

## **CCDWS**

### Common Communication Driver for Web Services Technical Design

## Table of Contents

1.	Introduction .....	1
2.	Related Documents .....	1
3.	Development Tools and Frameworks .....	1
3.1.	The Relay Framework.....	1
3.2.	gSOAP .....	1
3.3.	OpenSSL .....	2
4.	Logic Flow .....	3
4.1.	Configure .....	3
4.2.	Poll for Inputs.....	4
4.3.	Configure Destination .....	4
4.4.	Extract Payload from Input File.....	4
4.5.	Build Web Services Message.....	4
4.6.	Establish SSL Connection.....	4
4.7.	Transmit Web Services Message .....	4
4.8.	Receive Response .....	4
4.9.	Drop SSL Connection .....	5
4.10.	Write Output File .....	5
5.	Object Diagram .....	5
5.1.	CCcdws .....	5
5.2.	CSubmissionEntry .....	6
5.3.	CPayloadCDAnet.....	6
5.4.	CLogger .....	6
5.5.	CConfig.....	6
6.	Class Diagram .....	7
6.1.	CCcdws .....	7
6.2.	CSubmissionEntry .....	7
6.3.	CRelay.....	8
6.4.	CNTService.....	8
6.5.	CPayloadCDAnet.....	8
6.6.	CPayload .....	8
6.7.	CMsgPart .....	8
6.8.	CLogger .....	9
6.9.	CConfig.....	9
7.	Web Service Communications.....	9
7.1.	WSDL File .....	9
7.2.	Sample Request and Response Messages .....	10
8.	Compiling CCDWS .....	10
8.1.	Obtaining and Extracting the Source Code.....	10
8.2.	Development Environment .....	10
8.3.	Compiling.....	10

## Revision History

Date	Revision	CCDWS Version	Changes
2011-10-25	1.0	0.9.0 (beta)	❑ Initial revision
2011-11-01	1.1	0.9.0 (beta)	❑ Updated information on gSOAP licensing. ❑ Added <i>Compiling CCDWS</i> section
2011-11-03	1.2	0.9.0 (beta)	❑ Removed OpenSSL run-time library requirement
2011-11-07	1.3	0.9.1 (beta)	❑ Changed SignMessage option ❑ Removed all sections contained in <i>Technical Reference and Operating Manual</i> ❑ Added <i>Related Documents</i> section
2012-01-11	1.4	0.9.4 (beta)	❑ Added other related documents ❑ Updated WSDL ❑ Changes to include data types other than CDAnet
2016-11-1	1.5	1.0.5	❑ Changed Harder Software Logo ❑ Changed HIEC references to CLHIA

# 1. Introduction

This document provides technical design information for the Common Communications Driver for Web Services (CCDWS), sponsored by the Canadian Life and Health Insurance Association Inc. (CLHIA), and designed by Harder Software. CCDWS is designed to act as a communications interface between dental offices and healthcare insurers.

CCDWS is designed to run as a service application, polling for input request files. Upon discovery of an input file it will perform all tasks necessary to submit the claim and receive the adjudication response or acknowledgement in real time. It will operate in a parallel with the original CCD application, so that both carriers that accept dialup claims and those that accept web service claims can be supported simultaneously without modification to the dental software.

While the underlying architecture of CCDWS supports multithreading, it performs its operations in a single threaded fashion. Each input request will be submitted and output response received prior to its next submission. This will generally not be an issue, as the communications are very quick, and expected claims traffic in a dental office is low.

## 2. Related Documents

To obtain copies of the following documents, contact a participating HIEC member.

*CCDWS Requirements* was authored by Alberta Blue Cross to provide the requirements and overall parameters for development of CCDWS.

*CCDWS – Technical Reference* can be used by software vendors to configure and troubleshoot CCDWS when integrated with their dental or other practice software. This document was authored by Harder Software.

*CCDWS – Test Plan* can be used by software vendors and carriers to ensure that CCDWS is working as designed. This document was authored by Harder Software.

*Point of Service (PoS) Application Services Transport Specification, Draft 1.9* defines much of the underlying transport specific specifications used for SOAP transactions.

