HIPAAsuite RealTime Server

HIPAAsuite RealTime Server

Printed: September 2016 in (whereever you are located)

Publisher

HIPAAsuite

Managing Editor

Martin Scholl

Technical Editors

Juan Rivera

Cover Designer

T3 Design, Alameda, CA

Table of Contents

1 Introduction

1.1 The HIPAAsuite RealTime Server

What is the HIPAAsuite RealTime Server

Since the introduction of EDI (Electronic Data Interchange)transactions into healthcare with the ratification of the HIPAA (Health Insurance Portability and Accountability Act) law in 2003, an ever increasing part of the healthcare transactions between providers and payers is done using EDI transactions. This technology proved itself to reduce the costs of health care. It costs today only a fraction to file an electronic claim versus printing a claim form to paper and sending it to a the payer by mail.

EDI categorizes transactions between business partners into so-called "transaction sets". All expected content of such a transaction is standardized and precisely defined. The purpose is the fast and efficient communication by different computer systems without the interaction of human beings. Therefore, such transactions have to be flawless and predictable in composition. The National Institute of Standards (NIS) developed the X12 standards and through the workgroup of EDI monitors their guidelines. EDI is found in all sectors of American business, be it banking, retail, cellular phone billing, and many others.

Twelve initial sets of transactions were written into the 1996 HIPAA law. It was understood that this group of transactions was not complete and meant to be amended and further developed. The HIPAA EDI messages are designed to group transaction types by their identifiers. Transactions that start with the number '2'. are considered Real-Time transactions and meant to be answered in a short amount of time. Under HIPAA these are the 270/271 Eligibility and the 276/277 claim status transactions as well as the 278 Authorization and Review Request transaction. Until now the implementation of these transactions in production environments have been very spotty. Providers' practice management software systems and insurance carriers' EDI capabilities were often not able to create those response transactions within the 20 seconds that the standard identifies as Real-Time.

The File Transfer Protocol (FTP) that makes the back bone of the today's existing HIPAA EDI infrastructure is innately not designed to operate in Real-Time mode. Clearly, a different transport mechanism had to be found and many providers and payers worked on different methods to make it happen. An industry group, the Council for Affordable Quality Healthcare (CAQH), developed a precise standard to implement a workable and universally shared technology under the acronym of "CORE" (Committee on Operating Rules for information Exchange). The most promising technologies that were tried and tested are:

- **MIME**, an email-like transmission protocol, where different data types can be

attached and transported, and

- **SOAP**, an XML based transfer mechanism that is increasingly used to establish computer to computer communications.

Those two standards are implemented in the CORE Phase I and II protocols. These protocols have been adopted as part of the ACA, the Affordable Care Act, also known as Obama Care. Payers are now obligated to provide Real-Time interfaces for claim status, eligibility and payment advice.

HIPAAsuite decided that this complex technological challenge needed a simple and cost-effective solution. In 2013 we started with the layout and process analysis and in 2014 did we decide to earnestly invest time and resources to develop the HIPAAsuite RealTime Server.

HIPAAsuite has applications that automatically create responses to 270 and 276 request files in batch mode, which means requests were delivered and a response was given in 24 hours, often less, to be picked up or transferred via FTP.

The HIPAA Claim Status Responder and the HIPAA Eligibility Responder can be used to automatically create the 271 or 277 response files in batch mode. And now, with the HIPAA RealTime Server, these applications together provide a CORE Phase I and II certified solution.

The HIPAAsuite RealTime Server implements both a MIME and a SOAP web server which interacts with these two applications to provide a valid response file using the appropriate protocol in less than 20 seconds.

MIME and SOAP are complex protocols. It took us many months to fully understand the logic and flow of data files. The following documentation is written with the intent to educate the reader on the back ground as much as possible to understand and appreciate the complexities involved.

1.2 CORE

The Committee on Operating Rules for Information Exchange (CORE) was proposed by CAQH (the Council for Affordable Quality Healthcare) as an initiative to develop a solution that enables consistent provider access to healthcare administrative information before or at the time of service using their choice of electronic system. CAQH's mission is to drive the creation and adoption of healthcare operating rules for administrative transactions. It has brought together over 140 healthcare industry stakeholders working to facilitate patient insurance information to physicians and hospitals, thereby significantly improving insurance verification and promoting health plan-provider interoperability.

Founded in 2005, CAQH is an American non-profit organization based in Washington, DC that collaborates with healthcare providers, trade associations, and insurers; they create shared initiatives to streamline business in healthcare. Among these initiatives are CAQH CORE, which maximizes business efficiency and savings by developing and implementing federally mandated operating rules. They also provide the CAQH Index, which benchmarks progress and helps to optimize operations by tracking industry adoption of electronic administrative transactions.

CAQH's mission statement is: "To accelerate the transformation of business processes in healthcare through collaboration, innovation and a commitment to ensure value across stakeholders." CAQH CORE certifies and awards CORE Certification Seals to entities that create, transmit, or use the healthcare administrative and financial transactions addressed by the CAQH CORE Operating Rules. CORE Certification means an entity has demonstrated that its IT system, or product, is operating in conformance with applicable requirements of a specific phase of the CAQH CORE Operating Rules. HIPAAsuite obtained the CORE Phase I and Phase II certification for the HIPAA RealTime Server in January 2015.

Members of the CAQH non-profit alliance include Aetna, Anthem, America's Health Insurance Plans, AultCare, the Blue Cross Blue Shield Association, Blue Cross Blue Shield of Michigan, Blue Cross Blue Shield of North Carolina, Blue Cross Blue Shield of Tennessee, CareFirst Blue Cross Blue Shield, Cigna, Health Net, Horizon BlueCross Blue Shield of New Jersey, Kaiser Permanente and UnitedHealth Group.

The availability of information in real-time at the point of care can reduce medical errors, allow physicians and their patients to make informed decisions about treatment options and reduce administrative burdens. The challenges are equally well understood. Technology adoption rates, data security, and inconsistency associated with transactions and interactions between stakeholders limit the ability to realize a complete solution. FTP, a network protocol widely used in the healthcare industry for EDI transfers, is not a real time protocol by design, requiring unnecessary overhead; resources dedicated to

providing real time EDI transfer capability using FTP may be better served using a truly real time standard.

Two envelope standards (HTTP MIME Multipart and SOAP+WSDL) were selected by the CORE Phase II Connectivity & Security Subgroup from the initial long list of standards. They were shown to meet the CORE Phase II Connectivity criteria, have significant installed base in this industry, and perform well under real world transaction loads.

Since both these standards have significant merits, the Subgroup debated the advantages and challenges of having a single envelope standard versus both these envelope standards as part of the CORE Phase II Rule and Safe Harbor provisions. The major advantage of a rule based on a single envelope standard is that it would be more definitive and facilitate better interoperability. However, having just one standard would require implementers of the other envelope standard (i.e., the one that was not chosen) to modify their implementations to be CORE Phase II-compliant. Since both standards met the criteria and have large installed bases, convergence on a single standard would create a barrier to adoption of CORE Phase II Connectivity Rule by a large segment of the industry.

In the interest of further facilitating interoperability, CORE expects to move towards a single envelope standard in future phases. Given the current state of healthcare connectivity (i.e., use of many distinct connectivity methods), creating a CORE Phase II Connectivity rule with two envelope methods vastly improves the state of the market, while also providing an opportunity for education and greater experience with two standards that meet the growing market needs for connectivity. Taking this phased step enables the healthcare industry to make a more informed decision as it considers supporting a single envelope.

CORE operating rules are streamlining eligibility, benefits, and claims data by allowing providers to submit a request, using the electronic system of their choice to obtain a variety of coverage information for any insured patient and from any participating health plan. Providers will receive more consistent and predictable data, regardless of health plan.

CORE certification is proof that all CORE Rules and expected scenarios are met and requires an entity to perform testing of the relevant phase of the CORE Certification Test Suite developed by the CORE Testing Subgroup with a CORE-authorized Testing Vendor.

CORE Operating Rules

The Patient Protection and Affordable Care Act (ACA) defines operating rules as, “the necessary business rules and guidelines for the electronic exchange of information that

are not defined by a standard or its implementation specifications."

ACA Section 1104 applies to HIPAA covered entities and business associates engaging in HIPAA standard transactions on behalf of covered entities. The legislation requires that the standards and associated operating rules adopted by the Secretary will:

- Enable the determination of an individual's eligibility and financial responsibility for specific services prior to or at the point of care;
- Be comprehensive, requiring minimal augmentation by paper or other communications;
- Provide for timely acknowledgment, response, and status reporting that supports a transparent claims and denial management process (including adjudication and appeals);
- Describe all data elements (including reason and remark codes) in unambiguous terms, require that such data elements be required or conditioned upon set values in other fields, and prohibit additional conditions (except where necessary to implement State or Federal law, or to protect against fraud and abuse).

Section 1104 of the Patient Protection and Affordable Care Act (ACA) mandates this certification process for health plans only.

Per the ACA, health plans must file a statement with the department of Health and Human Services (HHS), in such form as the Secretary may require, certifying that their data and information systems are in compliance with any applicable transaction standards and associated operating rules; financial penalties for health plans are significant.

The adoption deadline for Eligibility and Claim Status EDI transactions (included in Phase I and Phase II of the CORE Operating Rules) was 1/1/2013, so far the HHS has not enforced this deadline with penalties and affected entities are still in the implementation phase.

On December 31, 2013, HHS issued a Notice of Proposed Rulemaking (NPRM) on the ACA-mandated health plan certification. The Department of Health and Human Services (HHS) accepted public comments on the NPRM through April 3, 2014 (previously March 3, 2014). The NPRM includes health plan certification requirements for the eligibility, claim status, electronic funds transfers (EFT), and electronic remittance advice (ERA) transactions. It defines two potential certification options for plans to meet with HHS compliance requirements:

- Option 1: HIPAA Credential: Under the HIPAA credential program it is proposed that health plans will attest to compliance with the HIPAA-mandated transaction standards and operating rules. Once HHS issues the Final Rule later in 2014, CAQH CORE, as the proposed administrator, would offer the ability to complete the necessary HIPAA Credential documentation.

- Option 2: CORE Certification: The NPRM proposes to adopt the existing CORE Certification Program, authored and administered by CAQH CORE. Health plans that successfully complete certification testing with a CORE-authorized testing vendor and submit the required documentation will receive a Phase III CORE Certification Seal demonstrating their compliance.

CORE Operating rules are divided into Phases I, II, and III. This product is compliant with Phases I and II; they are explained below.

Phase I

Phase I CORE Operating Rules apply only to ASC X12 005010X279A1 Eligibility and Benefit Request and Response (270/271) transactions; DDE (Direct Data Entry) transactions and web-based transactions are not a part of the Phase I scope.

Phase II

Phase II CORE Operating Rules apply to ASC X12 005010X279A1 Eligibility and Benefit Request and Response (270/271) and ASC X12 005010X212 Health Care Claim Status Request and Response (276/277) implementation guides; DDE (Direct Data Entry) transactions and web-based transactions are not part of the Phase II scope.

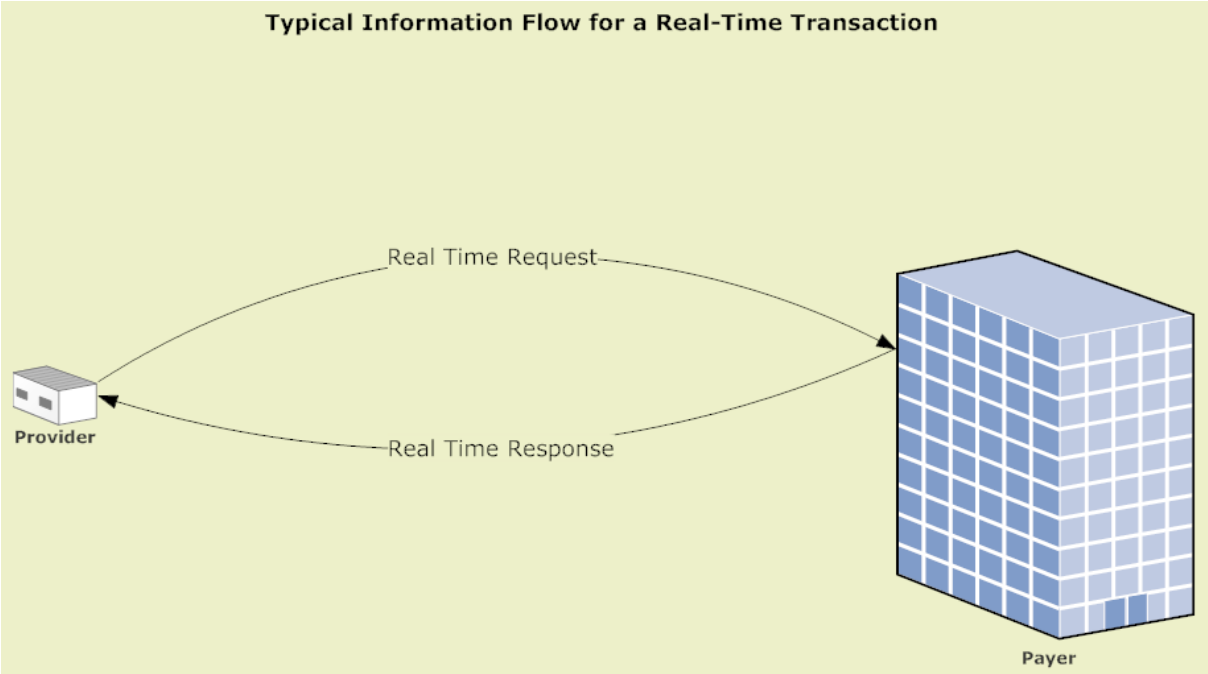
1.3 Real-Time and Batch

CORE regulations cover two process modes: Real-time and Batch.

Real-time transactions in EDI usually start with the number 2. For example 270/271 for eligibility, 276/277 for claim status and 278 for Authorizations in the health care industry. The prevalent transport mechanism in healthcare EDI is at this time the File Transfer Protocol (FTP). This specific internet protocol is not suitable for real-time transactions since it does not provide for a response. CORE defines real-time as less than 20 seconds and this is not achievable with FTP and the reason why the committee chose MIME and SOAP as transport protocols. Both of those protocols do exactly the same thing and do not influence the information flow.

There are two basic modes of transport defined in the CORE standard, true real-time transactions and batch transactions. We will explain the difference below.

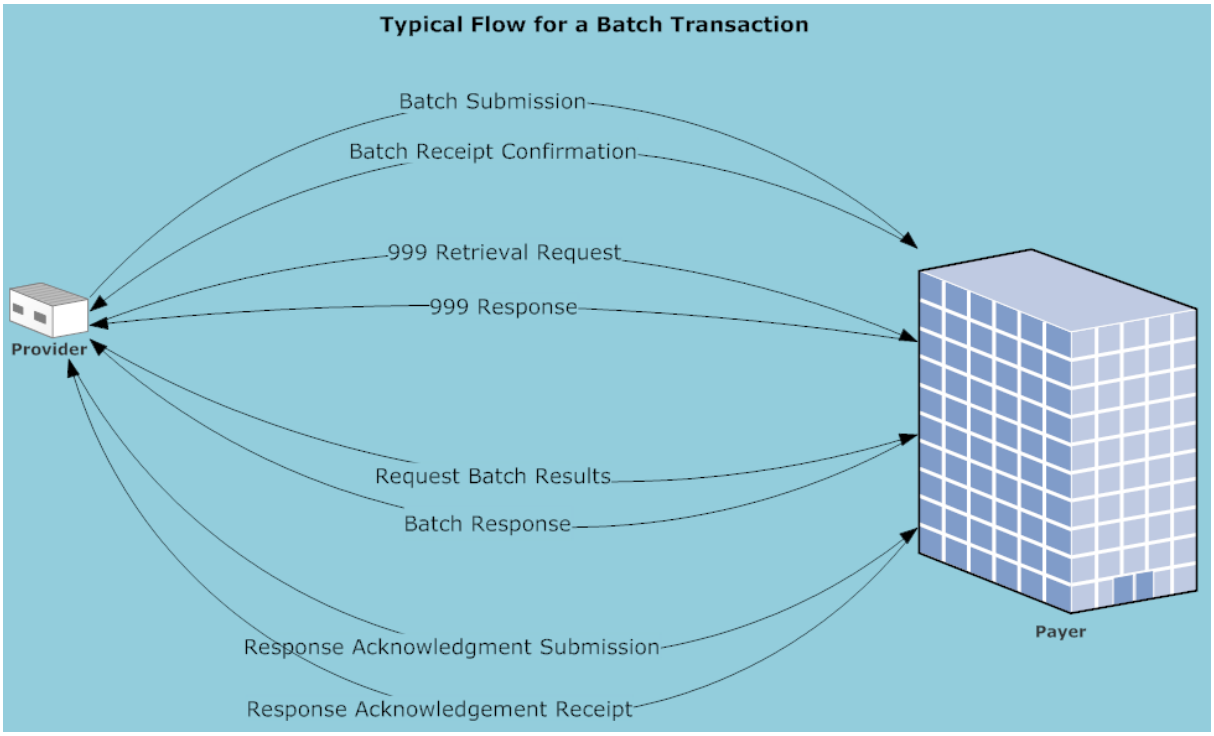
The following graphic shows the data flow in a typical real-time transaction



Data flow in a real-time transaction

This transaction is really simple, a request is sent out and a response is received back. No extra bit of information exchange is required.

