066 | Pool SelectItem Request |
| 1067 | Pool SelectItem Result |
| 1018 | Pool Check-Out |
| 1023 | Pool Check-Out Response |
| 1017 | Pool Check-In |
| 1038 | Pool Check-In All |
| 1020 | Pool Insert |
| 1021 | Pool Insert Response |
| 1016 | Pool Delete |
| 1019 | Pool Update |
| 1022 | Pool Update Response |
| 1026 | Remove ServiceProvider From Pool |
| 1027 | Add ServiceProvider To Pool |
| 9003 | ServiceProviderPool Delete List Request |
| 9004 | ServiceProviderPool Insert List Request |
| 1028 | Remover ServiceProviderType From Pool |
| 1029 | Add ServiceProviderType To Pool |
| 1063 | PoolServiceProviderTypes Insert List Request |
| 1061 | Pool ServiceProviderType Search Request |
| 1062 | Pool ServiceProviderType Search Result |
| 1068 | Pool ServiceProviderType Delete List |

Message types associated with maintenance of Service Provider Types. Used by HCE Administration Service:

|  |  |
| --- | --- |
| Message Type ID | Message Type Code |
| 1041 | ServiceProviderType Search Request |
| 1042 | ServiceProviderType Search Response |
| 1064 | ServiceProviderType SelectItem Request |
| 1065 | ServiceProviderType SelectItem Result |
| 1032 | ServiceProviderType Check-Out |
| 1037 | ServiceProviderType Check-Out Response |
| 1031 | ServiceProviderType Check-In |
| 1039 | ServiceProviderType Check-In All |
| 1034 | ServiceProviderType Insert |
| 1035 | ServiceProviderType Insert Response |
| 1033 | ServiceProviderType Update |
| 1036 | ServiceProviderType Update Response |
| 1030 | ServiceProviderType Delete |
| 1054 | Add Service Provider Types to MessageType |
| 1055 | Remove Service Provider Types from Message Type |

Message types associated with maintenance of Service Provider Types. Used by HCE Administration Service:

|  |  |
| --- | --- |
| Message Type ID | Message Type Code |
| 1043 | MessageType Search Request |
| 1044 | MessageType Search Response |
| 1056 | MessageType SelectItem Request |
| 1057 | MessageType SelectItem Result |
| 1045 | MessageType Check-In |
| 1046 | MessageType Check-Out |
| 1047 | MessageType Check-Out Response |
| 1048 | MessageType Check-In All |
| 1049 | MessageType Insert |
| 1050 | MessageType Insert Result |
| 1051 | MessageType Update |
| 1052 | MessageType Update Result |
| 1053 | MessageType Delete |
| 1058 | ServiceProviderType MessageType Insert List |
| 1059 | ServiceProviderTypeMessageType Search Request |
| 1060 | ServiceProviderTypeMessageType Search Result |
| 1069 | ServiceProviderType MessageType Delete List |

#### Data Model



|  |  |
| --- | --- |
| Entity | Description |
| ServiceProvider | Service provider is a data object class that represents a Service Provider entity |
| ServiceProviderType | Service Provider Type is a data object class that represents a logical grouping of Service Providers that have the same functionality, including message type that each type of Service Provider should be able to send and receive. |
| MessageType | Message Type is a data object class that represents information related to message type, including the message type id, description and xml schema location. This schema location is used during the validation process. |
| Pool | Pool is a data object class that represents the logical connection network within which they can communicate. |
| ServiceProviderPool | Service Provider Pool is a data object class that represents a logical grouping of Service Providers that form a communicating network... |

## Invocation Register

### Purpose

The Invocation Register stores and provides the information needed to enable Point of Service (PoS) systems to invoke services of other PoS systems directly, rather than via the HCE.

The purpose of this is that not all interactions are most efficiently handled by messages, as some require an interactive session to be most effective. As such, the service needs to invoke the other PoS system, passing it appropriate clinical context and authorization credentials.

Services of such type that operate outside of the messaging services are termed “out of band” services.



### Design Principles

This register has been designed and implemented to conform to the following principles:

* An Invocation Register administration service (similar to those for the Patient Register and the Consent Register) allows a Point of Service system to publish information about its out-of-band services in the Invocation Register.
* “List” capability allows a PoS system to request the Invocation Register to enumerate available out-of-band services.
* “Get” capability allows a PoS system to request the Invocation Register to provide details of a nominated out-of-band service, including:
  + The service type (http, https, SMS, FTP, etc)
  + The service location (e.g. URL )
  + The definition of any required or optional parameters to be provided by the invoking system. These would normally be in the form of name-value pairs.

### Workflow

From the perspective of the HCE Routing Service, the Invocation Register behaves like any other connected system, being the destination ServiceProvider for List and Get messages from other connected systems.

A typical workflow would be as follows:

* PoS system B uses Invocation Register Administration Services to record the out-of-band services it provides, and the invocation details for each.
* PoS system A access the information in the Invocation Register using standard Connected messages sent to the Invocation Register via the HCE Routing Service.
* PoS system A uses this information to access the required service provided by PoS system B directly.

# Infrastructure Services

These include Security Envelope, Exception Management Logging and Auditing Services, and Change Notification Services.

Because any actual production versions of these services will depend on the environment they are to be implemented in, these services should be considered demonstration ready rather than production ready at this point.

## Security Envelope

Security ensures that all Connection Engine Message interaction between the HCE Services, Health Domain Services, Service Providers and Message Management Services are completed by identified and authorized entities.

This security is based on positive identification and authorization of System Adapters, either those exposed within the HCE (by the HCE Services or Health Domain Services) or by the Collaboration System within a particular Service Providers.

## Exception Management, Logging and Auditing Services

### Purpose

These services provide a standardized approach to managing system exceptions that occur within the HCE and its services, registers and Adapters.