## 3. Development Tools and Frameworks

A set of development tools and frameworks are employed to simplify the development process. These include a relay framework developed by Harder Software, gSOAP, and OpenSSL.

### 3.1. The Relay Framework

Harder Software developed a Relay Framework to provide reusable communication components. This framework is employed for the CCDWS application to provide a basis for the Windows service, logging, and configuration functionalities.

### 3.2. gSOAP

gSOAP is an open source set of tools that provides automated SOAP and XML data binding for C and C++ applications. Using the HIEC CDAnet WSDL, functions and classes were generated to support the specified web service calls, including the WS-\* extensions. The CCDWS binary and related files may be distributed free of charge under the gSOAP public license. However, those generating binaries from source code for commercial

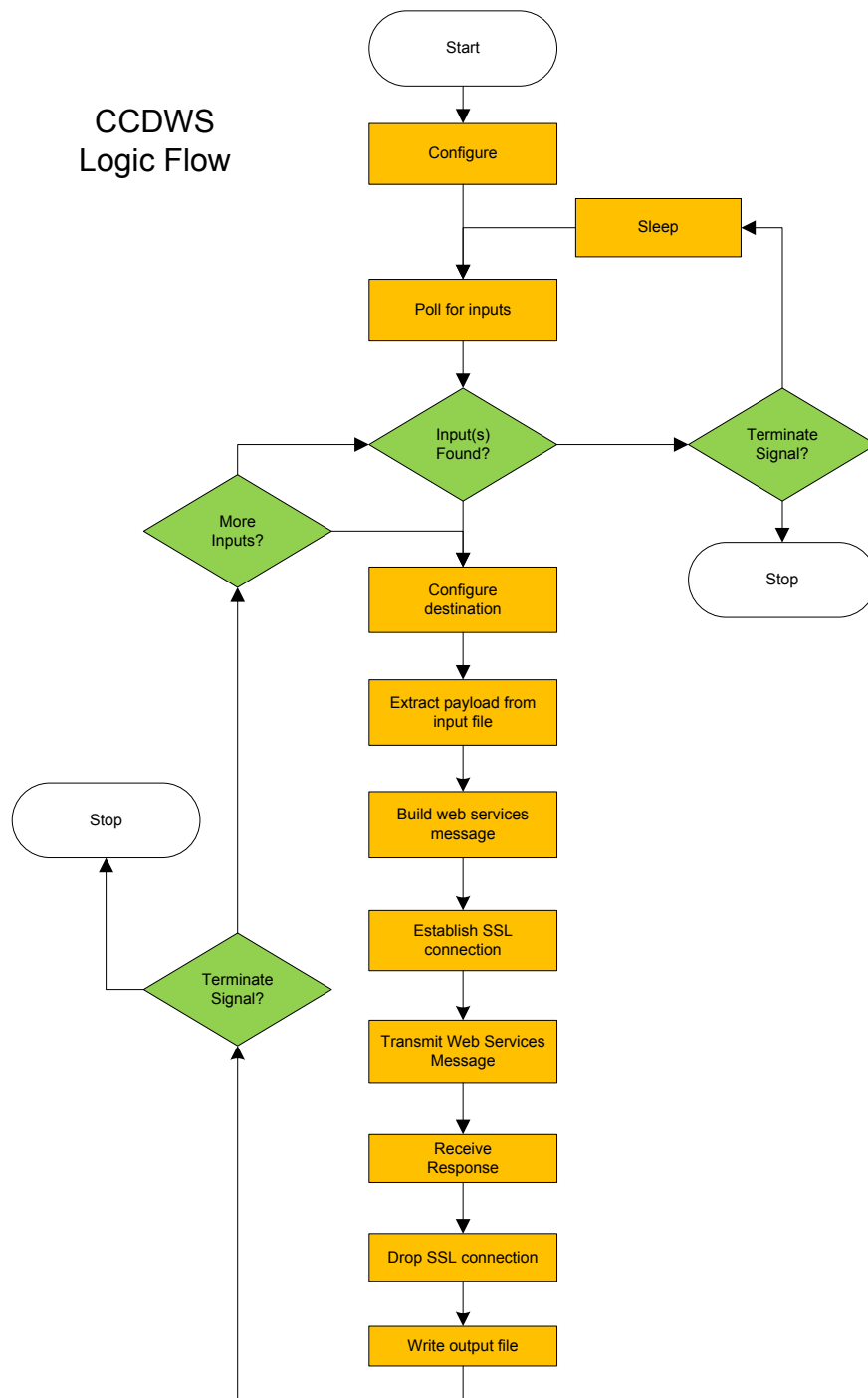
distribution require a commercial software development license. Note that Harder Software has obtained this license. For more information on gSOAP and its licensing, see <http://www.cs.fsu.edu/~engelen/soap.html>.