Compare this to the composition of a batch transaction in CORE. Here we distinguish between four distinct information exchanges



Data Flow in a batch transaction

- First is the batch submission. Here the payload consisting of multiple requests is transmitted and a confirmation receipt is sent back.
- Second, we have the retrieval of the functional acknowledgment, the 999 transaction which in more detail reports if all individual transaction sets have been accepted and/or how many have been rejected.
- Third, we the response retrieval. A request for the response file is sent and the payer sends the response transaction(s) back to the provider.
- Fourth, the provider acknowledges the receipt of the Batch Response with a 999 file that the payer receives and acknowledges.

We can see that the batch process is a lot more complex than the real-time process which is really compressed for the most efficient exchange of information. Also the time frame for the 4 part transactions is spread out. It is usually defined in the trading partner agreement how much time has to pass before the request for the acknowledgment and lastly the request for response retrieval can be sent. Usually an over-night period is required.

The HIPAAsuite RealTime Server supports both transmission modes. It reads and understands the envelope of the transaction and chooses the correct mode.

1.4 HIPAAsuite RealTime SERVER

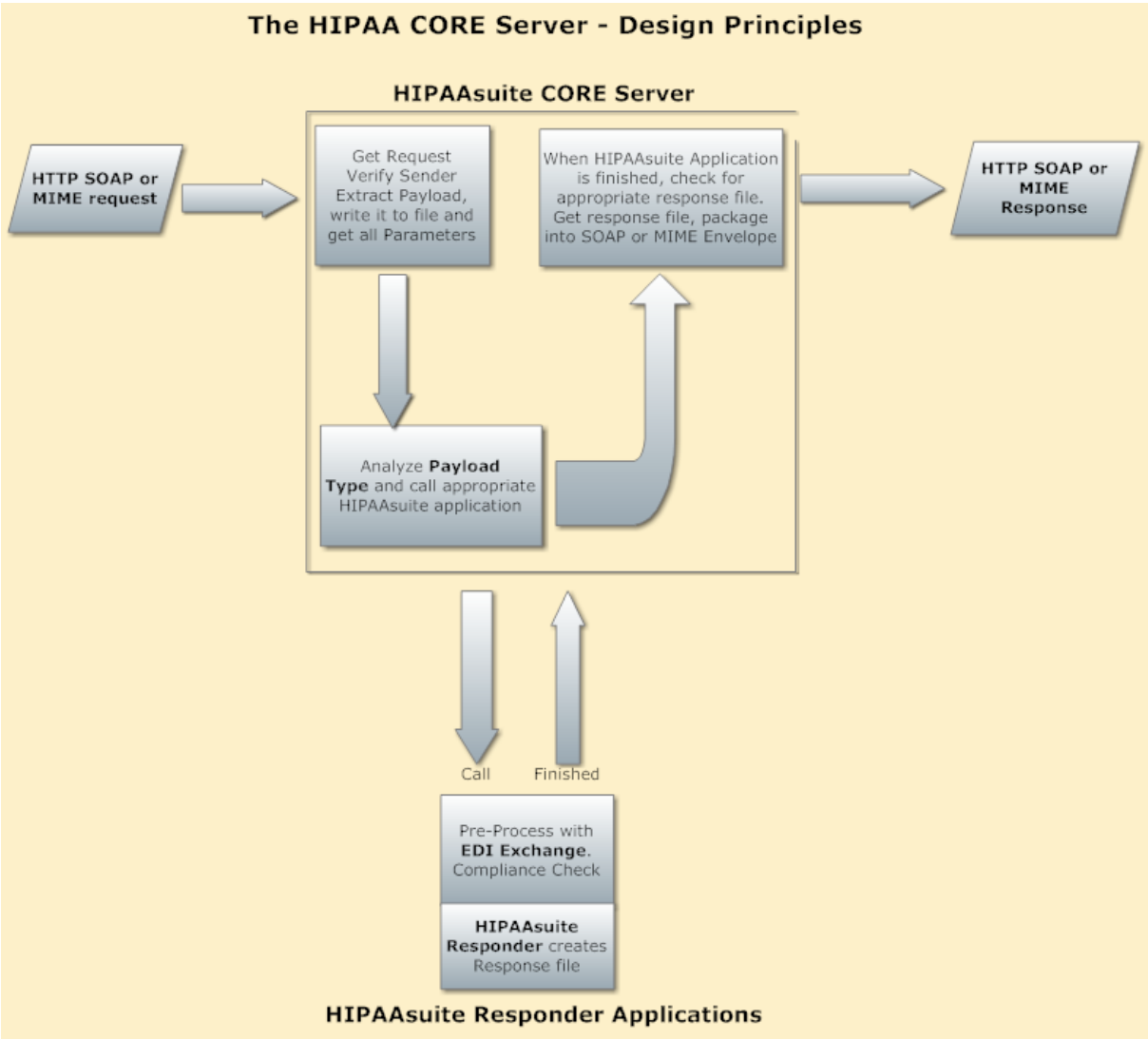
The *HIPAAsuite RealTime Server* is a is CAQH CORE Phase I and Phase II compliant EDI transfer host. It is meant to be used in conjunction with HIPAAsuite's *HIPAA Eligibility Responder*, *HIPAA Claim Status Responder*, and *EDI Exchange* applications.

HIPAAsuite RealTime Server will accept any incoming CORE compliant MIME or SOAP messages and extract the payload, which are HIPAA X12 005010X279A1 Eligibility and Benefit Request (270) or HIPAA X12 005010X212 Health Care Claim Status Request (276) transaction sets.

It will then relay the payload to the appropriate HIPAAsuite Responder application, which will parse and process the EDI file and generate responses and/or acknowledgments (999), as appropriate.

The *HIPAAsuite RealTime Server* will then (in the case of a real time transaction) wrap the response, or acknowledgment EDI file in the case of a failed EDI validation for the incoming request, and return it to the receiver or (in the case of a batch mode transaction) wait for an incoming response request or acknowledgment request to send back the appropriate EDI file.

HIPAAsuite RealTime Server is able to parse and respond to both request types (RealTime or Batch) wrapped in either CORE specified envelope (MIME or SOAP).



Basic design of the HIPAAsuite RealTime Server

On incoming requests, the *HIPAAsuite RealTime Server* will initially validate the sender; *HIPAAsuite RealTime Server* uses username/password authentication to validate trading partners. One might think that this is insecure to transmit a user's name and pass word over the internet, but since the transmissions itself is encrypted through the https standard, it is deemed save. We will soon implement a certificate-bound user verification with message level encryption.

HIPAAsuite RealTime Server is a port listener. This means that it constantly watches out for incoming communication attempts in the MIME or SOAP standards. Once it receives a request, the incoming message is parsed for metadata which is contained within the SOAP or MIME envelope and the payload which is extracted and saved. Afterward, *HIPAAsuite RealTime Server* calls the appropriate HIPAA Suite software to process the file and to generate the results which are either returned in real time or stored for later

transmission as part of a batch transaction. *HIPAAsuite RealTime Server*'s communications with the HIPAA Suite Responders is completely compartmentalized and will function so long as you have installed the appropriate HIPAAsuite responder application to deal with the expected incoming EDI requests; i.e., HIPAA Claim Status Responder is not required to respond to 270 Eligibility and Benefit requests and vice versa. The *HIPAAsuite RealTime Server* requires and builds upon the HIPAAsuite EDI Exchange application.

In addition to the CORE Phase I and II transactions, we built the HIPAAsuite RealTime Server to accept all HIPAA transactions. All 12 HIPAA transactions can be transmitted with HIPAAsuite RealTime Server and passed on to the appropriate HIPAAsuite Software application. For example, 837 claim files could be transmitted and passed on to the HIPAA Claim Master. We believe that SOAP and MIME can replace FTP as the preferred transport method of HIPAA EDI files and we designed HIPAAsuite RealTime Server accordingly.

HIPAAsuite RealTime Server is built on the Windows Communication Foundation (WCF) and written entirely in C#. WCF provides tools to implement and deploy a service-oriented architecture, ideal for distributed computing where services have multiple remote consumers; the clients themselves can also consume services from multiple servers. This makes WCF a great tool for building EDI exchange services and consumers, especially so given the default message envelopes are SOAP in WCF. WCF builds and publishes a WSDL (Web Services Description Language) file that can be accessed by appending "?WSDL" to the service's address. A service's WSDL provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns. Given that all CORE-compliant services and servers will be using the same set of EDI transactions, any WSDL describing a CORE-Compliant service will describe all CORE-Compliant services.

WCF clients connect to a service using a communication endpoint, a discoverable interface that separates the means of communication from the communication subsystems. Through these endpoints a layer of abstraction is created between the WCF environment and any client or consumed service. HIPAAsuite RealTime Server uses four endpoints in all, one each for the real time and batch modes for both SOAP and MIME envelopes. Since MIME real time and batch envelopes use a custom parser, both may use the same address. SOAP real time and batch, however, may not. Custom endpoints are used to handle SOAP messages for greater control over the SOAP headers used for authentication purposes; SOAP real time and SOAP batch use different envelopes, although both may be called SOAP. The SOAP batch envelope is encoded with MTOM (Message Transmission Optimization Mechanism), a SOAP optimization feature for sending and receiving large payloads which requires the SOAP message to be sent within a MIME envelope.

For ease of installation, HIPAAsuite RealTime Server is self-hosted, which means it uses any host computer as a server and does not require additional web services or servers like Microsoft Internet Information Services (IIS) or Apache. An early prototype was built as a local windows service, but was discarded to avoid the complexity of configuring access to a service which does not inherently run in the context of any particular user that would need administrative permissions to perform functions like reserving ports for the hosting of endpoints. As a self-hosted application, HIPAAsuite RealTime Server can be run easily with administrative credentials without the need for additional setup steps to configure the properties of a service or set up a web service in IIS. All HIPAAsuite RealTime Server needs are the HIPAAsuite responders, EDI Exchange, and the setup on the program itself.

HIPAAsuite RealTime Server is a formless windows form application. This means it is an application without an initial or main form to present an interface to the user; it remains running in the background. HIPAAsuite RealTime Server does present a notification icon when running, from which it can be configured. HIPAAsuite RealTime Server also has an optional debug console from which all incoming and outgoing EDI transactions are displayed along with several useful process indicators. This model was selected for HIPAAsuite RealTime Server because it is cuts back on resource overhead and is non-intrusive. The only configuration file it draws from stores user-specified options; unlike most web services it uses no webconfig file, all information regarding hosts, endpoints, behaviors, and contracts is set programmatically according to the user's preference for configuration on the fly. An option to open a test host on a port separate from the default secure host is provided and is the only occasion in which an unsecured host will expose an endpoint. This unsecured host coupled with the debug console make a useful and indispensable tool for inspecting incoming and outgoing envelopes in a test environment or, indeed, testing for CORE compliance.

In the following sections you will learn how to install, configure, and utilize HIPAAsuite RealTime Server.

1.5 Requirements

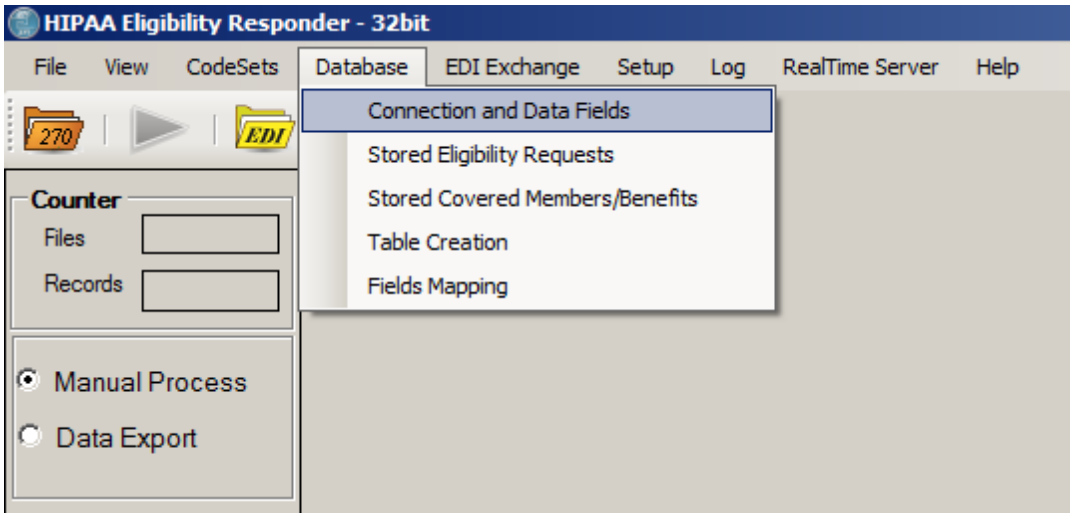
1.5.1 HIPAA Eligibility Responder

In order to receive and respond to ASC X12 005010X279A1 Eligibility and Benefit Requests (270) you must have installed HIPAA Eligibility Responder. Please download and install this software from HIPAAsuite and configure it as follows. The program itself comes with a detailed help documentation. This is a quick start guide.

Database Configuration

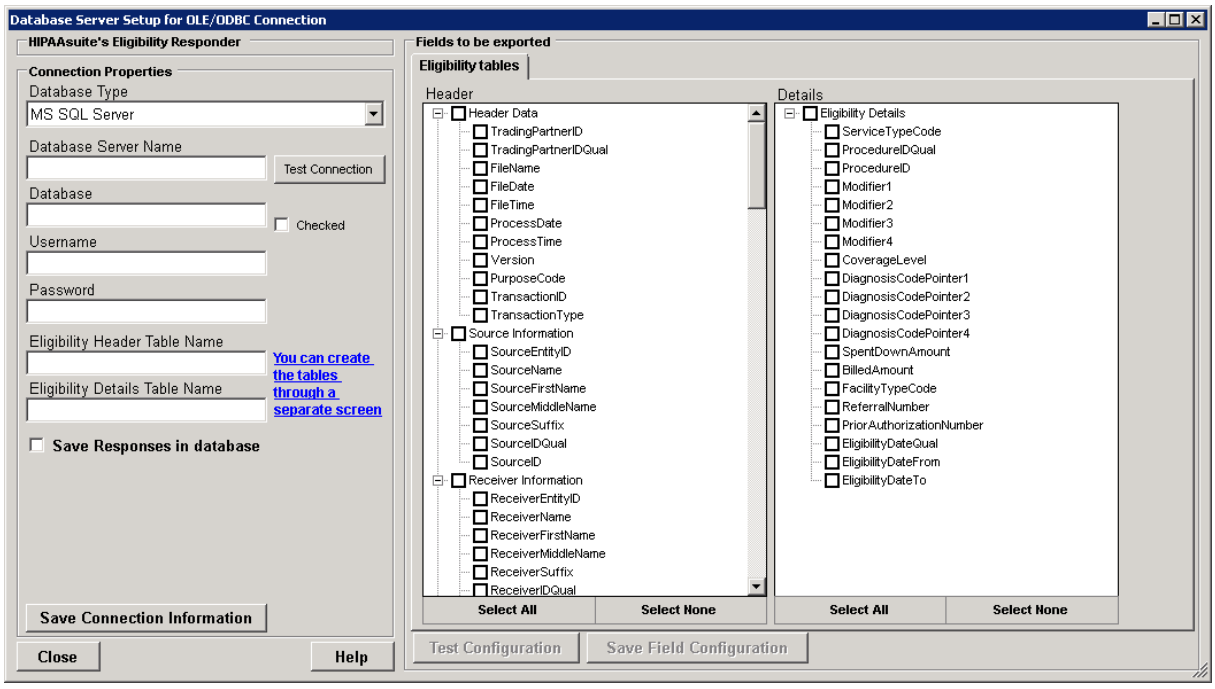
First is the Database configuration. This is where transactions will be stored.