### Design Principles

This service has been designed and implemented to conform to the following principles:

**Adapters**

Any exceptions that are raised during the processing of Connected Messages between systems and services via the HCE Routing Service are handled and logged by the Adapters of those various systems and services.

**Incoming Messages**

* Adapters always handle incoming messages synchronously, with both the Web Method call argument and the return value taking the form of a standard Connected Message (header + payload).
* When the processing of an incoming message is completed successfully, the return message will be of a different type, appropriate to the nature of the incoming message. (E.g. incoming query messages, successfully processed, may give rise to a returned query result message.)
* When the processing of an incoming message causes an exception to be raised, the returned message will be of the same type as the incoming message, and will carry the same payload. However the header of the returned message will specify the type of exception that occurred, expressed as an “Adapter Exception” type.
* A system exception raised during message processing is trapped within the Adapter at a point where the processing step can be most sensibly identified. At this point the general exception is translated into the appropriate Adapter exception, logged, and used to assemble the return message for the Web Method.

**Outgoing Messages**

* Adapters always provide some response back to the Point of Service system as a result of a message request.
* If an outgoing message is transmitted to the destination PoS Adapter via the HCE Routing Service and an exception is raised by the destination adapter as described above under “Incoming Messages”, the returned message will carry the error status indicator. The source Adapter will deal with the returned error status message close to its interface with its PoS system, or indeed may pass it back to be handled by the PoS system itself.
* In an analogous way, an exception raised during outgoing message processing is trapped at a point where the processing step can be most sensibly identified, is translated into the appropriate Adapter exception, is logged, and is passed back to be handled close to, or within, the PoS system. In such cases the message never reaches the HCE or the destination system, but nevertheless the exception is recognized, logged and handled in much the same way as if it had.

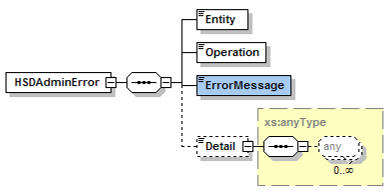
**HCE Administration Services**

As described earlier in this document, the HCE Administration Services service block is co-hosted with a Point of Service (PoS) system to facilitate its access to Registers.

It exposes multiple Web Service methods that do not use standard Connected messages for arguments and return values, so the exception handling and logging approach described above for Service Adapters is not applicable.

Instead, the following approach is used:

* An exception raised during processing within an Administration Services block, or an error status messages returned from a Register Adapter, is trapped at a point where the processing step can be most sensibly identified, is translated into the appropriate “HCE Admin Services” exception, is logged, and is passed back to the user interface (aspx) that has invoked the service.
* Because standardized return messages are not available to pass back a status message, and because web services cannot easily propagate system exceptions, the approach taken is to use SOAP exceptions to pass the HCE Admin Services exception back to the user interface layer.
* SOAP exceptions expose a ***Detail*** property, which is an XML element that can store any well formed XML snippet. The value of the Detail property is set to an XML representation of the HCE Admin Services error message, prior to raising the SOAP exception.
* When the SOAP exception is caught on the user interface, the value of the Detail property is used to generate a useful error message for the user.
* The schema used to encode the Detail property of the SOAP exception is as follows:



* + ***Entity***: defines which entity was being processed when the exception was raised.
  + ***Operation***: defines which operation was being executed on the entity when the exception was raised.
  + ***ErrorMessage***: is an additional error message to be displayed.
  + ***Details***: is an optional element that allows for extra information to be transferred in accordance with any appropriate XML schema.

## Change Notification Service

### Purpose

The need for this service has arisen from the expectation the PoS systems (or their associated Adapters) will maintain caches of information about Service Providers with whom they frequently exchange messages.

Caches are used to improve the responsiveness of the PoS system from the user’s perspective, by reducing the need to access the Service Provider Register.

The Change Notification Service is a service of the Service Provider Register that

* recognizes when significant changes have been made to Service Provider information in the Register, and
* sends unsolicited messages to all other Service Providers to enable them to update their cache.

### Design Principles

This service has been designed and implemented to conform to the following principles:

* Each Service Provider system (PoS systems and HCE-related services and registers) incorporates a Service Provider Register administration tool (utilizing the HCE Administration Services service block) to maintain its own data in the HCE Service Provider Register.
* The Service Provider Register (of course) contains a full inventory of the Service Providers configured to connect via the HCE Routing Service to other Service Providers.
* Whenever any item of information about a Service Provider (services offered, message types supported, etc) is changed in the Service Provider Register, a notification message is routed to every other Service Provider, identifying the Service Provider whose information has been changed, and the new value of each element of data.

# Additional Design Patterns

## Overview

In the course of addressing certain potential implementations, a number of other areas have been considered and the following design patterns are offered as an indication of the HCE Design Team’s thinking in these particular areas

## Health Domain Registers Overview



|  |  |
| --- | --- |
| Service Component | Description |
| Patient Register | A Patient is a person playing the role of a healthcare consumer. The patient register is an entity which is responsible for storing patient details. |
| Practitioner Register | A Practitioner is a person playing the role of a healthcare provider. The practitioner register is an entity which is responsible for storing practitioner details. |

### Data Models

As far as practicable, data design follows the principles of the Health Level 7 Reference Information Model.

The RIM specifies a common model for representing actions or events (Acts) and the relationships between them (ActRelationships). It also specifies a way to represent information about people, animals, organizations and things (Entities), the roles these entities play (Roles) and the ways in which these roles are involved in different action or events (Participations).

The RIM is conceptually applicable to any information domain involving entities playing roles and participating in acts.

### Patient, Practitioner, and ServiceProvider

Using a blend of the terminology of the RIM and the terminology of HCE:

* A Patient is a person playing the role of a healthcare consumer
* A Practitioner is a person playing the role of a healthcare provider
* A ServiceProvider is a system within an organization playing the role of a healthcare information provider or healthcare information consumer. Information is exchanged between one ServiceProvider and another by using Messages that contain information about Patients and/or Practitioners.