### **3.3. *OpenSSL***

OpenSSL is an open source toolkit implementing Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols, as well as a general purpose encryption library. OpenSSL is used to provide the strong encryption required by the CCDWS requirements. See <http://www.openssl.org> for more information.

## 4. Logic Flow

This diagram describes the flow of logic through the CCDWS application from the time it starts until it stops. Functional components are described in more detail following the diagram.



### 4.1. Configure

When the CCDWS application starts, it must first configure itself. To do so, it performs the following steps:

- Read the configuration file ccdws.ini. The default location of this file is c:\ccd\ccdws.ini. However, this

may be overridden by the `-c` command line option. If running as a service, this will have been stored in the registry so that it can recall this option each time it starts. All configuration options will be stored in a CConfig class instance for retrieval as necessary. See the section entitled *Configuration Options* for details on the various options.

- b. From the [Destination X] sections, retrieve all of the destinations that CCDWS will monitor for inputs, and store in a destinations list.

## **4.2. Poll for Inputs**

Each of the destination directories are to be stored beneath the directory defined as *RootDir*. By default this is `c:\ccd`, but can be altered in the configuration file. A section [Destination X] in the configuration file means that CCDWS should look in the X directory beneath the *RootDir*. All files with the input base name (configurable with the *InputFileName* option) in any of the destination directories will have their filenames stored in an unordered submissions list for processing.

If no inputs are found, then CCDWS sleeps for a configurable number of seconds (one second by default), and polls again, otherwise it continues on to process each input.

## **4.3. Configure Destination**

During this process, the next entry from the submission list is selected. From this entry, the destination is obtained, and CCDWS then looks to the configuration file for details. These will include endpoint, optional action, and various security options. See the *Configuration Options* section for details.

## **4.4. Extract Payload from Input File**

Once the destination is configured, the input file is opened and the payload retrieved. The file handle remains open after this as in Windows it will disallow other applications from attempting to update it while it is being processed. Additionally, the output file will be created with the file handle also remaining open.

## **4.5. Build Web Services Message**

Wrap the message in a SOAP wrapper, with appropriate WS-Addressing and/or WS-Security extensions. Options for these are stored in the destination section of the configuration file.

## **4.6. Establish SSL Connection**

Use the CA certificate and client certificate/key pair to establish the SSL connection, based on SSL options stored in the destination section of the configuration file. The *ConnectTimeout* configuration option will determine how long to wait for a connection before timing out.

## **4.7. Transmit Web Services Message**

Submit the wrapped message to the endpoint and action specified in the destination section of the configuration file. The *ReadWriteTimeout* configuration option will determine how long to wait for delivery of the message until it times out.

## **4.8. Receive Response**

Once the message has been submitted, block until the response is received. The *ReadWriteTimeout* configuration option will determine how long to wait for the response until it times out.

