Microsoft Health Connection Engine 2.1

Adapter Development Guide

11/1/2006

Microsoft

Table of Contents

[Introduction 1](#_Toc151804478)

[Overview 1](#_Toc151804479)

[Health Connection Engine (HCE) Background 1](#_Toc151804480)

[The Strategic Challenge for Connected Health 1](#_Toc151804481)

[HCE Description 2](#_Toc151804482)

[Design Principles 2](#_Toc151804483)

[What HCE is not 3](#_Toc151804484)

[Health Connection Engine Service Blocks 4](#_Toc151804485)

[Connection Engine Message 7](#_Toc151804486)

[Message Structure 7](#_Toc151804487)

[Message Schema 9](#_Toc151804488)

[Message Management Services 10](#_Toc151804489)

[Routing Service 10](#_Toc151804490)

[Implementation Overview 11](#_Toc151804491)

[Message Flow 12](#_Toc151804492)

[BizTalk Configuration 14](#_Toc151804493)

[Adapters 15](#_Toc151804494)

[Purpose and Design Principles 15](#_Toc151804495)

[Overview 16](#_Toc151804496)

[Implementing an Adapter 16](#_Toc151804497)

[Adapter Base Class 16](#_Toc151804498)

[Adapter Classes 18](#_Toc151804499)

[Adapter Web Service 20](#_Toc151804500)

[Adapter Message Flow 21](#_Toc151804501)

[SystemAdapterBase – Instantiating an Adapter 21](#_Toc151804502)

[SendConnectionEngineMessage – Sending a Message 22](#_Toc151804503)

[ReceiveConnectionEngineMessage – Receiving a Message 22](#_Toc151804504)

[GetInteractiveSessionInformation -- Generating an Interactive Session URL 23](#_Toc151804505)

[Configuration Parameters 24](#_Toc151804506)

[Exception Handling 26](#_Toc151804507)

[Appendix 1 – Connection Engine Message XML Schema 26](#_Toc151804508)

[Appendix 2 – Adapter Base Class Diagram 39](#_Toc151804509)

[Appendix 3 – Process Message source code 40](#_Toc151804510)

# Introduction

This document provides an introduction to the Health Connection Engine (HCE) and design information about the Adapters which are used to enable ISV systems to connect to the HCE and participate in a connected healthcare solution supported by the HCE platform.

The sections within this document provide the following background and design information:

|  |  |
| --- | --- |
| Section | Description |
| Conceptual Background | Background material providing an overview of the HCE, its purpose and overall design principles |
| Health Connection Engine Service Blocks | Background information providing a logical representation of the service provided by the HCE. |
| Connection Engine Message | Design information about the XML message structure used to pass messages between a connected ISV system and the HCE Routing Service |
| Message Management Services | Background material explaining the functionality provided by the Routing Service within the HCE |
| Adapter | Design information related to implementing a adapter |

This document is intended to be provided as part of the Adapter Development Kit and should be read in conjunction with the other resources provided within that kit.

# 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 features help ensure 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 improved 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

Figure 3 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 Figure 4. Further information about the definition and use of each element and complex type within the schema can be found in section 7 of this document.