In RIM terms

* Person and Organization are subclasses of Entity.
* Patient, Practitioner and ServiceProvider are subclasses of Role.

It is important to recognize that a “Role” is an ***instance*** of a Patient, Practitioner or Service Provider, not a ***type -***  **Roles participate in Acts.**

The NZ NHI contains data about a patient. This data can find its home in the RIM classes Entity, LivingSubject (a subclass of Entity), Person (a subclass of LivingSubject), Role and Patient.

### Patient Register

#### Supported Message Types

Nil in HCE 2.1

#### Data Model



|  |  |
| --- | --- |
| Entity | Description |
| Patient | A person playing the role of a healthcare consumer |
| Person | The person table exists to align with the RIM principle that Patient is a role that is played by a Person, and that Person is an Entity in its own right independent of the patient role. The columns in the Person table and in the PersonName table are properties of the Person rather than of the Patient. |
| PersonName | Over time, a Person’s name may change, but at any point in time one name (usually the most recent one) is primary. The various names that have ever been associated with a person are termed “aliases” of that person.  The admin user interface should not allow aliases to be deleted. All updates should be handled by creating a new alias and flagging it as primary. If the new alias matches a previous alias in all respects (i.e. familyName, all givenNames and preferredGivenNameIndex), the existing ailias should be retained and should be flagged as primary. There is no need to provide explicitly for alias insert because update provides it implicitly. |
| MergedPatientID | As described above, patientID is the unique identifier that is used to identify a patient in messages that pass between connected systems via the HCE.  If, through error, two different patientID’s have been assigned to the same patient, a “merge” process is carried out in which one of the patientID’s is deemed to be the primary one, and use of the other is discontinued.  The merge process assumes that the correct patient details are already recorded against the primary patientID. Merging involves the following steps:   1. All patient references that previously used the discontinued patientID are changed to use the primary one. 2. A row is added to table MergedPatientID to cross-reference the two patientID’s. In the new row, patientID is the primary patientID and mergedPatientID is the discontinued patientID. 3. The row in table Patient that contains the discontinued patientID is physically deleted. |

### Practitioner Register

#### Supported Message Types

Nil in HCE 2.1

#### Business Logic

The patient register provides patient search functionality using any available data columns: patientID, address (whole or part), dateOfBirth, dateOfDeath, gender and/or name (whole or part).

If the search criteria includes a name element and/or a patientID, the search is carried out giving due consideration to all aliases and to both primary and merged patientID’s. However, results returned by the patient register to the calling system include only the primary alias and/or the primary patientID.

The intention is to implement the same patient search algorithm as used in the Connected Showcase.

#### Data Model



|  |  |
| --- | --- |
| Entity | Description |
| Practitioner | The individual playing the role of healthcare provider. |
| Person | (As in patient register) |
| PersonName | (As in patient register) |
| MergedPractitionerID | (As in patient register) |

## Asynchronous Messaging, Store and Forward and Conversation Chaining

### Purpose

“Store and Forward” provides the capability for asynchronous messaging between two collaborating Service Providers.

In conjunction with the Connected System and Adapter in use within a Service Provider, it also manages the chaining of related messages using a conversation identifier.



Figure - Asynchronous Messaging and Store and Forward conceptual design

### Design Principles

This service has been designed to conform to the following principles:

* Each collaborating system is directly interfaced to its adapter using procedure calls in either direction.
* All Adapters support both:
  + message sending to a web service exposed by the HCE Routing Service
  + message receiving via a web service exposed by the Adapter.
* Adapters store messages that are to be sent until contact can be established with the HCE Routing Service.
* The HCE Routing Service provides header validation of Connection Engine Messages received from sending Adapters, and responds synchronously with an acknowledgement and/or error response as appropriate.
* The HCE Routing Service uses the Store & Forward Service to store messages that have been received from a sending adapter until contact can be established with the receiving adapter.
* Like the Service Provider Register, the HCE Store & Forward Service responds synchronously to requests from the Routing Service.
* Receiving adapters provide payload validation of messages received from HCE, and respond synchronously with an acknowledgement and/or error response as appropriate. Error messages from Receiving adapters are routed back to the Sending adapter asynchronously for action.
* Responding PoS systems respond (synchronously or asynchronously) to their sending adapter with application-level acknowledgement of each message. The sending adapter associates the acknowledgement message with the conversation identifier of the received message, before sending it asynchronously to the HCE.
* An adapter receiving an application-level acknowledgement message recognizes it as such, and does not generate a reply. (Note that HCE processes received application-level acknowledgement messages in exactly the same way as any other message.)
* It is the responsibility of the responding PoS system, in conjunction with its adapter, to associate any future message with the same conversation identifier where appropriate. Any such subsequent messages are handled by a receiving adapter in the same manner as new messages (responses etc), even though they may contain the same conversation identifier.

### “Pull” Functionality

* A collaborating PoS system may wish to issue a request to the HCE to forward messages stored by the HCE Store & Forward Service. This situation may occur if the PoS system has been off-line for a period, or if it is normally off-line and it comes on-line for a period.
* It is not necessary for an Adapter Message Store to provide pull functionality.
* The HCE Store & Forward Service provides this capability by accepting a “Poll for pending messages” message, which the Store & Forward Service synchronously responds to with an indication of whether or not messages are queued for the calling PoS system.
* Having received such a message, the Store & Forward Service connects to the PoS system’s web service, sends the stored messages one at a time in the normal way, and then sends a “Pending messages complete” message to the PoS adapter to indicate that it contains no further messages for that PoS system.