## 4.9. Drop SSL Connection

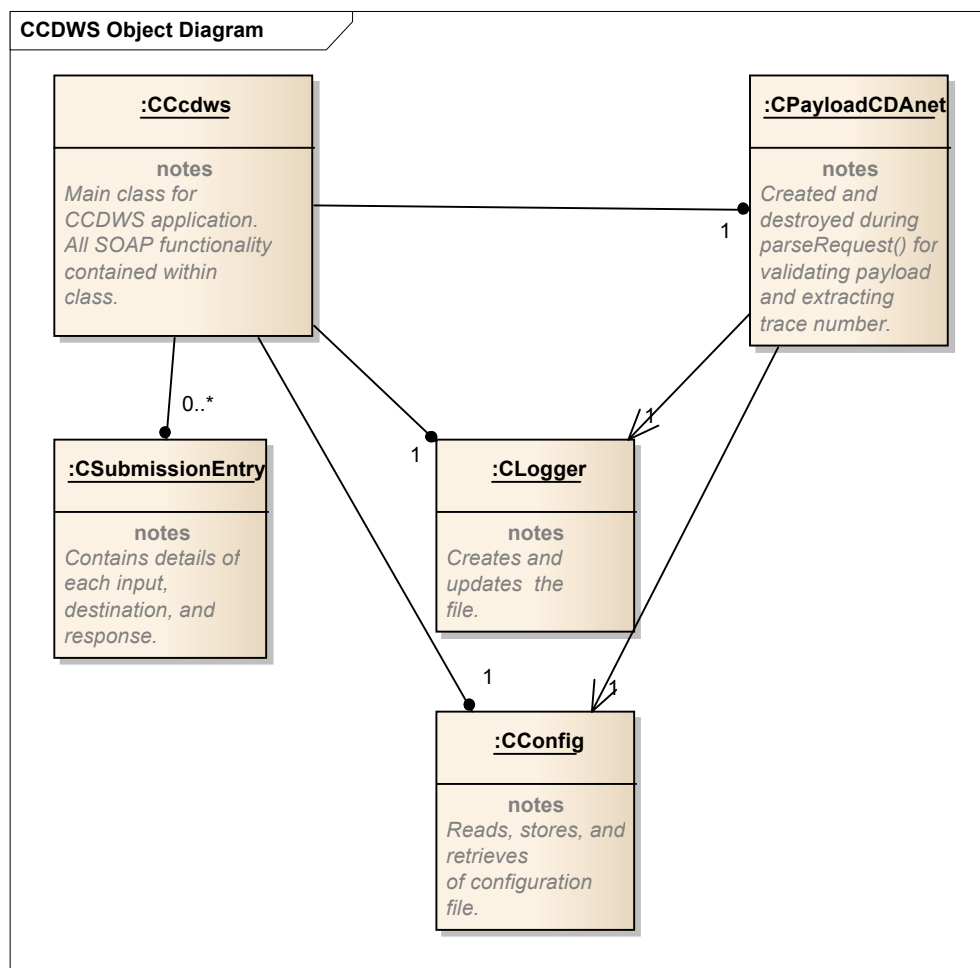
Elegantly close the SSL and socket connection.

## 4.10. Write Output File

Regardless of the success of the prior processes, the output file must be written according the output file format specifications. If an error is detected by CCDWS, then that appropriate status code will be included in the response and no payload written. The output file handle will then be closed as well as the input file handle. The input filename will then be renamed with the first character as an underscore ('\_') so that CCDWS will not attempt to submit it again.

## 5. Object Diagram

The CCDWS application contains a small number of instantiated classes when operating. These are shown in the following diagram, and described below:



### 5.1. CCcdws

The CCcdws class is the main CCDWS application class, containing the main functional loop as shown in the logic flow diagram. A single instance of the CCcdws class is created. All files are read and written, and all SOAP

functions performed within it.

**Contains:** vector of CSubmissionEntry, CPayloadCDAnet, CLogger, CConfig

## **5.2. CSubmissionEntry**

A vector of CSubmissionEntry objects is contained in the CCdws object each time it polls for new inputs. It can contain zero or more objects, depending on how many inputs are discovered. Each entry will be populated with the destination id, input filename extension, input and output content and any other data required to submit a claim request and receive its response. Some of this will be gathered from the destination section of the configuration file. After each submission is completed it is removed from the vector and the next one then processed.