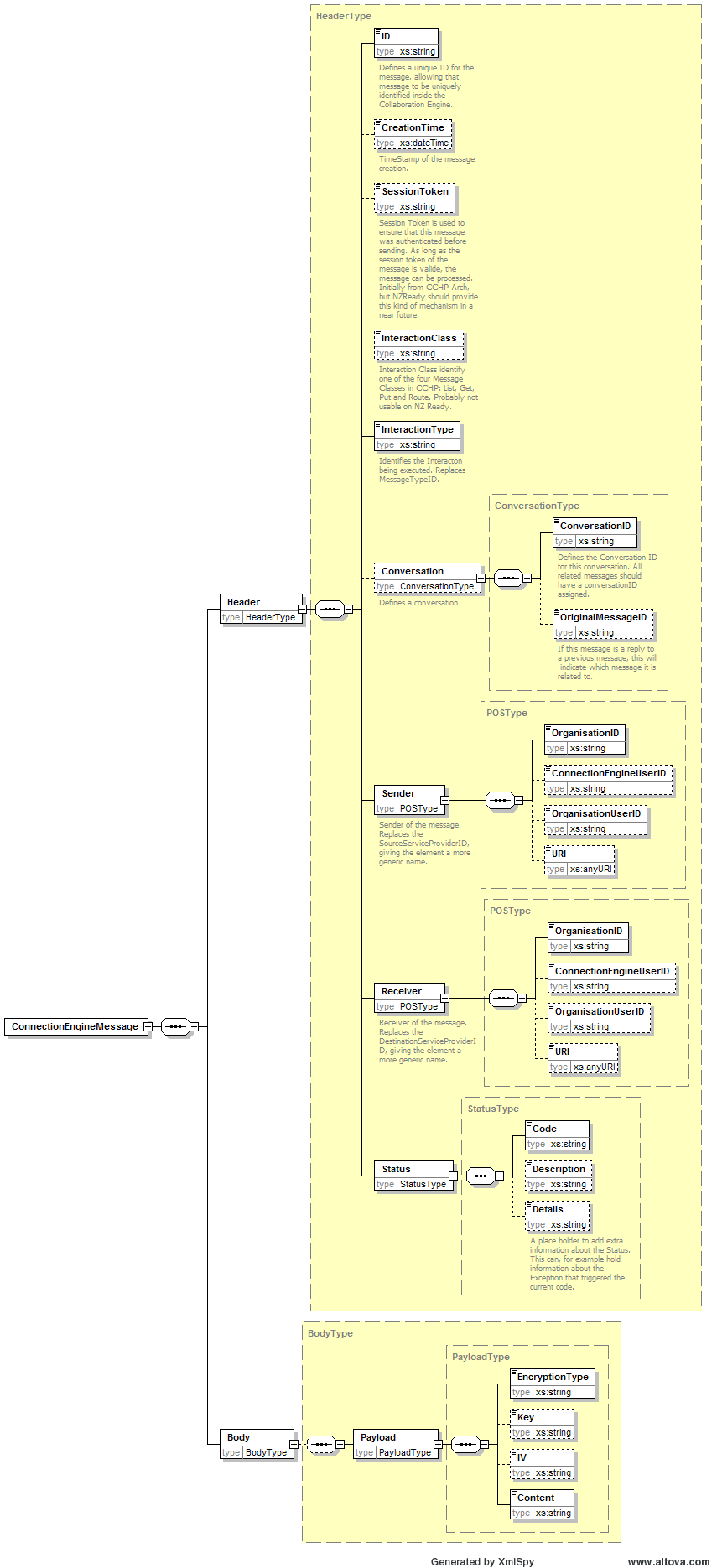


Figure - Connection Engine Message XML Schema

# 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 6 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):

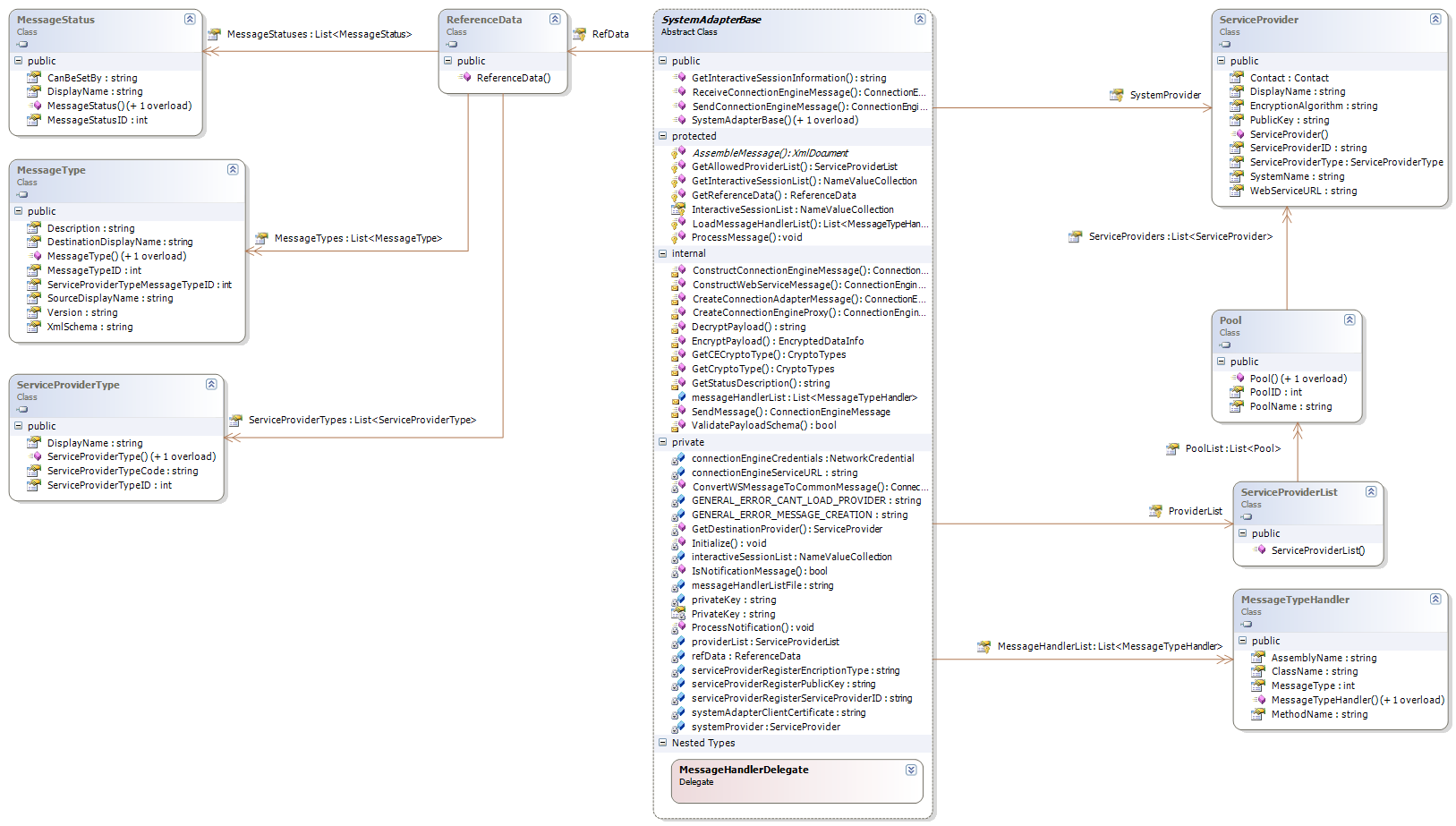


Figure - Adapter Base Class Structure

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.

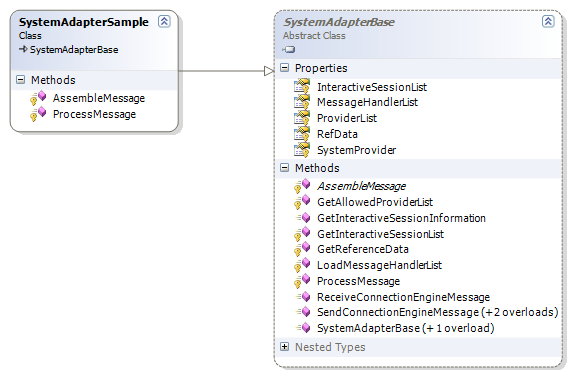


Figure – Example Adapter class inherited from the Adapter Base class

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 6.4 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
  + ReceiveConnectionEngienMessage
  + 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 9 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 tunneling). |
| 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 |

# 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 |  |
| Namespace | <http://Microsoft.ConnectionEngine.Messaging> |

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Appendix 2 – Adapter Base Class Diagram