### Configuration

Each PoS system should be configured with one of the following asynchronous messaging attributes:

* ***Always receives asynchronously*** (HCE never transmits without first receiving a “poll for pending messages” message.)
* ***May receive asynchronously*** (HCE always attempts to transmit messages and, if unable to connect, stores messages and retries periodically. Also HCE responds to “poll for pending messages” messages.)
* ***Never receives asynchronously*** (HCE always attempts to transmit messages and, if unable to connect, sends “failed to deliver” messages to the initiating PoS system.)

The following timeouts should be configured for each message type for each PoS system

* ***Period between retries to send*** (for systems configured as “may receive asynchronously”)
* ***Maximum period to store a message*** (for systems configured as “always receives asynchronously” or “may receive asynchronously”) When this period expires, the message is deleted from the queue and HCE sends a “failed to deliver” message to the initiating PoS system.

## Connection Engine Adapter

### Purpose

The Connection Engine Adapter (CEA) provides an approach to interconnecting multiple instances of the HCE and other integration eco-systems, in the expectation that these will conform to diverse standards and will be operated within diverse jurisdictions.



Figure - Connection Engine Adapter Context

### Design Principles

This component has been designed to conform to the following principles:

* It supports an arbitrary network topology interconnecting multiple integration eco-systems.
* It represents a true federation model, with peer-to-peer relationship between all engines.
* An instance of the CEA is used to interconnect any pair of engines and to manage all aspects of message flow between the two engines.
* Each PoS system is associated with its “home” instance of the HCE (or equivalent).
* A PoS Locator Service allows the Service Provider Register in each HCE to be aware of PoS systems associated with all HCE instances, and also to understand the network topology to the extent that it can route messages via CEA’s to any PoS system associated with any HCE instance.
* When a message is routed between PoS systems anywhere in the network, the message retains its meaning from link to link, irrespective of the extent to which it undergoes identifier or coding translations within one or more CEA’s.
* Each CEA that is connected to a source HCE is registered in the HCE’s Service Provider Register, along with all PoS systems that are accessed through it. The following payload encryption mechanism allows a CEA to decrypt message payloads so that they can be translated into the form required by the destination HCE:
  + Each CEA registers its public key with the source HCE.
  + All destination PoS systems that are accessed through that CEA are registered in the source HCE’s Provider Register, and each such system is registered with the CEA’s public key. In effect, the CEA is trusted as a proxy for every PoS system that is accessed through it.
  + The source PoS system, in dispatching a message to a destination PoS system via the CEA, uses the CEA’s public key to encrypt the message. It obtains this public key from the destination PoS system’s record in the HCE’s Service Provider Register.
  + This enables the CEA to decrypt the payload, translate it as required, and re-encrypt it with the real public key of the destination PoS system before forwarding it through the destination jurisdiction’s HCE to the final destination.
* The CEA is implemented as two standard Adapters (CSAs) each connected to one of the integration engines, together with a “Message Translator” that interfaces the two of CSA’s.
* A summary of the functions of each of the components of the CEA is depicted in the following diagram.

Figure - CEA Components

**HCE**

**1**

**HCE**

**2**

**Collaboration System**

**Adaptor**

**1**

Message direction

Operates synchronously

with HCE

1

Receives message from

HCE

1

Decrypts using

destination public key

from HCE

1

jurisdiction

Unpacks clinical payload

Sends message to

Message Translator

**Collaboration System**

**Adaptor**

**2**

Operates synchronously

with HCE

2

Receives message from

Message Translator

Packs clinical payload

Encrypts using

destination public key

from HCE

2

jurisdiction

Sends message to HCE

2

**Message Translator**

May operate fully

asynchronously, or may

operate synchronously with

either CSA

1

or CSA

2

Receives message from CSA

1

Translates source and

destination addresses from

HCE

1

to HCE

2

standards

Translates clinical context and

semantics of message from

HCE

1

to HCE

2

standards

Sends message to CSA

2

**HCE**

**1**

**HCE**

**2**

**Connection System**

**Adaptor**

**1**

Message direction

Operates synchronously

with HCE

1

Receives message from

HCE

1

Decrypts using

destination public key

from HCE

1

jurisdiction

Unpacks clinical payload

Sends message to

Message Translator

**Connection System**

**Adaptor**

**2**

Operates synchronously

with HCE

2

Receives message from

Message Translator

Packs clinical payload

Encrypts using

destination public key

from HCE

2

jurisdiction

Sends message to HCE

2

**Message Translator**

May operate fully

asynchronously, or may

operate synchronously with

either CSA

1

or CSA

2

Receives message from CSA

1

Translates source and

destination addresses from

HCE

1

to HCE

2

standards

Translates clinical context and

semantics of message from

HCE

1

to HCE

2

standards

Sends message to CSA

2

**Connection Engine Adapter**

## Connection Engine Message Translation Service

### Purpose

The need for this service has arisen from the expectation that changes will occur over time to the schema of the Connected Message itself.

Accordingly there may exist at the same time, Service Provider systems that use different versions of the Connected Message.

### Design Principles

This service has been designed to conform to the following principles:

* Each implementation of HCE must continue to support both incoming and outgoing messages that conform to all Connected message schemas used by any of the connected Service Provider systems.
* The HCE must expose a separate web service for each supported version of the Connected message schema.
* Each Service Provider system must be configured to access the HCE web service that supports the message version it uses.
* The HCE Service Provider Register must maintain a record of which message schema is supported by the incoming message Web Service of each Service Provider.
* Outgoing messages from the HCE Routing Service to the destination Service Provider must generate the message conformant with the message schema that is supported by the destination Service Provider.