**Contained by:** CCdws

**Cardinality:** 0..\*

## **5.3. CPayloadCDAnet**

For CDAnet data types, an instance of CPayloadCDAnet is contained within the parseRequest() method of the CCdws object. This object is used to detect input parse errors and to extract the office sequence number (CDAnet field A02) that is to be placed in the first field of the output file. Other data types will generate an object of the parent class CPayload, and perform no special parsing, storing a 0 in the first field of the output file. It references the CConfig and CLogger objects contained within the CCdws object, to obtain configuration information and write to the log file.

**Contained by:** CCdws

**Cardinality:** 1

## **5.4. CLogger**

The CLogger class creates an output log file with an interface to allow multiple objects to simultaneously write to it. An instance of CLogger is contained within CCdws, and referenced by CPayloadCDAnet, so that both classes can write to the log.

**Contained by:** CCdws

**Referenced by:** CPayloadCDAnet

**Cardinality:** 1

## **5.5. CConfig**

The CConfig class reads from a Windows-style configuration file, storing it internally. It has various methods to extract sections, options, and other information. An instance of CConfig is contained within CCdws, and referenced by CPayloadCDAnet, so that both classes can write to the log.

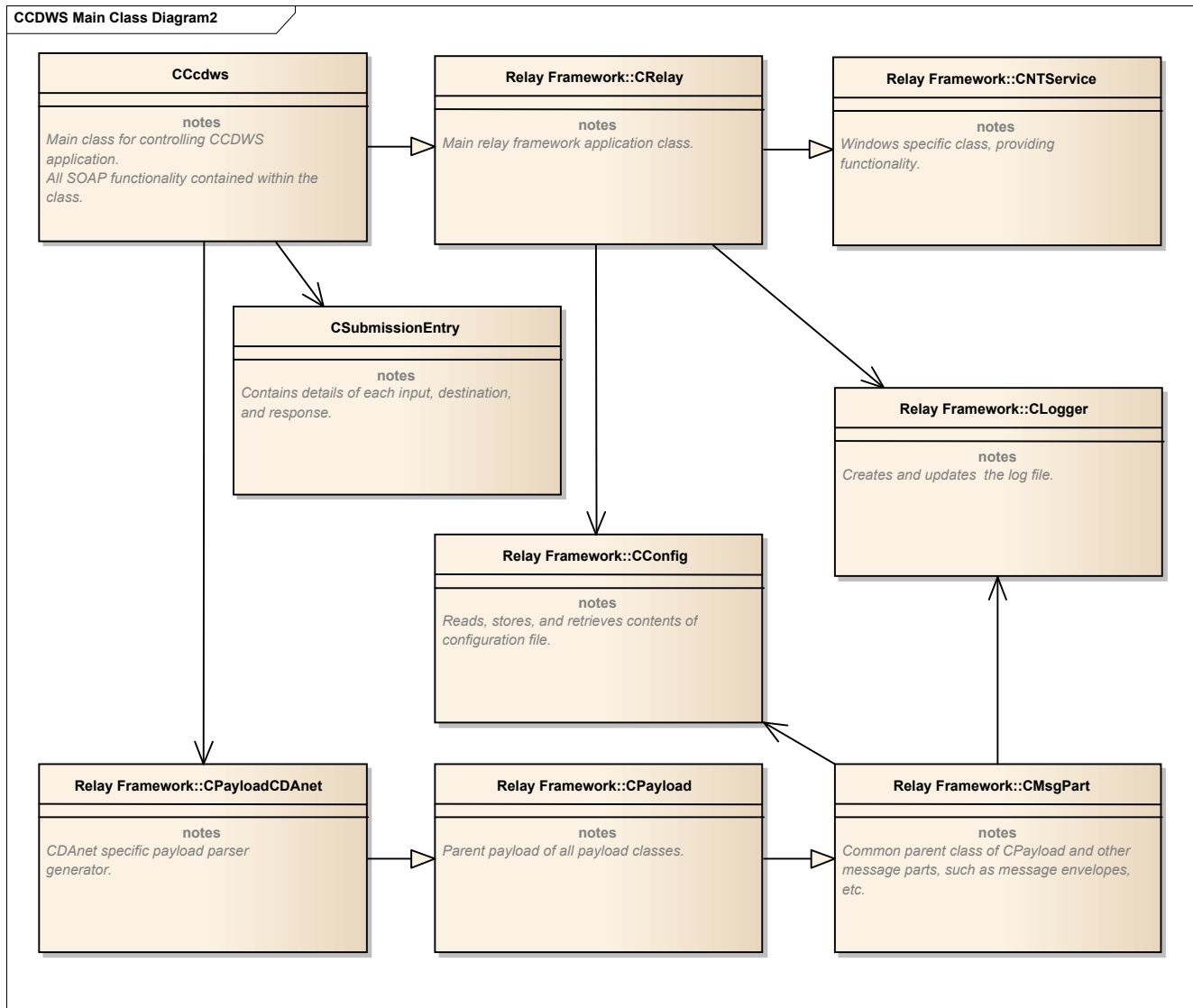
**Contained by:** CCdws

**Referenced by:** CPayloadCDAnet

**Cardinality:** 1

## 6. Class Diagram

The following is a simplified diagram containing the main classes within the CCDWS application. Unfilled arrowheads define associations. Filled arrowheads define generalizations. The classes contained within the relay framework are identified by the “Relay Framework” label prefix. Others are specific to the application.



### 6.1. CCcdws

This is the class that contains the main functional method called `run()`, implementing the CCDWS application logic flow defined in the *Logic Flow* section.

**Specialization of:** CRelay

**Associated with:** CSubmissionEntry

### 6.2. CSubmissionEntry

CSubmissionEntry is a storage class, used to contain data elements associated with a specific submission (input / output). Each submission will have its own entry. All entries are stored in a vector, which will then be spanned

to handle submissions one at a time.

**Associated with:** CCcdws

### **6.3. CRelay**

CRelay is part of the relay framework, and is the class that all implementations using the framework use to create new specialized classes. It contains numerous properties and virtual methods that can be used and overridden to define new classes containing the behaviour desired within an application. Within the CCDWS application it provides the methods to create and control the behaviour of the CConfig and CLogger classes. In the Windows environment, it is a specialization of the CNTService class. The CRelay class is not instantiated directly.