In the Database dropdown menu, select "Connection and Data Fields".



The Database dropdown menu will present options specific to the Database it will be checking for eligibility and patients. The Database Connection and Data Fields option will lead to a form where all necessary Database settings can be configured.

Fill in your database connection details.



This form stores all necessary Database settings.

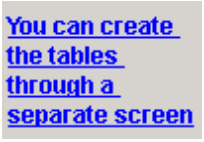
Fill all fields and be sure to check "Save Responses in database". This presents additional tables necessary for correct response behavior.

Click the "Test Connection" button to verify your database connection details are correct. If you have not created the database or tables a pop-up message will inform you the

database could not be opened or the tables were not found.

If a database does not exist, consult your local Administrator.

If you have not yet created the tables, you may click the blue underlined text "You can create the tables through a separate screen" to open a window that will automatically create them for you.



This text links to the table creator.

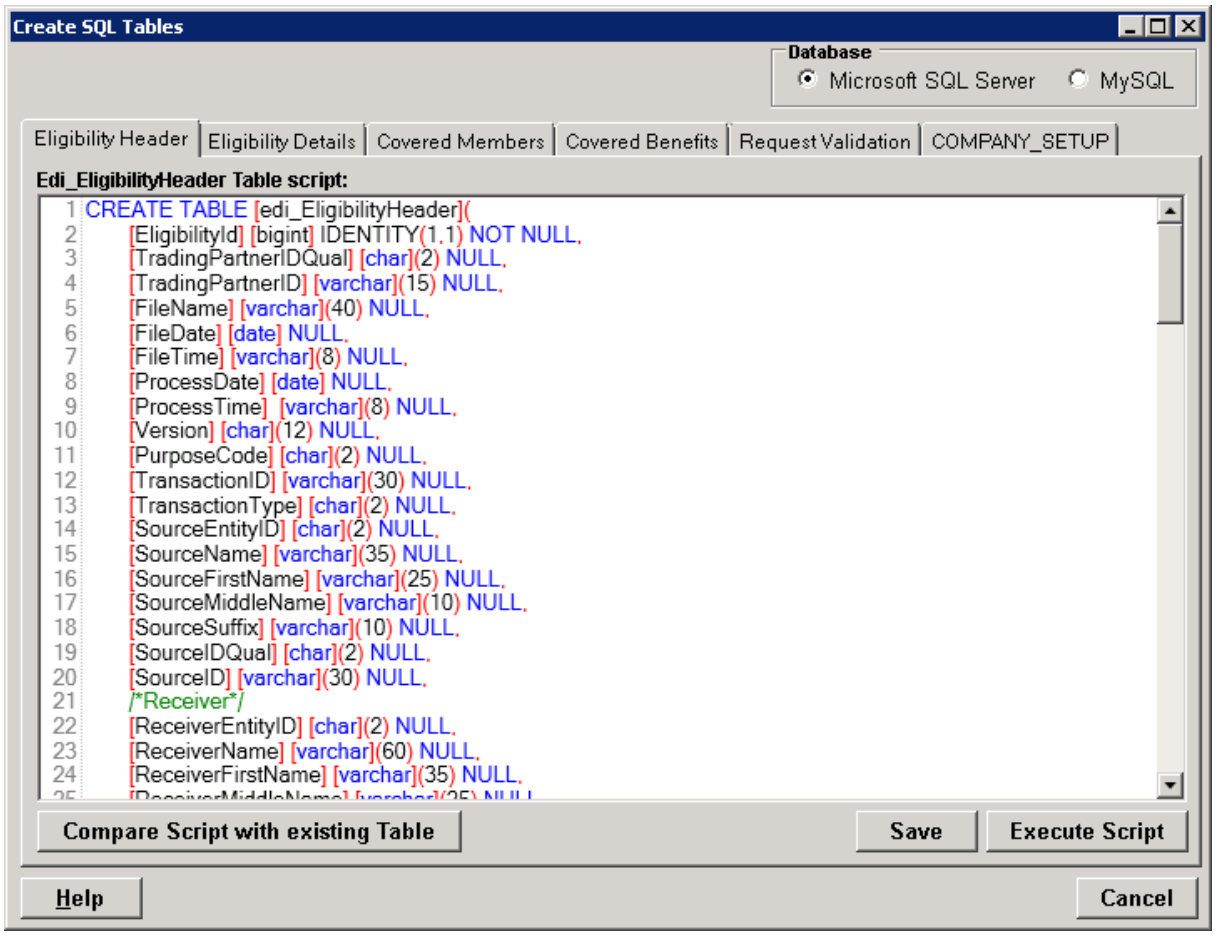
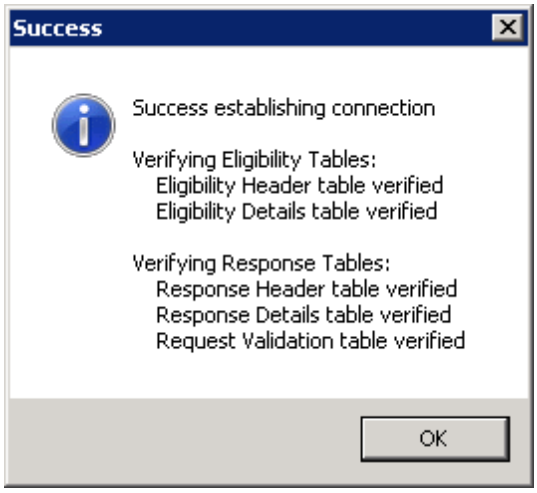


Table creation made simple.

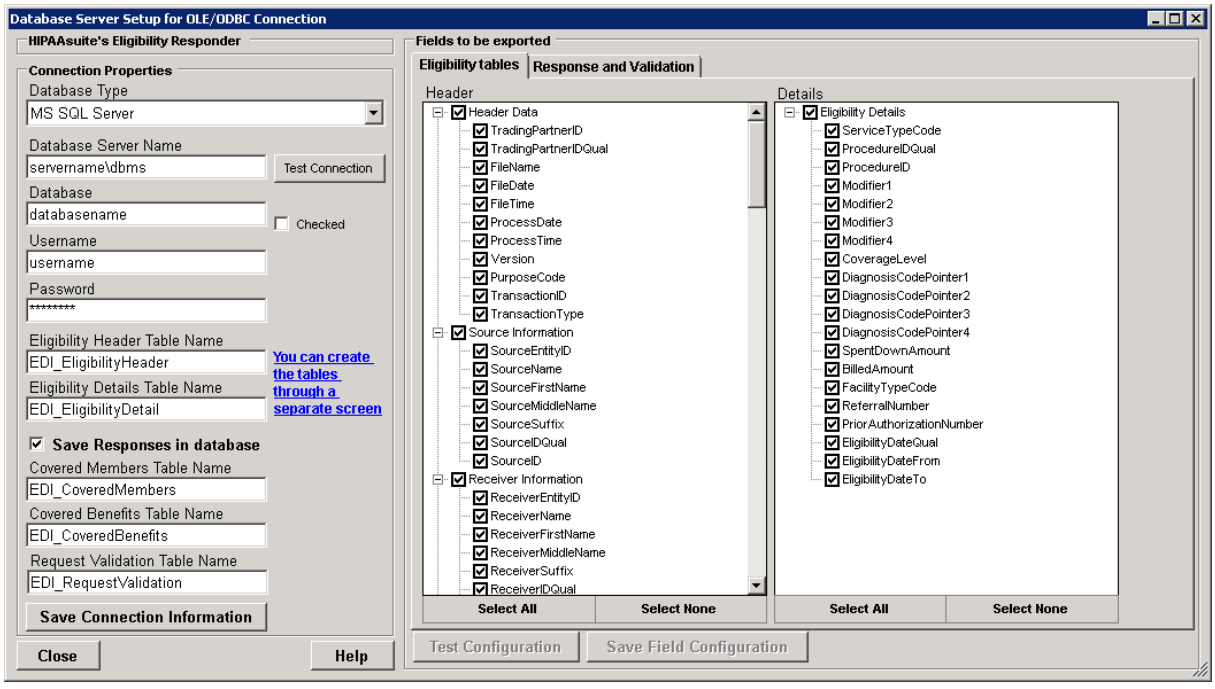
Testing your database connection with the "Test Connection" button should now yield the following message:



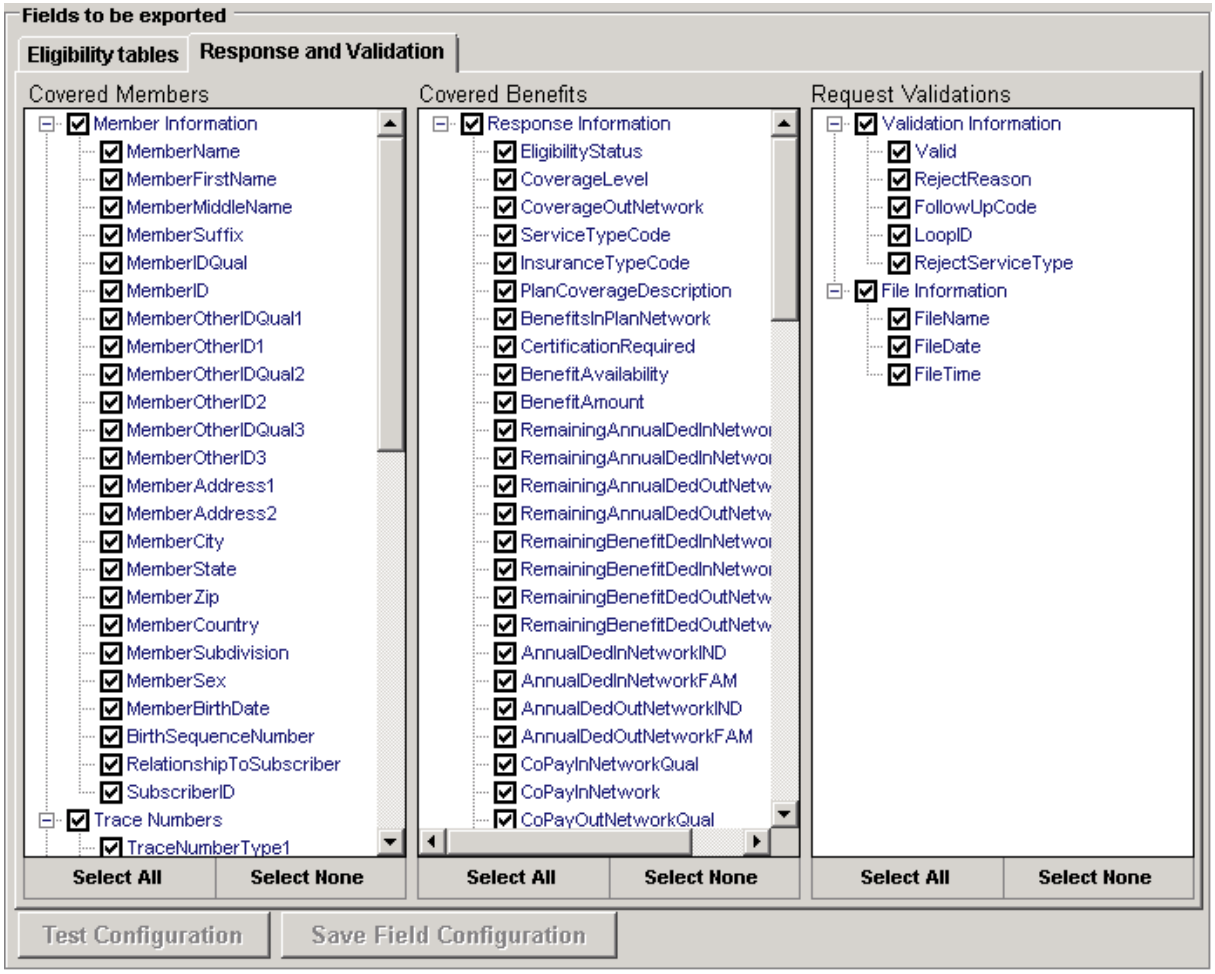
Connection test successful.

All six tables are necessary. If there are any missing tabs, go back to the Database Connection and Data Fields menu and verify "Save Responses in database" under "Connection Properties" is checked.

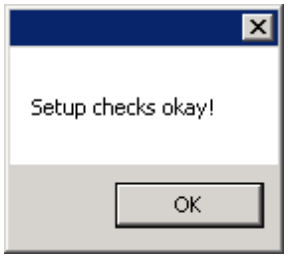
After creating the tables, the field configuration must be tested. In the Database Connection and Data Fields menu make sure all fields are selected for both tabs. Click the "Select All" button near the bottom of the form to check all fields. Then, click "Test Configuration", and later the "Save Field Configuration" button.



"Select All" buttons near the bottom make field selection quick.



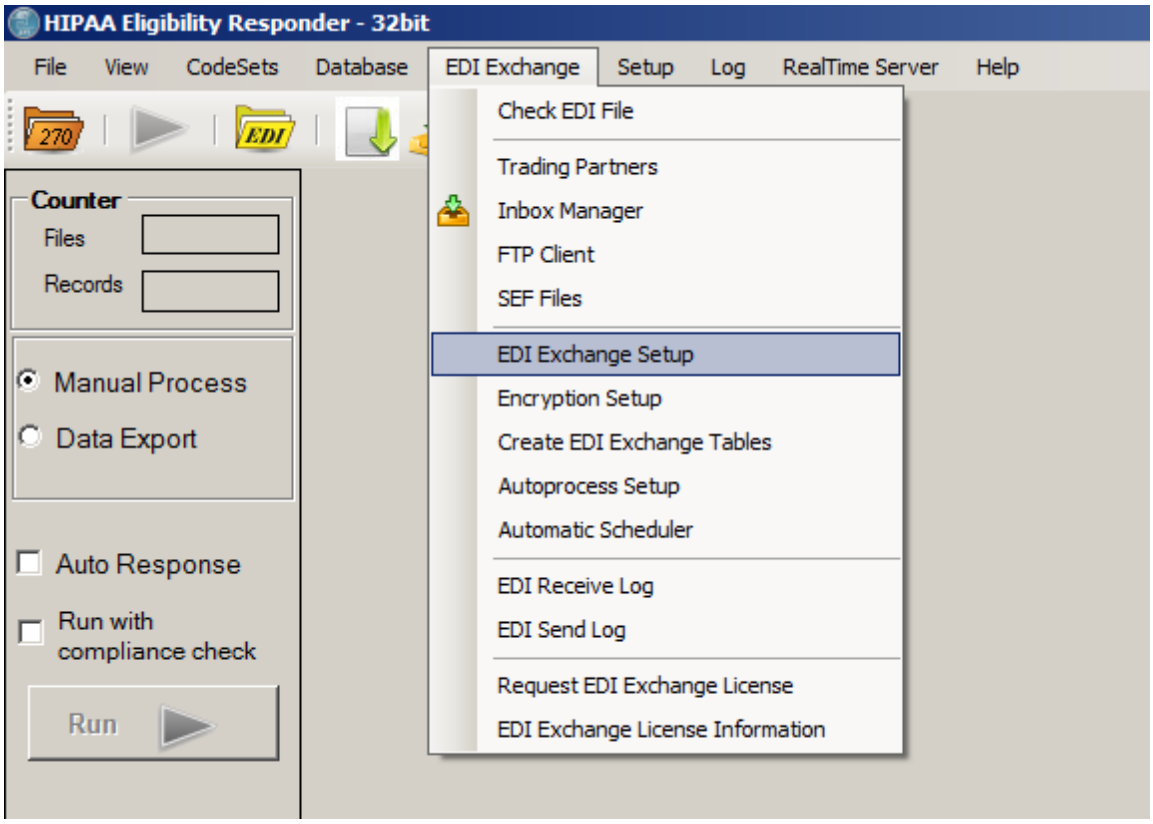
The Response and Validation fields are also necessary.



Setup successful.

The Database is now configured and HIPAA Eligibility Responder can generate Eligibility and Benefit responses. Follow the next steps to automate this process.