# Appendix 1 – Connection Engine Message XML Schema

The following tables provide a definition of each element and complex type within the Connection Engine Message XML schema:

|  |  |
| --- | --- |
| Element | ConnectionEngineMessage |
| Notes |  |
| Diagram | tmp0000 |
| Namespace | <http://Microsoft.ConnectionEngine.Messaging> |

|  |  |
| --- | --- |
| Element | ConnectionEngineMessage/Header |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **HeaderType** |

|  |  |
| --- | --- |
| Element | ConnectionEngineMessage/Body |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **BodyType** |

|  |  |
| --- | --- |
| Complex Type | BodyType |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |

|  |  |
| --- | --- |
| Element | BodyType/Payload |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **PayloadType** |

|  |  |
| --- | --- |
| Complex Type | ConversationType |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |

|  |  |
| --- | --- |
| Element | ConversationType/ConversationID |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | ConversationType/OriginalMessageID |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Complex Type | HeaderType |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |

|  |  |
| --- | --- |
| Element | HeaderType/ID |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | HeaderType/CreationTime |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:dateTime** |

|  |  |
| --- | --- |
| Element | HeaderType/SessionToken |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | HeaderType/InteractionClass |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | HeaderType/InteractionType |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | HeaderType/Conversation |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **ConversationType** |

|  |  |
| --- | --- |
| Element | HeaderType/Sender |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **POSType** |

|  |  |
| --- | --- |
| Element | HeaderType/Receiver |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **POSType** |

|  |  |
| --- | --- |
| Element | HeaderType/Status |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **StatusType** |

|  |  |
| --- | --- |
| Complex Type | PayloadType |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |

|  |  |
| --- | --- |
| Element | PayloadType/EncryptionType |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | PayloadType/Key |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | PayloadType/IV |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | PayloadType/Content |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Complex Type | POSType |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |

|  |  |
| --- | --- |
| Element | POSType/OrganisationID |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | POSType/ColnnectionEngineUserID |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | POSType/OrganisationUserID |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | POSType/URI |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:anyURI** |

|  |  |
| --- | --- |
| Complex Type | StatusType |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |

|  |  |
| --- | --- |
| Element | StatusType/Code |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | StatusType/Description |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

|  |  |
| --- | --- |
| Element | StatusType/Details |
| Namespace | http://Microsoft.ConnectionEngine.Messaging |
| Diagram | tmp0000 |
| Type | **xs:string** |

# Appendix 2 – Health Connection Engine Service Index

Services delivered in HCE version 2.1:

* Message Management Services
  + Web Service Facade
  + Routing Service
    - Validation
  + Monitoring Service
* Connected Adapter
  + Adapter Base Class
  + Adapter Reference Implementation(s)
  + Adapter Software Development Kit (SDK)
* Health Services Directory Registers
  + HCE Registry Services
    - Service Provider Register
    - Invocation Register
    - Change Notification Service
  + Health Domain Registry Services
    - Patient Register
    - Practitioner Register
    - Consents Register
* Health Services Directory Administration Services
  + Service Provider Administration Services
  + Patient Administration Services
  + Practitioner Administration Services
  + Consents Administration Services
* Administration Portal
  + Portal Desktop
  + Service Provider Administration Portal
  + Patient Administration Portal
  + Practitioner Administration Portal
  + Consents Administration Portal
* Security Envelope
* Service Provider