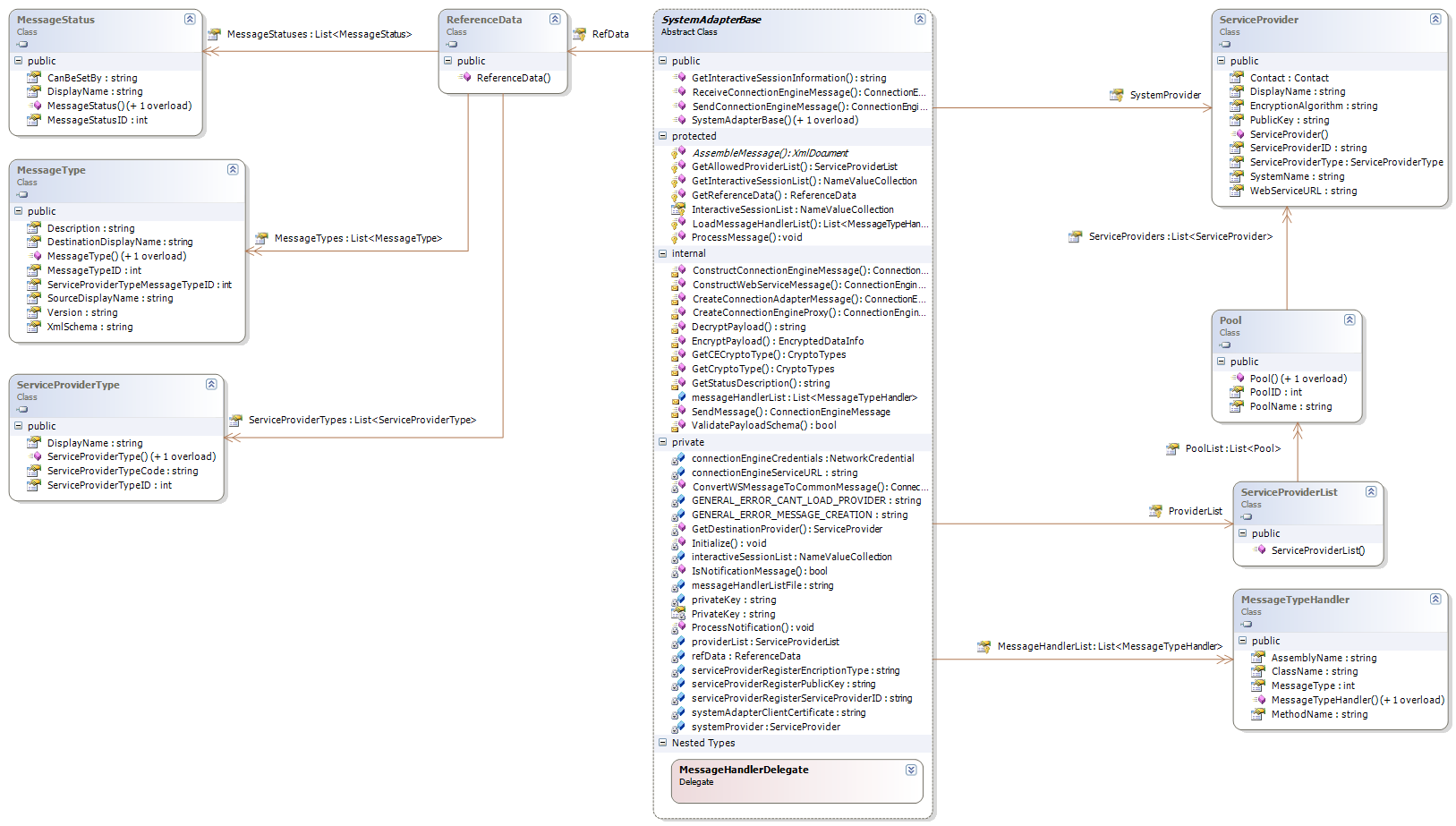


Figure - Adapter Base Class Diagram

# Appendix 3 – Process Message 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 section 8 [↑](#footnote-ref-2)