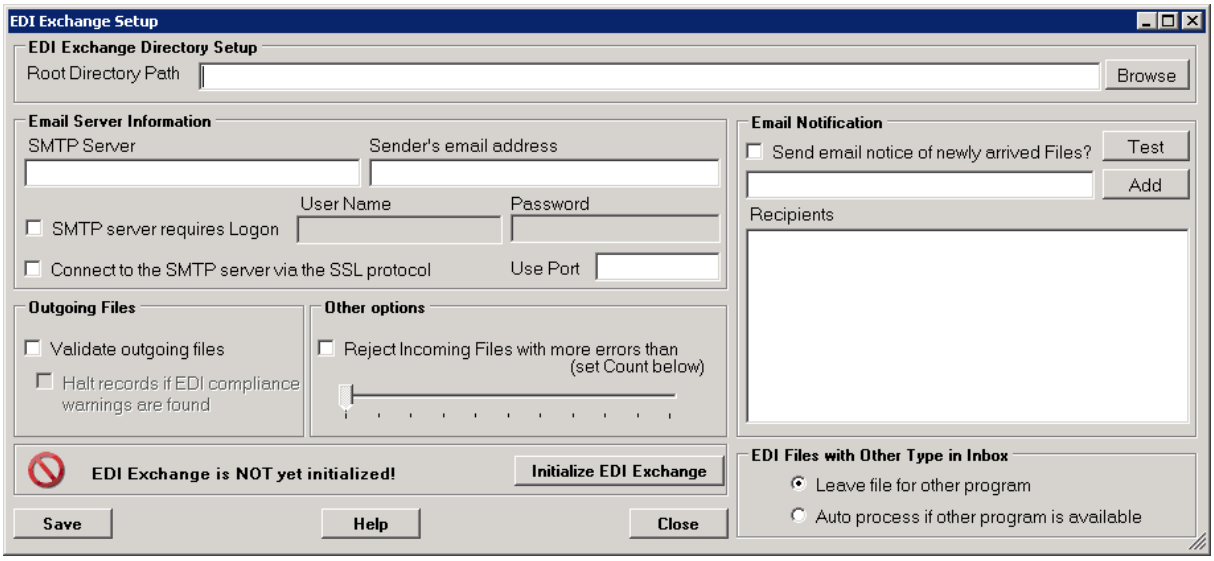
In the EDI Exchange dropdown menu, select "EDI Exchange Setup".



EDI Exchange Setup menu option.

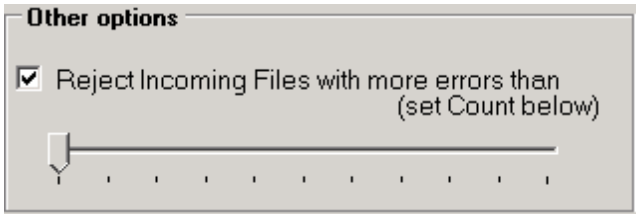
The following window contains all necessary information to initialize EDI Exchange and automate the Eligibility and Benefit request/response process.

Specify your Root Directory Path. This is where all incoming, outgoing, suspended, and rejected 270/271 files will be stored along with their respective acknowledgment EDI files.



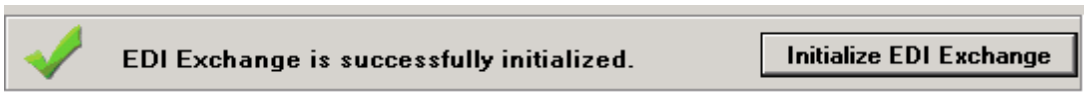
EDI Exchange setup window.

For COREII compliance, bad incoming EDI files must be rejected. Check "Reject Incoming Files..."



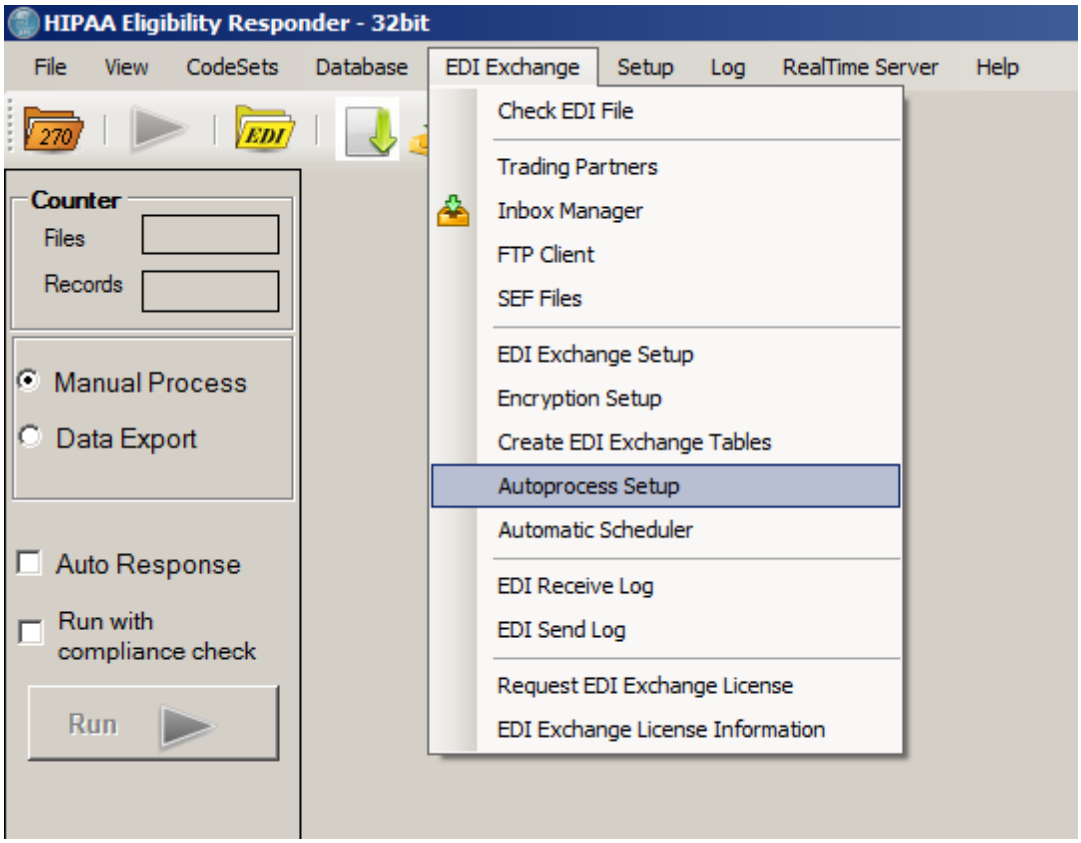
Reject files with errors.

Click "Initialize EDI Exchange". This will check your database connections and create all necessary Inbox subdirectories. Afterward, it should look like this:



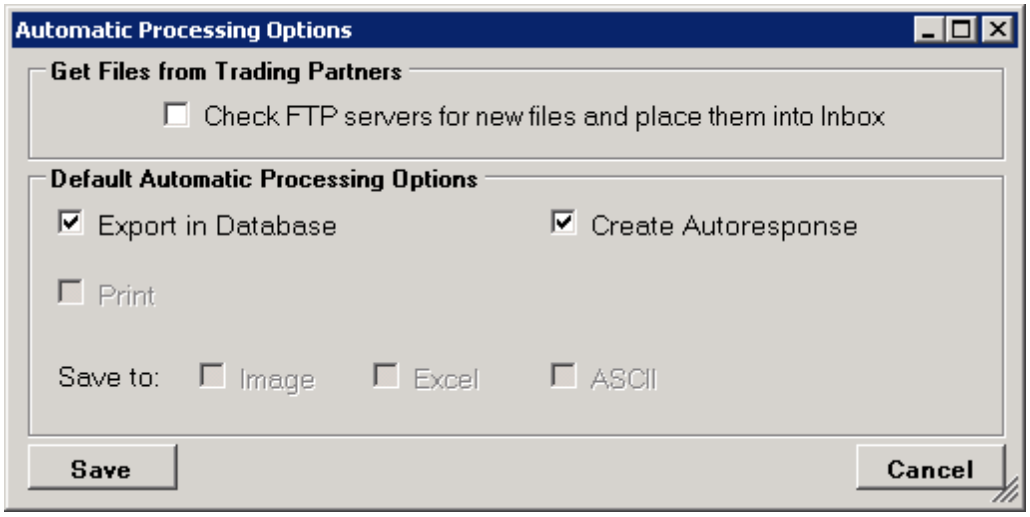
Successful initialization.

Finally, the autoprocessing options need to be set. In the EDI Exchange dropdown menu, select "Autoprocess Setup".



Autoprocess setup menu option.

In the following window, check options "Export in Database" and "Create Autoresponse."



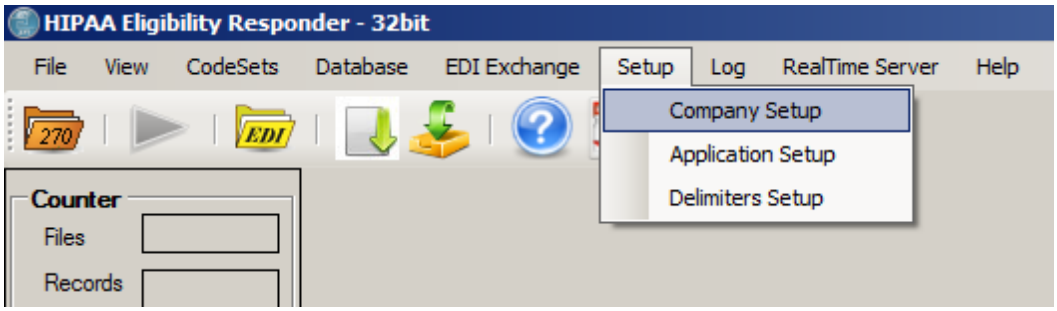
DB Export and Autoresponse checked.

Company Setup

Now, Company Setup needs to be completed. This identifies you in both EDI files and

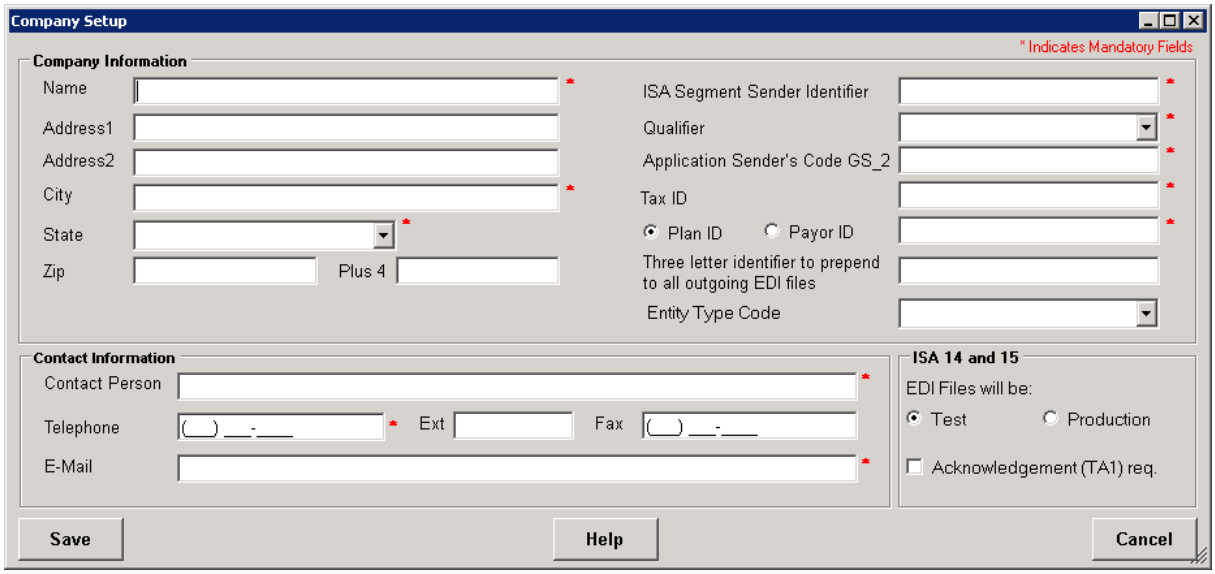
SOAP/MIME message data.

From the Setup drop-down menu, select "Company Setup".



Company Setup menu option.

Fill in the Company Setup form. Set the Production flag in the ISA 14 and 15 section to Production when you are done testing.



All fields marked with red asterisks are mandatory.

1.5.2 HIPAA Claim Status Responder

In order to receive and respond to ASC X12 005010X212 Health Care Claim Status Requests (276) you must have installed HIPAA Claim Status Responder and configured it as follows.

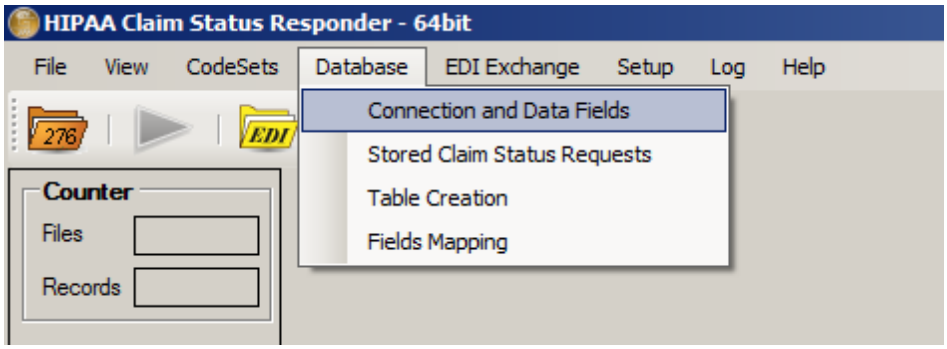
Please download and install this software from HIPAAsuite and configured it as follows. The program itself comes with a detailed help documentation. We just give below a quick start guide.

If you have not completed the Company setup, please refer to the HIPAA Claim Status

Responder guide.

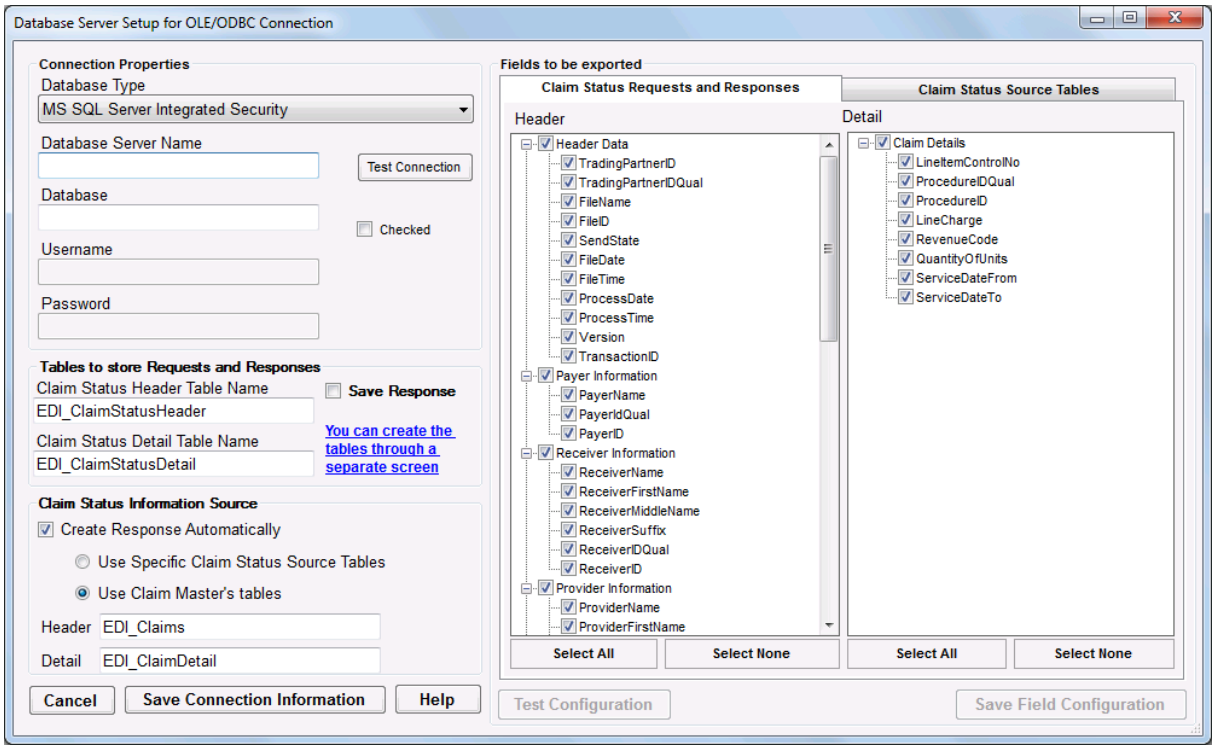
Next, the database needs to be configured.