# Appendix 3 – Message Types

## Summary of support message types

|  |  |  |
| --- | --- | --- |
| ID | Description | XML Schema File |
| 1 | Patient Referral | EMR.xsd |
| 2 | Laboratories Order Data | EMR.xsd |
| 3 | Laboratory Order Result | report\_v3.xsd |
| 4 | Clinical Decision Support Result | XMLschema.xml |
| 5 | Pharmacy Dispensing Status | HPPharmacyPatient.xsd |
| 6 | ServiceProvider Select Result | serviceprovider.xsd |
| 7 | Patient Record Request | EMRRequest.xsd |
| 8 | Patient Record Summary | EMR.xsd |
| 1000 | Service Provider Search Request | ServiceProviderID.xsd |
| 1001 | Service Provider Search Result | serviceproviderlist.xsd |
| 1002 | Health Service Directory Reference Data Request | hsdreferencedatarequest.xsd |
| 1003 | Health Service Directory Reference Data Result | hsdreferencedata.xsd |
| 1004 | Healthcare User Identifier Request | hcuidentifierrequest.xsd |
| 1005 | Healthcare User Identifier Result | hcuidentifier.xsd |
| 1201 | Invocation Registry Item | InvocationRegistryItem.xsd |
| 1200 | Invocation Registry Item Request | InvocationRegistryItemRequest.xsd |
| 1014 | Service Provider Search Request | ServiceProviderSearchRequest.xsd |
| 1008 | Service Provider Lock | serviceprovideridrequest.xsd |
| 1006 | Service Provider Delete | serviceprovideridrequest.xsd |
| 1007 | Service Provider UnLock | serviceprovideridrequest.xsd |
| 1009 | Service Provider Update | serviceprovidereditrequest.xsd |
| 1010 | Service Provider Insert | serviceprovidereditrequest.xsd |
| 1012 | Service Provider Update Result | serviceprovider.xsd |
| 1013 | Service Provider Check- Out Response | serviceprovideredit.xsd |
| 1016 | Pool Delete | poolidrequest.xsd |
| 1017 | Pool Check-In | poolidrequest.xsd |
| 1018 | Pool Check-Out | poolidrequest.xsd |
| 1019 | Pool Update | pooleditrequest.xsd |
| 1020 | Pool Insert | pooleditrequest.xsd |
| 1021 | Pool Insert Response | pool.xsd |
| 1022 | Pool Update Response | pool.xsd |
| 1023 | Pool Check-Out Response | pool.xsd |
| 1026 | Remove ServiceProvider From Pool | ServiceProviderPoolInsertDelete.xsd |
| 1027 | Add ServiceProvider To Pool | ServiceProviderPoolInsertDelete.xsd |
| 1015 | Service Provider Search Response | ServiceProviders.xsd |
| 1024 | Pool Search Request | poolsearchrequest.xsd |
| 1025 | Pool Search Response | pools.xsd |
| 1028 | Remover ServiceProviderType From Pool | poolserviceprovidertypeeditrequest.xsd |
| 1029 | Add ServiceProviderType To Pool | poolserviceprovidertypeeditrequest.xsd |
| 1030 | ServiceProviderType Delete | serviceprovidertypeidrequest.xsd |
| 1031 | ServiceProviderType Check-In | serviceprovidertypeidrequest.xsd |
| 1032 | ServiceProviderType Check-Out | serviceprovidertypeidrequest.xsd |
| 1033 | ServiceProviderType Update | serviceprovidertypeeditrequest.xsd |
| 1034 | ServiceProviderType Insert | serviceprovidertypeeditrequest.xsd |
| 1035 | ServiceProviderType Insert Response | serviceprovidertype.xsd |
| 1036 | ServiceProviderType Update Response | serviceprovidertype.xsd |
| 1037 | ServiceProviderType Check-Out Response | serviceprovidertype.xsd |
| 1011 | Service Provider Insert Response | serviceprovider.xsd |
| 1038 | Pool Check-In All | poolunlockall.xsd |
| 1039 | ServiceProviderType Check-In All | serviceprovidertypeunlockall.xsd |
| 1040 | ServiceProvider Unlock All | serviceproviderunlockall.xsd |
| 1041 | ServiceProviderType Search Request | serviceprovidertypesearchrequest.xsd |
| 1042 | ServiceProviderType Search Response | serviceprovidertypes.xsd |
| 1054 | Add Service Provider Types to MessageType | ServiceProviderTypeMessageTypeLinkEditRequest.xsd |
| 1055 | Remove Service Provider Types from Message Type | ServiceProviderTypeMessageTypeIDRequest.xsd |
| 1064 | ServiceProviderType SelectItem Request | serviceprovidertypeidrequest.xsd |
| 1065 | ServiceProviderType SelectItem Result | serviceprovidertype.xsd |
| 1063 | PoolServiceProviderTypes Insert List Request | poolserviceprovidertypelinkListEdit.xsd |
| 2000 | Validate User Request | usercredentials.xsd |
| 2001 | Validate User Response | hsdprincipal.xsd |
| 1043 | MessageType Search Request | messagetypesearchrequest.xsd |
| 1044 | MessageType Search Response | messagetypelist.xsd |
| 1045 | MessageType Check-In | messagetypeidrequest.xsd |
| 1046 | MessageType Check-Out | messagetypeidrequest.xsd |
| 1047 | MessageType Check-Out Response | messagetype.xsd |
| 1048 | MessageType Check-In All | messagetypeunlockall.xsd |
| 1049 | MessageType Insert | messagetypeeditrequest.xsd |
| 1050 | MessageType Insert Result | messagetype.xsd |
| 1051 | MessageType Update | messagetypeeditrequest.xsd |
| 1052 | MessageType Update Result | messagetype.xsd |
| 1053 | MessageType Delete | messagetypeidrequest.xsd |
| 1058 | ServiceProviderType MessageType Insert List | ServiceProviderTypeMessageTypeLinkListEdit.xsd |
| 1059 | ServiceProviderTypeMessageType Search Request | ServiceProviderTypeMessageTypeSearchRequest.xsd |
| 1060 | ServiceProviderTypeMessageType Search Result | ServiceProviderTypeMessageTypeList.xsd |
| 1056 | MessageType SelectItem Request | messagetypeidrequest.xsd |
| 1057 | MessageType SelectItem Result | messagetype.xsd |
| 1066 | Pool SelectItem Request | poolidrequest.xsd |
| 1067 | Pool SelectItem Result | pool.xsd |
| 1061 | Pool ServiceProviderType Search Request | poolServiceProviderTypeSearchRequest.xsd |
| 1062 | Pool ServiceProviderType Search Result | poolServiceProviderTypelist.xsd |
| 1068 | Pool ServiceProviderType Delete List | poolserviceprovidertypelinkListEdit.xsd |
| 1069 | ServiceProviderType MessageType Delete List | ServiceProviderTypeMessageTypeLinkListEdit.xsd |
| 1070 | ServiceProvider Select Request | serviceprovideridrequest.xsd |
| 1071 | ServiceProvider Select Result | serviceprovider.xsd |
| 9003 | ServiceProviderPool Delete List Request | serviceproviderpoollinkListEdit.xsd |
| 9004 | ServiceProviderPool Insert List Request | serviceproviderpoollinkListEdit.xsd |

# Appendix 4 – Routing Service BizTalk Implementation

This section contains logical and physical implementation details for the ReceiveMessage, ValidateMessage and ProcessMessage orchestrations implemented within the Routing Service.

The diagrams in this section have been generated based on the BizTalk Server 2006 implementation for each orchestration.

#### ReceiveMessage Orchestration

Begin

Receive InboundMessage from prtConnectionEngineInbound:

Receives Connection Engine Message

.

Send returnMessage to prtConnectionEngineInbound

:

End

Call Orchestrations.ConnectionEngineValidateMessage

sub-process:

.

Decide

Construct message returnMessage

:

ConstructReturnMessage

Is Empty : Inbound Message. Head…

Construct message msgConnectionEngine

:

InitializeMsgConnectionEngine

Perform ConversationID = System.Guid.NewGuid();:

Construct message returnMessage

:

SetTimeoutStatusCode

Construct message returnMessage

:

SetGeneralErrorStatusCode

Else

Else

Message Is Valid: InboundMess…

Decide

Decide

Construct message msgConnectionEngine:

Initialize MsgConnectionEngine

Call Orchestrations.ConnectionEngineProcessMessage

sub-process:

Catch System.Net.WebException: CatchTimeout

Catch System.Exception: CatchGeneral

Figure - ReceiveMessage logical implementation

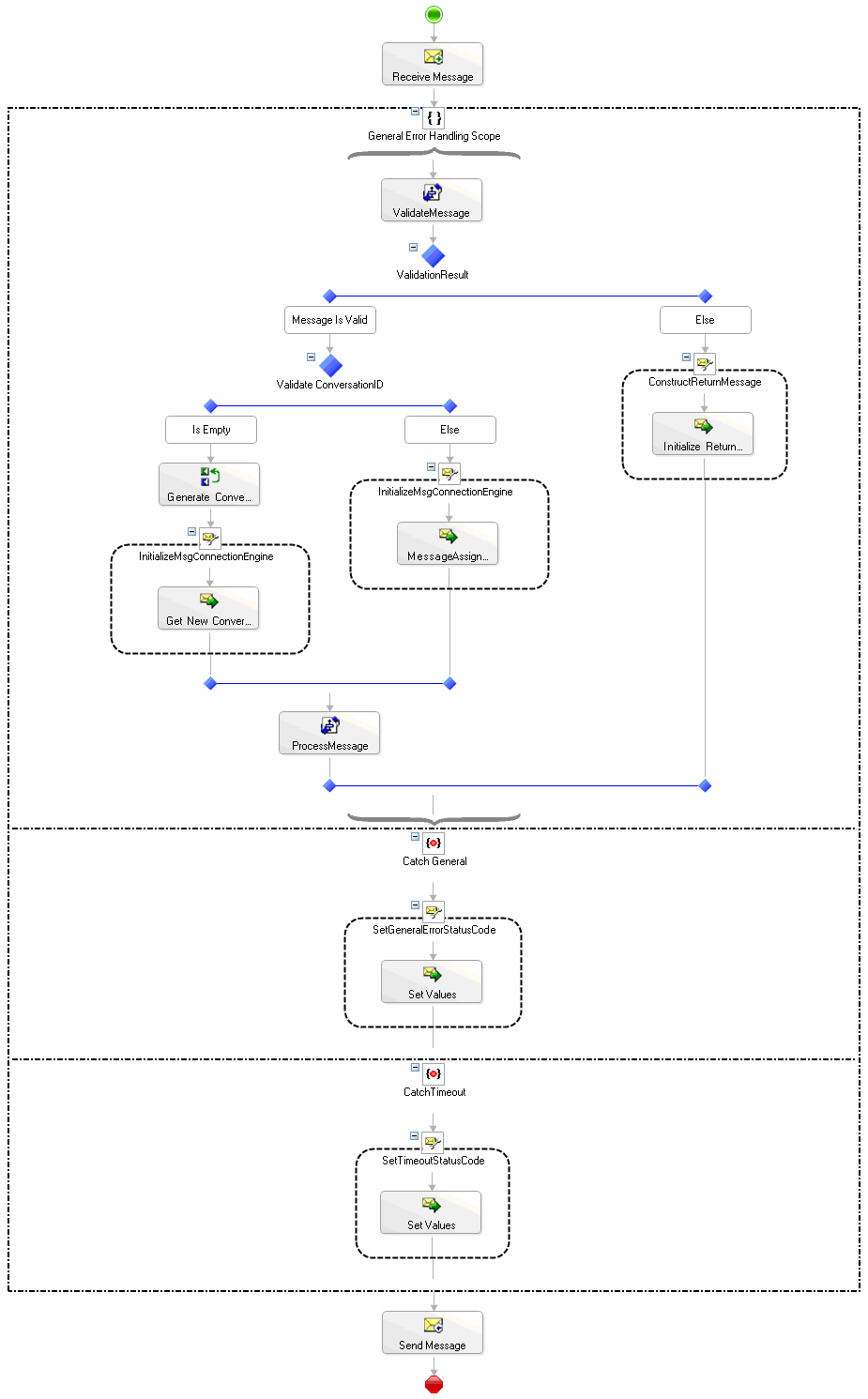


Figure - ReceiveMessage physical implementation

#### ValidateMessage Orchestration

Construct message msgValidateRequest:

Construct msgValidateRequest

Send msgValidateRequest to prtConnectionEngineValidation:

Receive msgValidateResponse from prtConnectionEngineValidation:

Construct message ValidatedMessage:

:

Construct Validated Message

End

Perform prtConnectionEngineValidation(Microsoft XLANGs.BaseTypes.Address)

=System.Configuration.ConfigurationSettings.AppSettings.Get(“HSD.Validation.DestinationURL”);:

Begin

Figure - ValidateMessage logical implementation

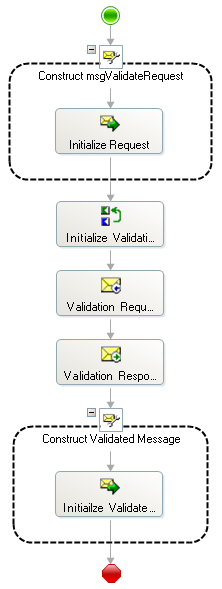


Figure - ValidateMessage physical implementation

#### ProcessMessage Orchestration

- MessageToProcess.Header.Receiver.URI

Construct message msgProcessRequest

Construct msgProcessRequest

Send msgProcessRequest to prtConnectionEngineProcess

Receive msgProcessResponse from prtConnectionEngineProcess

Construct message ProcessResult:Construct

ProcessResult

End

Perform prtConnectionEngineProcess (Microsoft.XLANGs.BaseTypes.Address)