**Specialization of:** NTService (Windows only)

**Generalization of:** CCcdws

**Associated with:** CConfig, CLogger

### **6.4. CNTService**

In the Windows environment, CNTService is a generalization of CRelay, and is contained in the relay framework. This provides the application with the functionality necessary for the application to execute as a Windows service, while still allowing it to execute as a console application. Additionally, it takes command line arguments that will install or uninstall it as a service, and also point to the location of the configuration file to be stored in the system registry so it can be accessed when the service is started. The CNTService class is not instantiated directly.

**Generalization of:** CRelay

### **6.5. CPayloadCDAnet**

CPayloadCDAnet is a payload class within the relay framework that can read, parse, generate, and write CDAnet v2/v3/v4 payloads. Internally, it relies on separate CDAnet message classes that are not shown in this diagram. It is a specialization of the CPayload class.

**Specialization of:** CPayload

### **6.6. CPayload**

The CPayload class is used as a parent for all custom payload classes within the relay framework. It provides little functionality, but acts as the general type for associated classes to store and reference payloads. This class is generated by CCDWS if the data type is not CDAnet.

**Specialization of:** CMsgPart

**Generalization of:** CPayloadCDAnet

### **6.7. CMsgPart**

The CMsgPart class is part of the relay framework, providing all higher level methods to read, write, parse, generate, etc. It provides these to more specific message components, such as CPayload, CEnvelope, etc. The latter is not used within CCDWS, as message enveloping is performed with the gSOAP functions. Additionally, it references CLogger and CConfig classes so that any specialized classes have the ability to read configuration options and write to a log messages. The CMsgPart class is not instantiated directly.

**Generalization of:** CPayload

## 6.8. CLogger

The CLogger class is part of the relay framework, providing text logging functionality. The LogLevel configuration option can be used to control the amount of information written. This is configured automatically by the CRelay class. CLogger can also write to stdout and stderr if configured to do so. This is helpful when running the application in a console. A single instance of CLogger is often shared among calling classes so that multiple objects can write to the same log file.