In the Database dropdown menu, select "Connection and Data Fields".



The Database dropdown menu.

This will open a form where you can fill in your database connection details. This is the database HIPAAsuite Claim Status Responder will be checking for claim status.



This form stores all necessary Database settings.

Fill all fields and be sure to check "Save Responses in database". This presents additional tables necessary for correct response behavior.

Click the "Test Connection" button to verify your database connection details are correct. If you have not created the database or tables a pop-up message will inform you the database could not be opened or the tables were not found.

If a database does not exist, consult your local Administrator.

If you have not yet created the tables, you may click the blue underlined text "You can create the tables through a separate screen" to open a window that will automatically create them for you.

This text links to the table creator.

The table creation window. Here you can create the tables with the necessary fields or compare the scripts to your own tables.

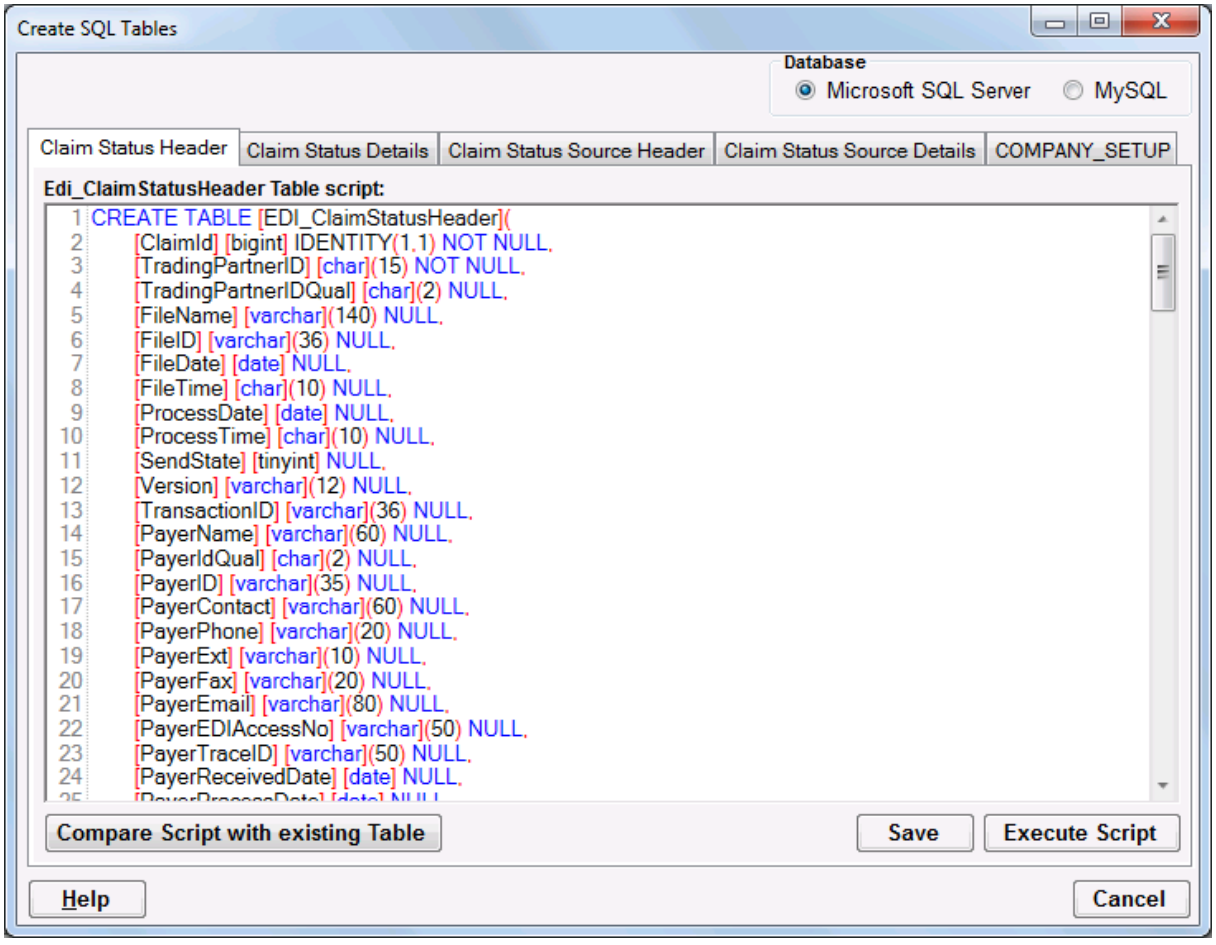
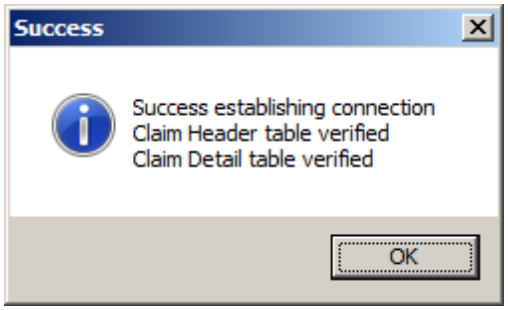


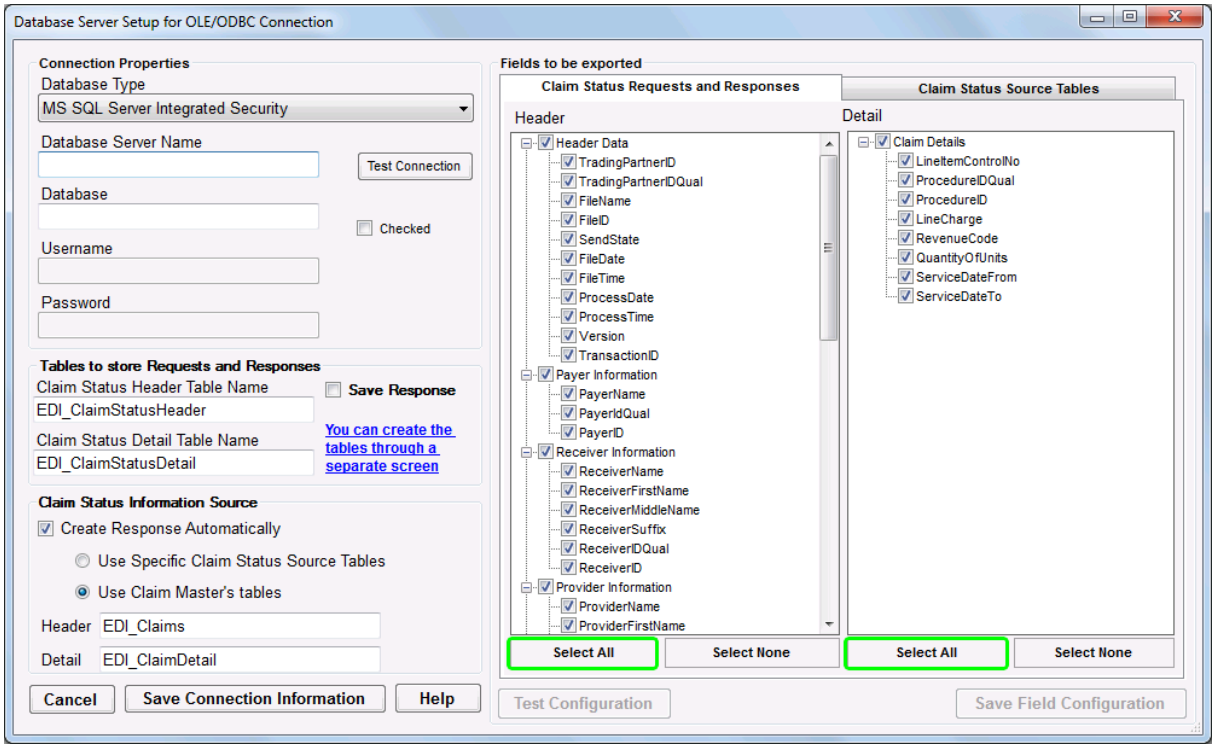
Table creation made simple.

Testing your database connection with the "Test Connection" button should now yield the following message:



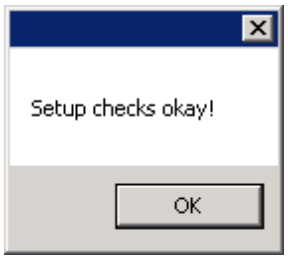
Connection test successful.

After creating the tables, the field configuration must be tested. In the Database Connection and Data Fields menu make sure all fields are selected for both tabs. Click the "Select All" button near the bottom of the form to check all fields. Then, click "Test Configuration", and later the "Save Field Configuration" button.



"Select All" buttons near the bottom make field selection quick.

Click the "Test Configuration" to verify the existence tables and fields within the database, then "Save Field Configuration" to save.



Setup successful.

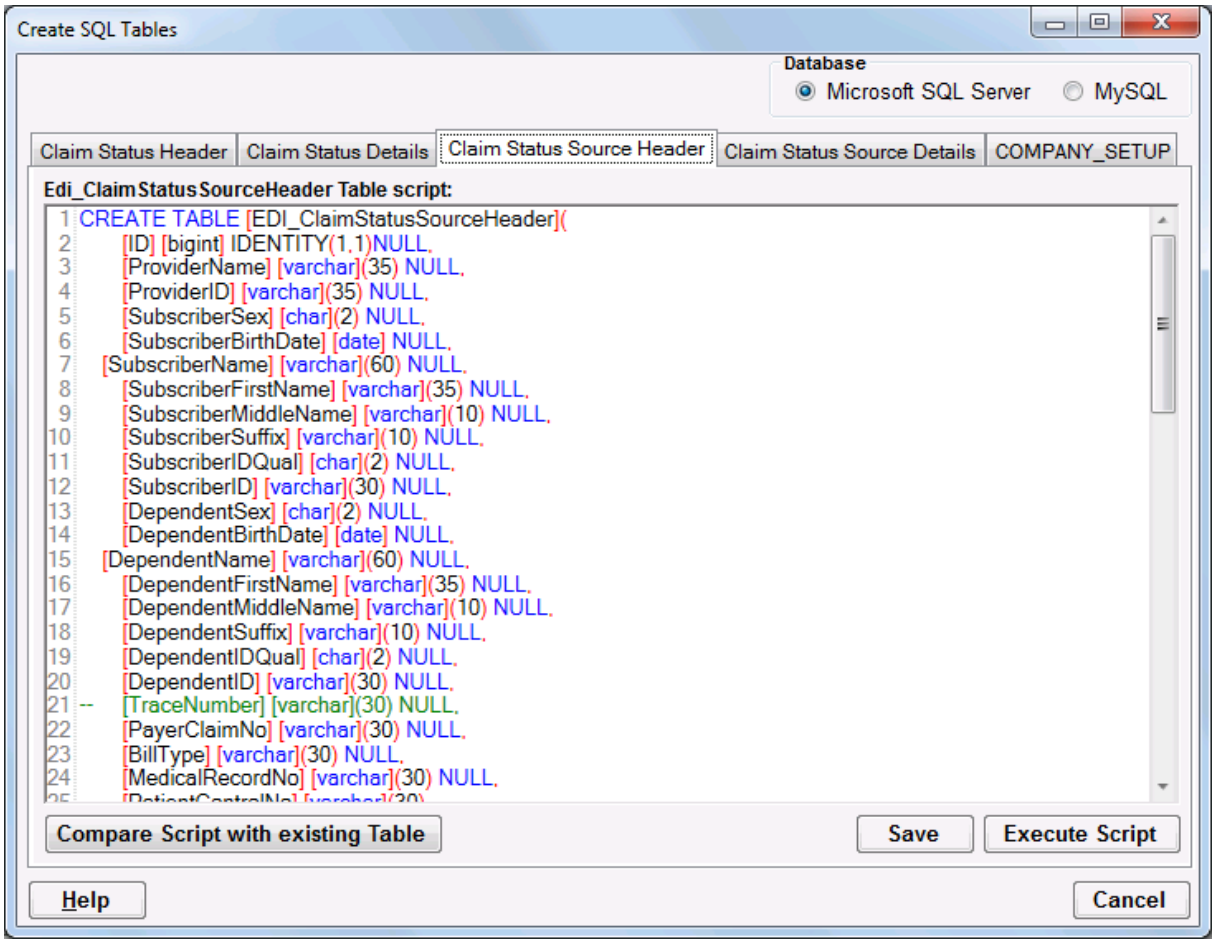
The Database is now configured and HIPAAsuite Claim Status Responder can answer 276 Claim Status requests. Follow the next steps to select a source for Claim Status information.

Claim Status Information Source

There are two ways to draw Claim Status information from DB. One uses Source tables that can be created from the HIPAA Claim Status Responder itself. The other uses the HIPAA Claim Master application's tables with a slight modification.

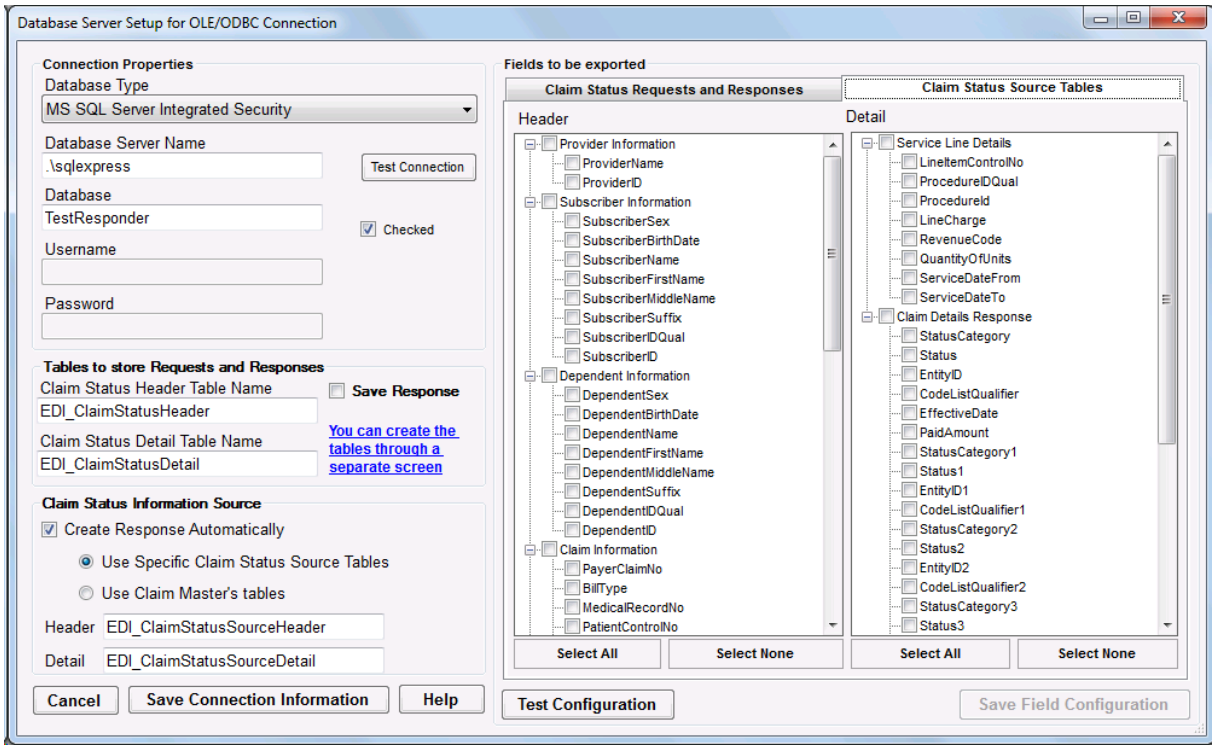
Source Tables

The source tables are created in the same window as the rest of the Claim Status tables. These are the *Claim Status Source Header* and *Claim Status Source Details* tables. If you are using the Claim Master tables, ignore this section.



Source table creation.