Begin

Figure - ProcessMessage logical implementation

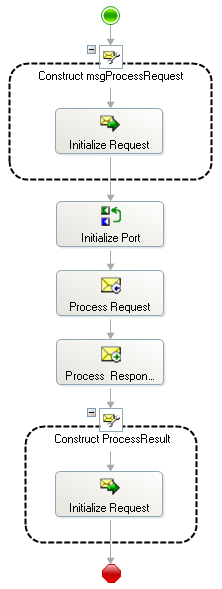


Figure - ProcessMessage physical implementation

# Appendix 5 – Adapter Base Class Diagram

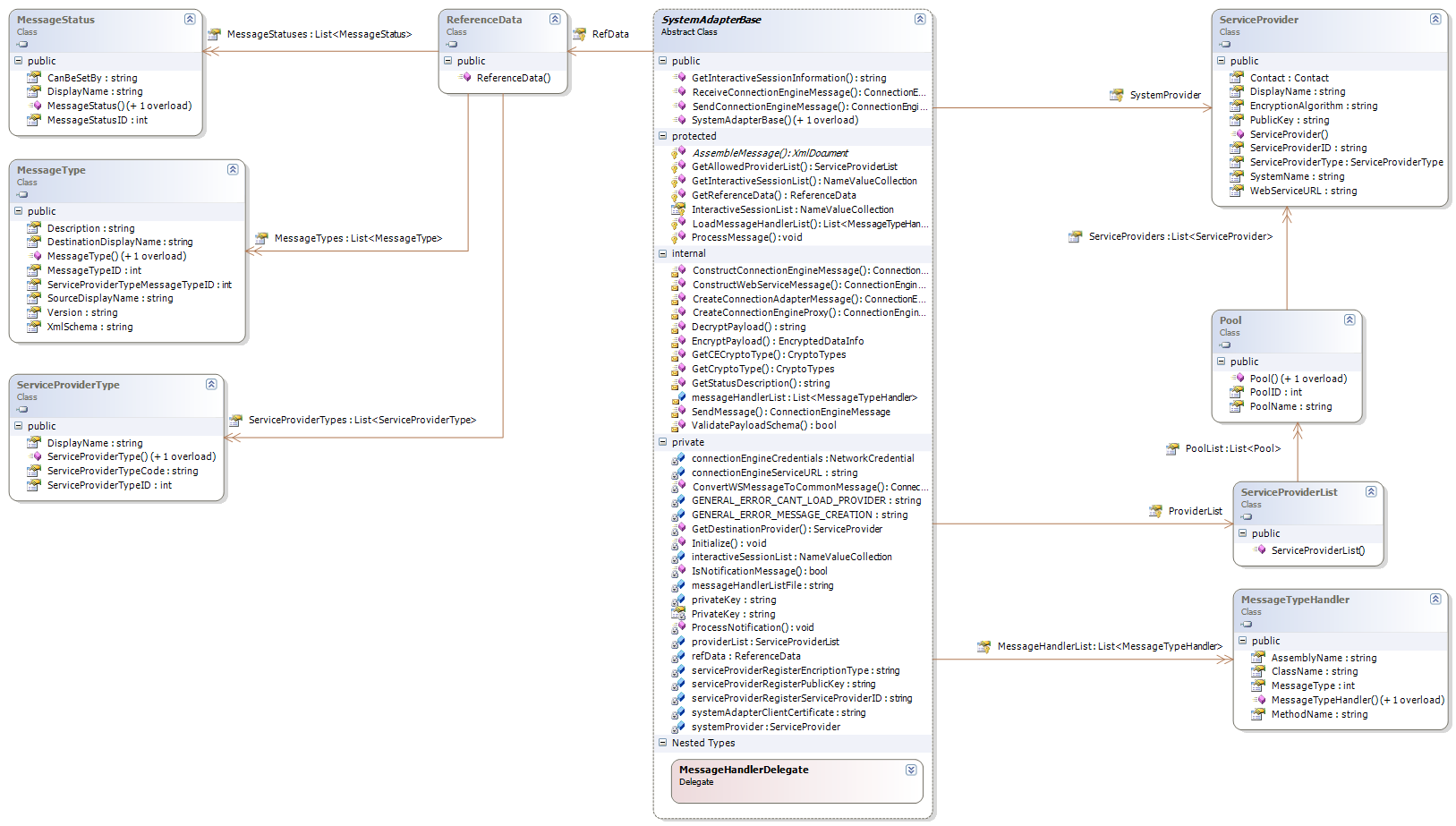


Figure - Adapter Base Class Diagram

# Appendix 6 – ProcessMessage source code

/// <summary>

/// Process a Collaboration Engine compatible Payload based on its Message Type.

/// </summary>

/// <param name="xmlMessage">Collaboration Engine compatible Payload to be processed</param>

/// <param name="msgType">The Message Type that defines the Payload to be processed.</param>

/// <param name="resultXmlMessage">An XML representation of the returning payload.</param>

/// <param name="resultMsgType">A representation of the Message Type for the returning payload.</param>

/// <returns>Return a boolean indicating if the Process generated a new message type or only an acknowledgment.</returns>

protected virtual void ProcessMessage(System.Xml.XmlDocument xmlMessage, int msgType, out System.Xml.XmlDocument resultXmlMessage, out int resultMsgType)

{

resultXmlMessage = new System.Xml.XmlDocument();

try

{

// Finds the MessageTypeHandler using an anonymous method inside the Find predicate.

common.MessageTypeHandler msgHandler = MessageHandlerList.Find(delegate(common.MessageTypeHandler node)

{

if (node.MessageType == msgType)

return true;

else

return false;

});

System.Reflection.Assembly assembly = Assembly.LoadFrom(System.AppDomain.CurrentDomain.RelativeSearchPath + @"\" + msgHandler.AssemblyName);

Type t = assembly.GetType(msgHandler.ClassName);

MessageHandlerDelegate dlgt = (MessageHandlerDelegate)

Delegate.CreateDelegate(

typeof(MessageHandlerDelegate),

t, msgHandler.MethodName);

dlgt.Invoke(xmlMessage, out resultXmlMessage, out resultMsgType);

}

catch (System.Exception ex)

{

common.SystemAdapterException sysEx = new common.SystemAdapterException(

common.SystemAdapterExceptionCause.PayloadContentNotValid,

"An error occurred processing the payload.",

ex);

throw sysEx;

}

}

1. An expanded version of this diagram can be found in Appendix 5 – Adapter Base Class Diagram [↑](#footnote-ref-2)