**Associated with:** CRelay, CMsgPart

## 6.9. CConfig

This class provides configuration functionality within the relay framework. A single instance is often shared among calling classes. It can read a configuration file and store it, so that specific items can be recalled as necessary.

**Associated with:** CRelay, CMsgPart

# 7. Web Service Communications

The CCDWS application is designed to support web service communications. This means that the dental office must be able to access the internet.

Web services transmit XML encoded messages use the Simple Object Access Protocol (SOAP), typically transported using HTTP over TCP/IP in conjunction with other web standards, such as TLS/SSL secure communications. Additionally, web service extensions such as WS-Addressing and WS-Security are used to support secure delivery of messages. Response to the web services are returned similarly.

## 7.1. WSDL File

Specifications for web services are defined using the Web Service Definition Language (WSDL). For the CCDWS, this is a very simple specification as a wrapper for existing CDAnet message formats. It is very tolerant of the actual payload.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns="urn:hiec:v1" xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="urn:hiec:v1" name="HIECService" targetNamespace="urn:hiec:v1">
  <wsdl:types>
    <xs:schema version="1.0" targetNamespace="urn:hiec:v1"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:complexType name="HIECTransaction">
        <xs:all>
          <xs:element name="Format" nillable="false" type="xs:string"/>
          <xs:element name="Value" nillable="false" type="xs:string"/>
        </xs:all>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="HIEC_Input">
    <wsdl:part name="request" type="tns:HIECTransaction"/>
  </wsdl:message>
  <wsdl:message name="HIEC_Output">
    <wsdl:part name="response" type="tns:HIECTransaction"/>
  </wsdl:message>
  <wsdl:portType name="HIECServicePort">
    <wsdl:operation name="HIECService">
      <wsdl:input message="HIEC_Input" xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
wsaw:Action="urn:hiec:v1:HIECService"/>
```

```

        <wsdl:output message="HIEC_Output" xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
wsaw:Action="urn:hiec:v1:HIECServiceResponse"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="HIECServiceSoapHttp" type="HIECServicePort">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsaw:UsingAddressing wsdl:required="true"/>
    <wsdl:operation name="HIECService">
        <soap:operation soapAction="urn:hiec:v1:HIECService"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="HIECService">
    <wsdl:port name="HIECService" binding="HIECServiceSoapHttp">
        <soap:address location="https://hiec.ab.bluecross.ca/P/HIECService"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

## 7.2. Sample Request and Response Messages

The documented entitled *CCDWS – Technical Reference and Operating Manual* contains request and response messages for a sample web service transaction.

# 8. Compiling CCDWS

In general, it should not be necessary to compile CCDWS, as Harder Software maintains the source code and provides updated binaries as necessary. However, the code is available in the case that it is necessary to troubleshoot or otherwise make custom alterations to the code. Note that these changes will not be supported by Harder Software.

## 8.1. Obtaining and Extracting the Source Code

The source code will be available in a zip file through a HIEC affiliate, and will come packaged in a zip file. Once it is available, unzip it under any development path. It will create a directory *ccdws* with various subdirectories.

It will be necessary to obtain a copy of OpenSSL, including Windows binaries. Find the latest copy at <http://www.openssl.org/related/binaries.html>. This will point to a location for obtaining Windows binaries. Ensure that the download file also contains the source. Once downloaded, extract and move the main directory to a subdirectory *openssl* under the *ccdws* directory. Ensure that the static libraries are available. These should exist in the *openssl/out32* directory.

## 8.2. Development Environment

CCDWS was compiled for the Windows environment using Visual Studio 2010. You should have this or a more recent version of Visual Studio available.

## 8.3. Compiling

To compile CCDWS, use Visual Studio to open the *ccdws.sln* solution file. Ensure that the 32-bit Release configuration is the default, and perform a build. There should be no errors or warnings.

If a failure occurs finding the OpenSSL include or library files, ensure that it is properly downloaded and extracted to the *openssl* subdirectory. It may be necessary to alter a link input location or other related configurations to match the structure of the *openssl* directories.

Once CCDWS is successfully compiled, it should be available in the *Release\Win32* subdirectory.