After creating the Source tables, the HIPAA Claim Status Responder must be configured to use them. In the DataBase Connections setup screen, check "*Create Response Automatically*" and select "*Use Specific Claim Status Source Tables*".

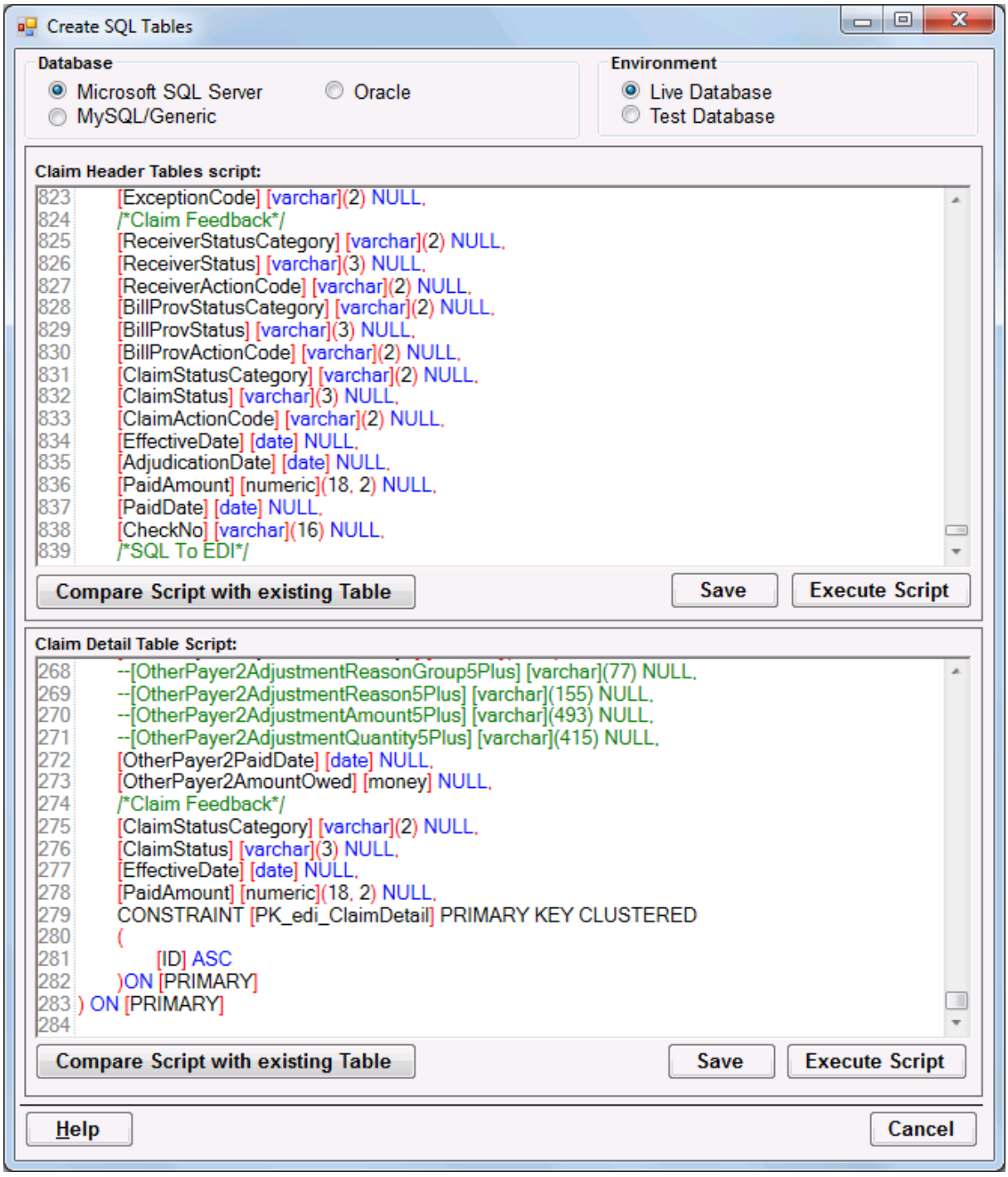


Specific Source Tables selected.

In the same screen, select the "*Claim Status Source Tables*" tab and select all fields. Remember to save your configuration.

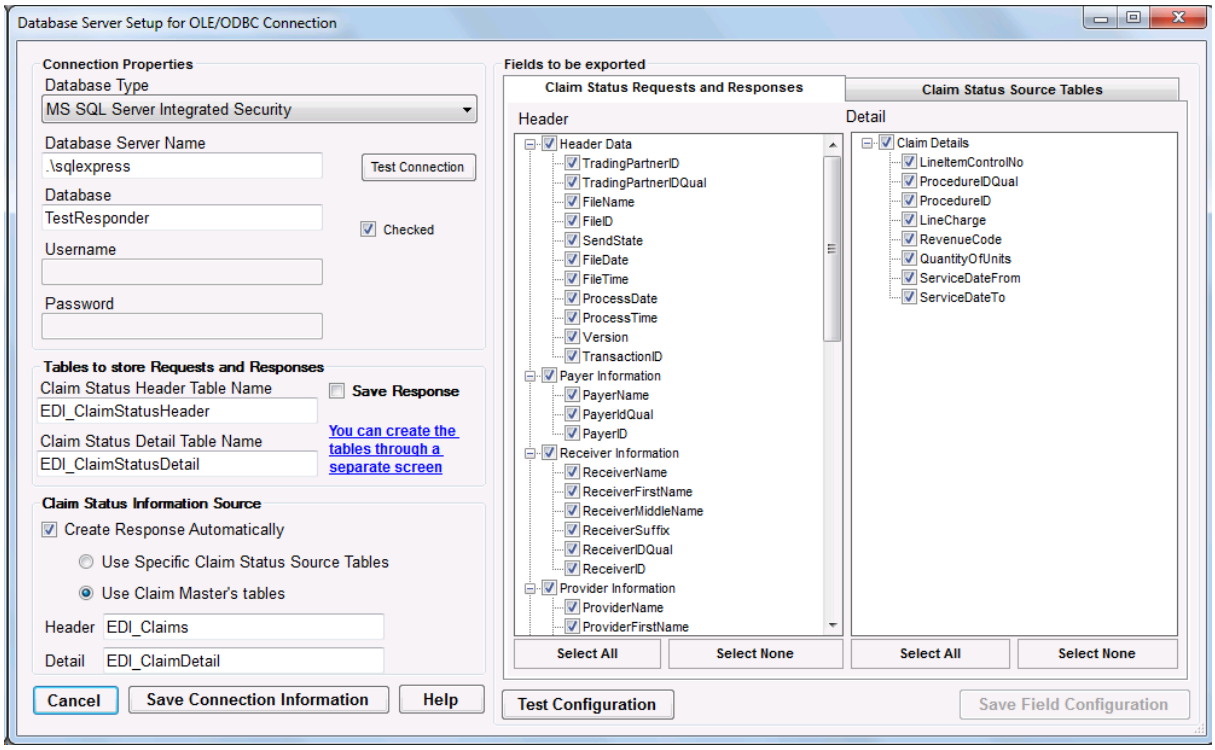
Claim Master Tables

To use the HIPAA Claim Master tables as a source, the Claim Master tables need to be modified. In HIPAA Claim Master, open the *Table Creation* window and remove the double hyphen ("--") at the beginning of any line in the *Claim Feedback* section of both scripts. If you are using the Source tables, ignore this section.



Your Create Table form should look like this.

Save and execute both scripts. The tables should now have the necessary Claim Status information fields. Now we set up the HIPAA Claim Status Responder to use these. In the DataBase Connections window, check "*Create Response Automatically*" and select "*Use Claim Master's Tables*".



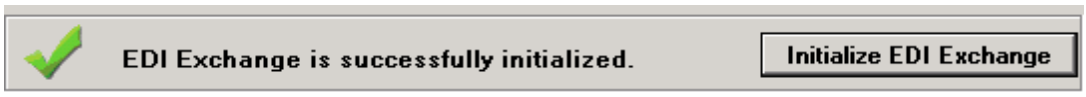
Using the Claim Master Tables.

Then select the "*Claim Status Source Tables*" and check all fields. Remember to save your configuration.

Autoprocess Options

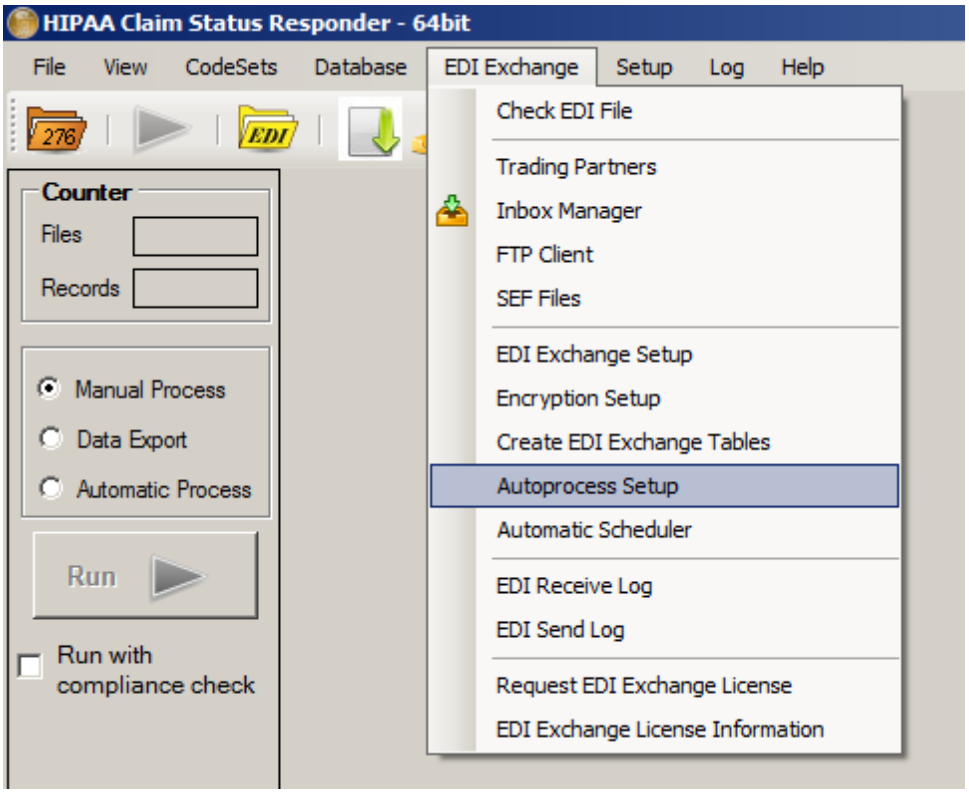
If you have not yet set up EDI Exchange, instructions to do so can be found in the HIPAA Eligibility Responder setup section.

To verify EDI Exchange is setup, select "EDI Exchange Setup" from the EDI Exchange dropdown menu and look for a green checkmark indicating EDI Exchange has been initialized successfully.



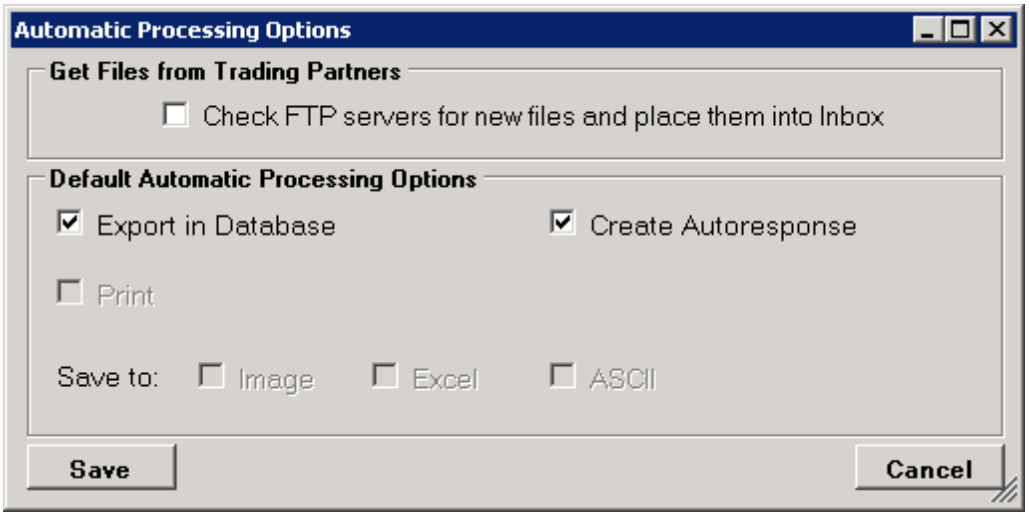
Edi Exchange is initialized.

Finally, the autoprocessing options need to be set. In the EDI Exchange dropdown menu, select "Autoprocess Setup".



Autoprocess Setup menu option.

In the following window, check options "Export in Database" and "Create Autoresponse."



Autoprocessing options.

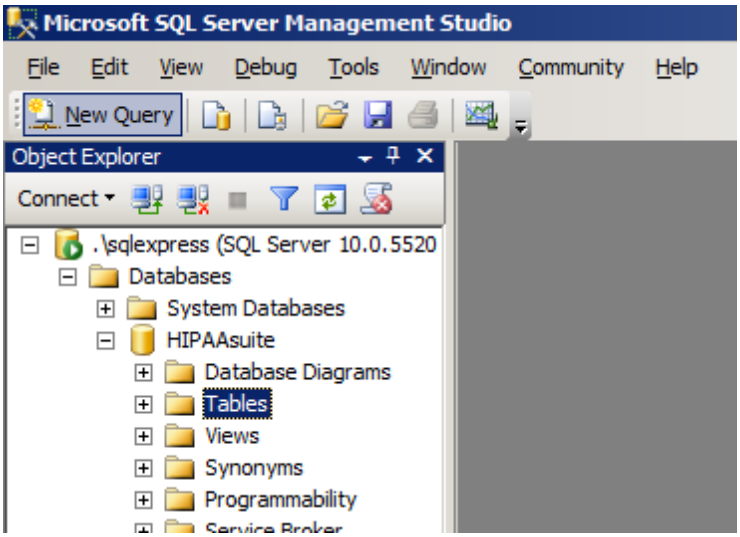
1.5.3 Testbed Data

In order to facilitate testing with clients, the CORE testbed data has been included in the form of scripts. We received these data files from EDIfecs and used them during the

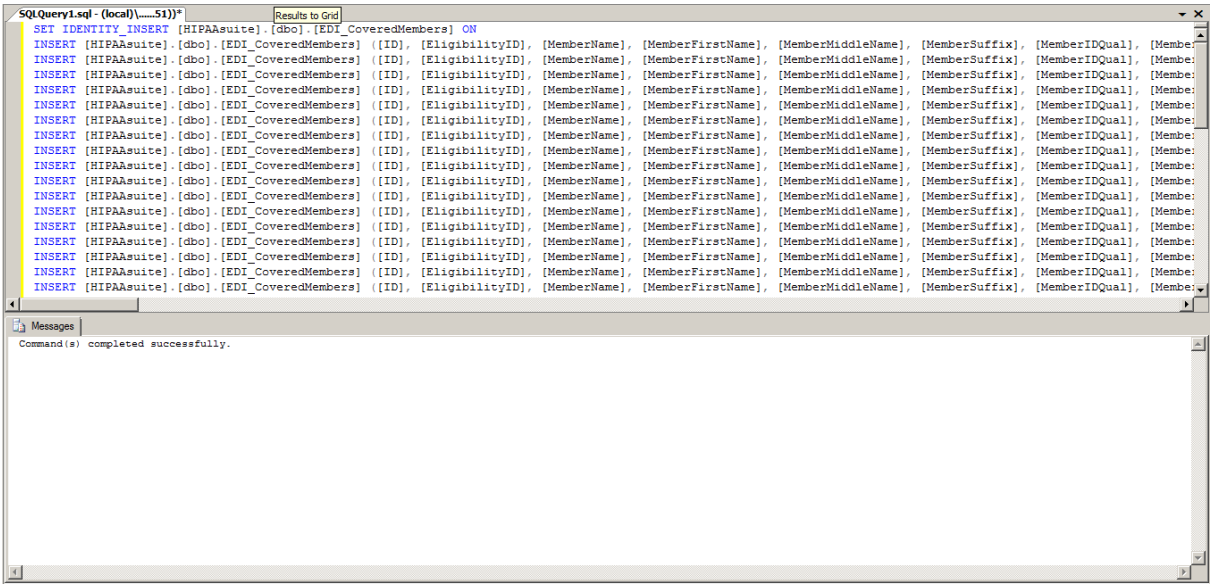
certification process.

This data consists of multiple subscribed clients, their dependents (if any), and all relevant medical coverage data. To execute the scripts and insert the test data into your database, follow these instructions. We recommend you execute the scripts in a clean or test database.

First, open MS SQL Server Management Studio and log into your database server. Click "New Query". This will open a query window in which you can execute queries directed at your database.



Open the "SQL Write Testbed Data HEADER.txt" file, copy its contents and paste them within the query window. Execute the query. This will insert Covered Member info into the HIPAAsuite database created by HIPAAsuite Eligibility Responder. Each row contains the identity of a single covered member.



After inserting covered members into table EDI_CoveredMembers, their eligibility details must be filled in. Without these, there is no data with which to respond to an Eligibility request. Repeat the same steps as above: copy the contents of "SQL Write TESTBED DATA.txt", paste them into a clean query window, and execute the query.

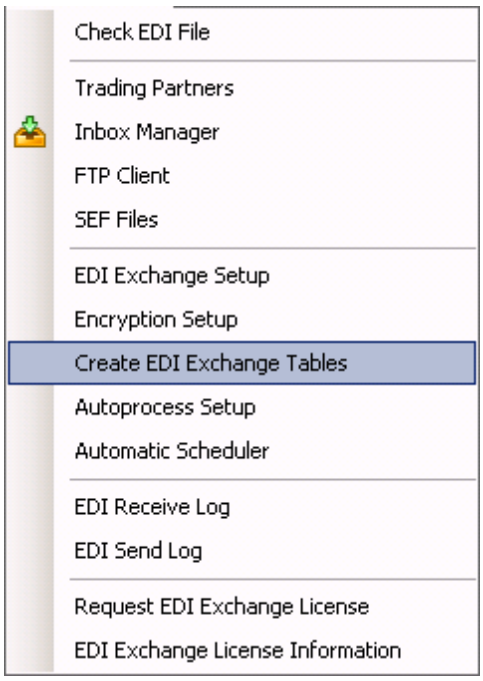
These rows contain all of the necessary information to describe a patient's benefit coverage. Each row describes a single Eligibility of Benefit code for a single patient. It will contain the Coverage Level, Service Type Code, time period, any monetary amounts for co-payment, deductibles, and the like, any percentage amounts for co-insurance, their availability within or without the network, etc. The single row describes the insurance information for a service type without the need for multiple entries as represented in an EDI file. This cuts down on overhead compared to storing each EB segment on its own.

1.5.4 EDI Exchange

EDI exchange comes with all HIPAAsuite application. But in order to work it must be licensed and correctly set up. Again the following is only a quick start guide and more detailed information can be found in the HIPAAsuite application.

In order to send EDI files to a trading partner, the relevant tables must be set up in your database. These tables will store your Trading Partners' details and EDI file transfer logs.

First, select "Create EDI Exchange Tables" in either HIPAAsuite Eligibility Responder or Claim Status Responder under the "EDI Exchange" drop-down option on the menu bar.



The "EDI Exchange" drop-down menu.

The following window will be displayed.

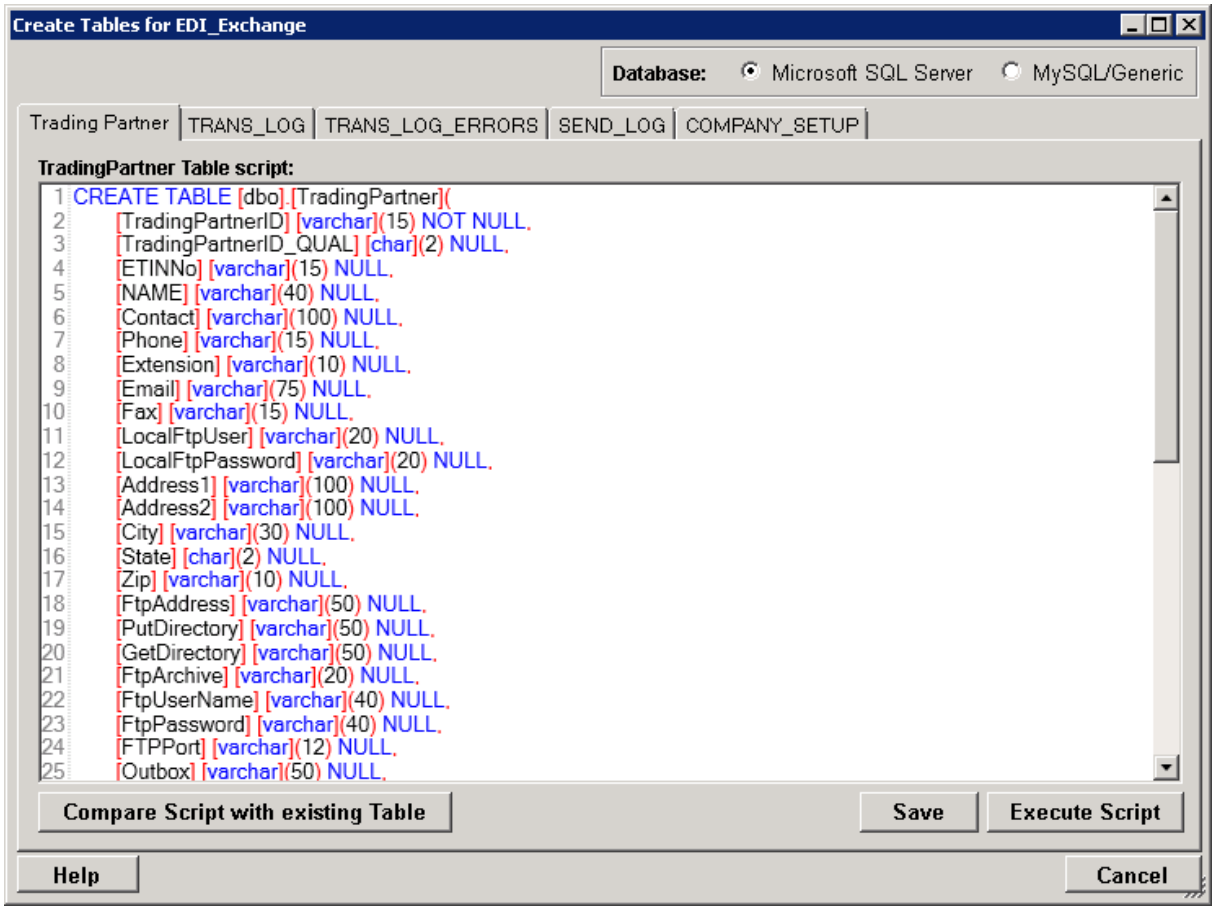
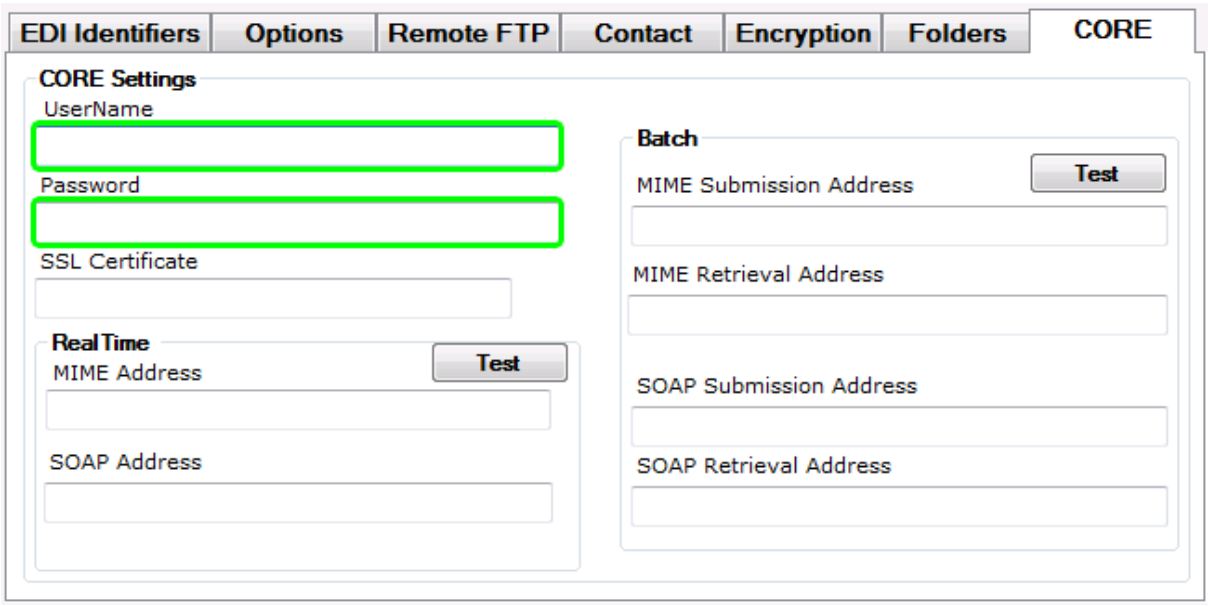


Table creation for EDI Exchange.

Execute the CREATE TABLE script for any table not in your database. The tables are the same for HIPAAsuite Eligibility Responder and Claim Status Responder, so this only needs to be done once.

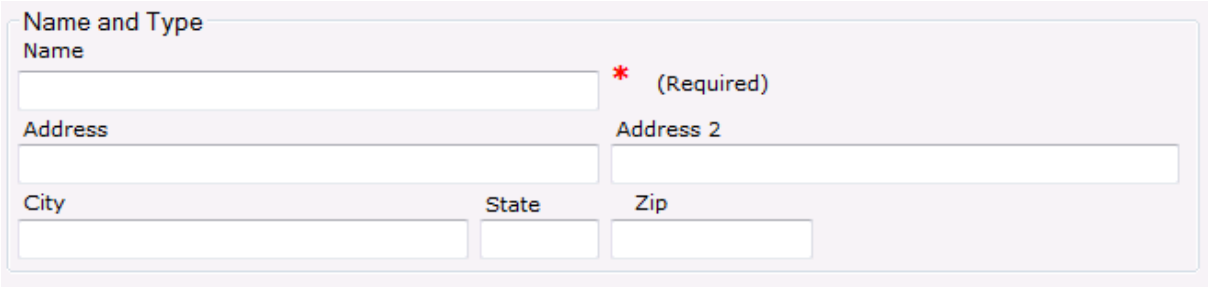
1.5.5 Trading Partners

The HIPAASuite RealTime Server uses username/password tokens to authenticate Trading Partners. The authentication token must match the username and password specified for a specific Trading Partner. These are set up in the Eligibility/Claim Status Responder's *EDI Exchange* ▸ *Trading Partners* menu.



Username and Password fields for authentication token.

The Trading Partner's Name must also match the SenderID in the SOAP or MIME message.



Name field for a Trading Partner.

The Name, Username, and Password fields are all case-sensitive.

1.6 Features

Here is a list of features provided by the HIPAAsuite RealTime Server:

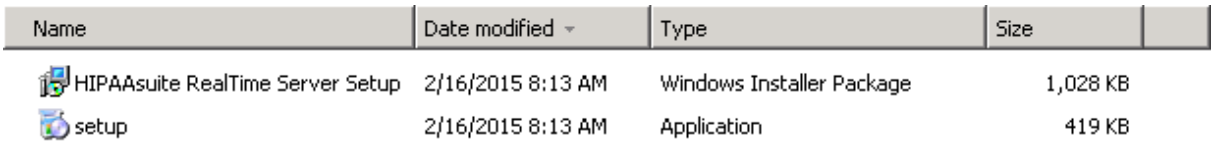
- Username/Password client authentication.
- Accepts ASC X12 005010X279A1 Eligibility and Benefit Requests (270) in SOAP and MIME formats.

- Accepts ASC X12 005010X212 Health Care Claim Status Request (276) in SOAP and MIME formats.
- Interacts with other HIPAAsuite programs to generate 271 and 277 responses and acknowledgments.
- Finds correct response to each request and sends it to client automatically (in Real Time) or on request (in Batch mode).
- Rejected Real Time transactions automatically return a Functional Acknowledgement (999).
- Debug mode allows you to see all incoming and outgoing messages along with many helpful process messages on the console screen.
- Default port hosted uses HTTPS when a valid domain certificate is provided.
- Test port allows you to test your connections using HTTP.

2 Installation and Setup

2.1 Installation

Once you download the compressed file with the HIPAAsuite RealTime Server, you see the following files.

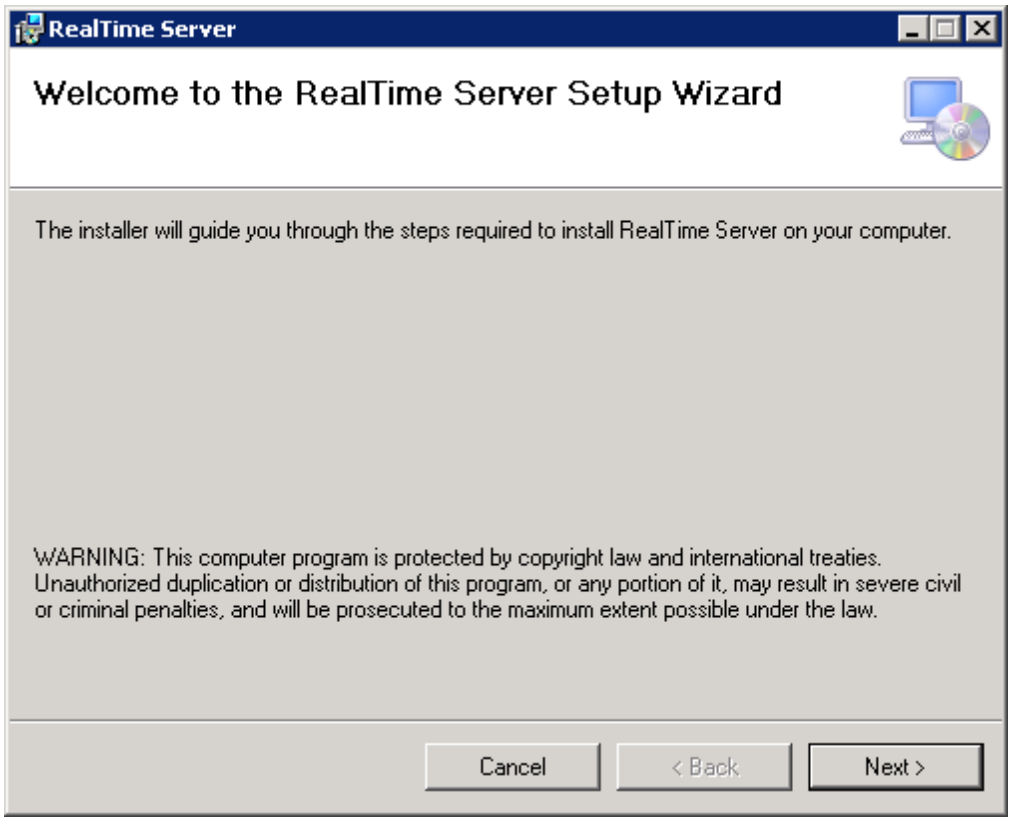


Name	Date modified	Type	Size
HIPAAsuite RealTime Server Setup	2/16/2015 8:13 AM	Windows Installer Package	1,028 KB
setup	2/16/2015 8:13 AM	Application	419 KB

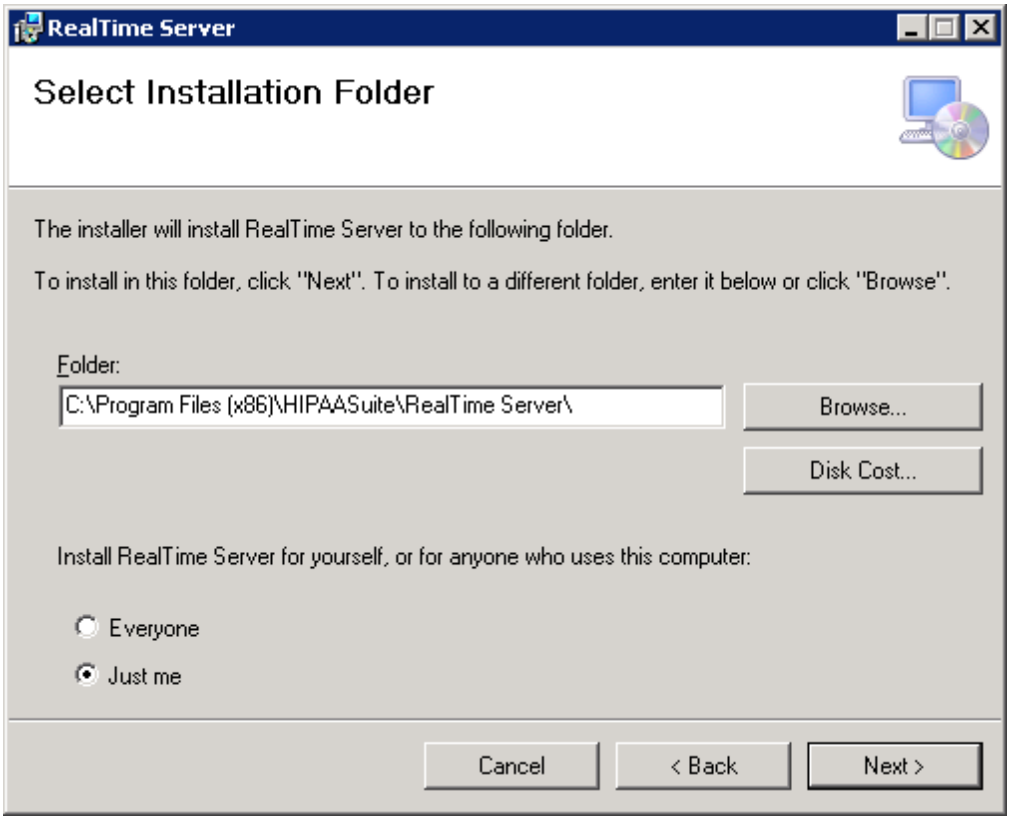
The installation package contains two files. Run "setup".

Please run setup! The HIPAAsuite RealTime Server uses several Windows components that have to be merged into the kernel. The Setup.exe program takes care of that. But without these dll's installed the server will not work.

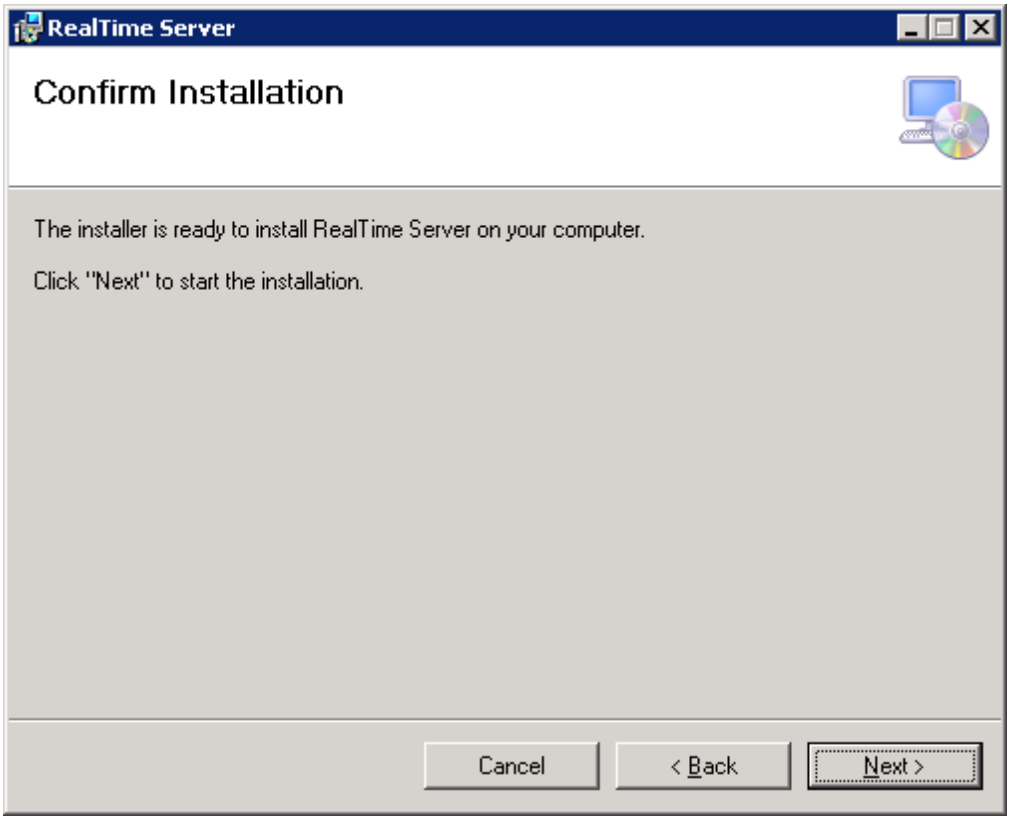
After setup finishes it call the actual install program and the following screen appears.



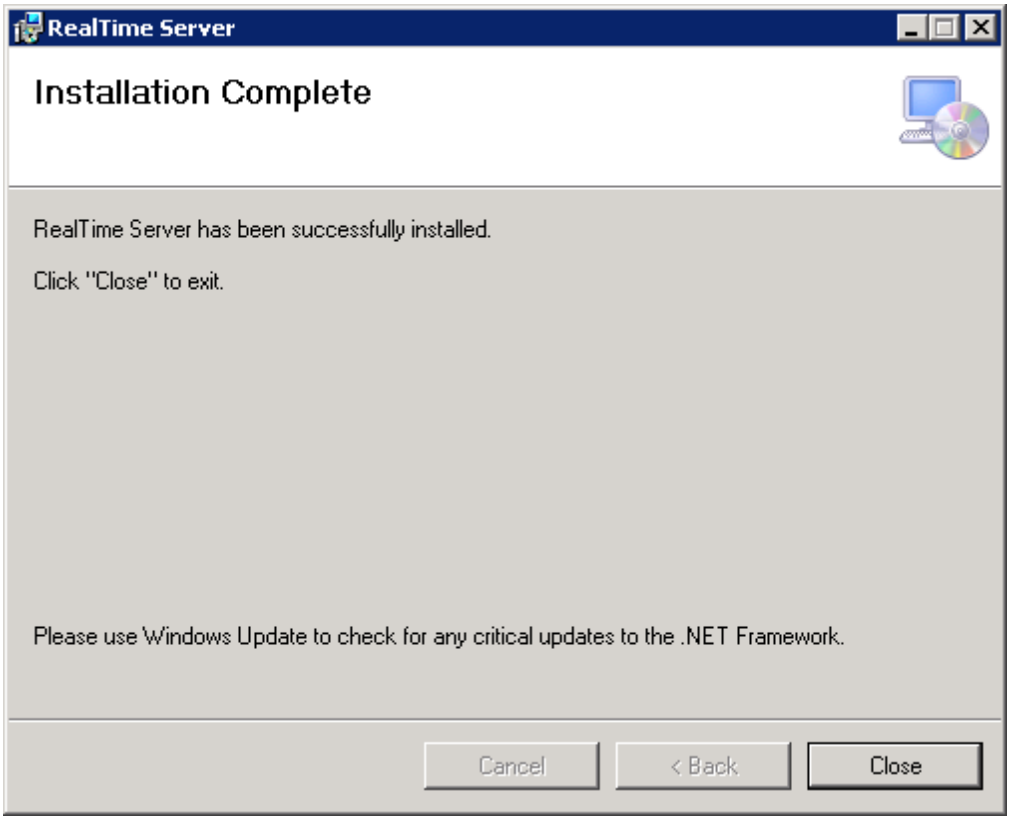
Click on 'Next'



Select your installation folder.



Click on 'Next'



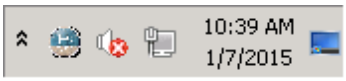
The installation finishes

You are ready to configure HIPAAsuite RealTime Server.

2.2 Setup

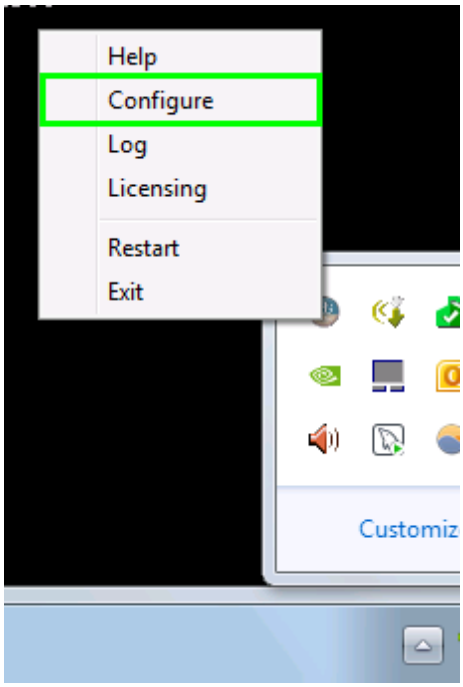
Before you can use your HIPAAsuite RealTime Server, you must first configure Database connection settings as well as the domain and ports it will be hosted in. Any changes made to the configuration will take effect after the HIPAAsuite RealTime Server is restarted.

1. To configure HIPAAsuite RealTime Server, right click the HIPAAsuite RealTime Server notification icon.



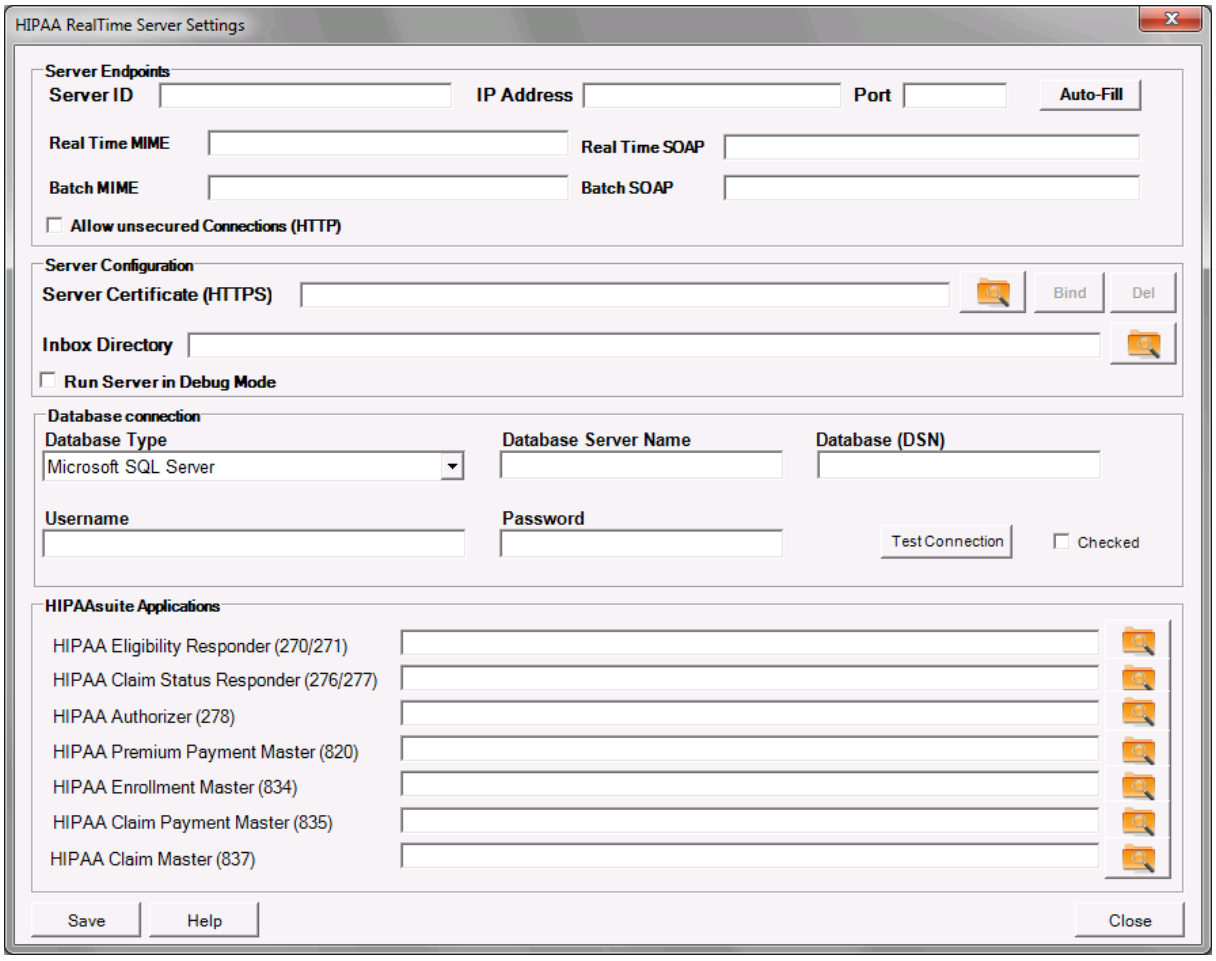
HIPAAsuite RealTime Server notification icon. The icon may be hidden.

2. Select "Configure"



HIPAAsuite RealTime Server notification icon menu.

3. The following screen will appear. The configuration file is stored in C:\ProgramData \HIPAAsuite\HIPAA RealTimeServer and should not be modified while the server is in operation.



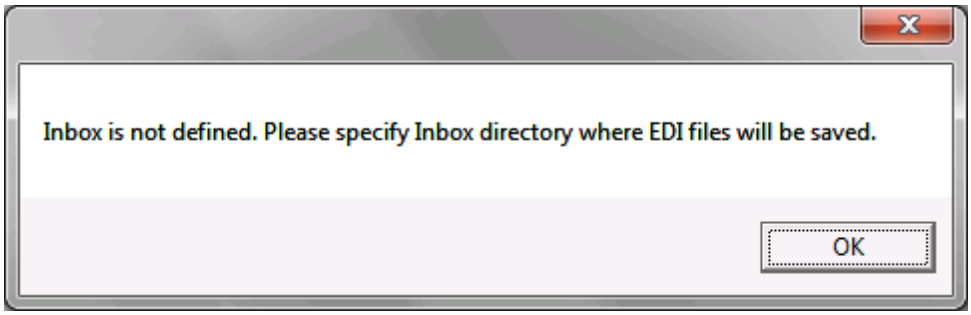
The HIPAAsuite RealTime Server setup window.

Server Configuration

Here you will configure your Inbox directory, SSL certificate, and toggle Debug mode.

Inbox Directory:

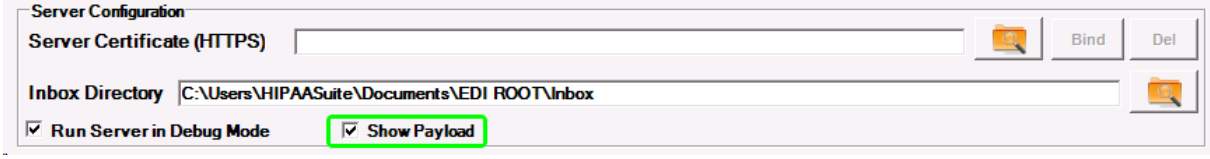
The HIPAAsuite inbox folder used to store all incoming EDI files are stored here. This folder is automatically generated by the HIPAAsuite responder applications when setting up the location of your EDI Exchange Root Directory path. Without an inbox directory an incoming EDI file cannot be written to disc. This field is mandatory and not filling it in will yield an error message.



Run Server in Debug Mode:

Checking this option will enable debug mode. In debug mode, the server displays a console through which incoming and outgoing transactions, their relevant metadata and useful notifications can be seen. For further information, see here.

Checking the Debug Mode option will present the option of displaying the entirety of the incoming and outgoing EDI payloads. The "Show Payload" option is off by default and if left unchecked, only the ISA segments will be presented, as these contain no identifying information on the subjects of the EDI transactions.



"Show Payload" option checkbox in Server Configuration settings.

SSL Certificate:

The Authentication Certificate you will be using to encrypt the transport layer. The SSL Certificate can be bound to or unbound from the port the secure service will be hosted on when running the HIPAAsuite RealTime Server as an Administrator. More here

Server Endpoints

The domain/address the HIPAAsuite RealTime Server will be hosted in; EDI clients will send and receive messages to and from this address.

Server ID:

The name you wish to give your HIPAAsuite RealTime Server. This name will be used to identify you in the SOAP or MIME envelope formats.

IP Address:

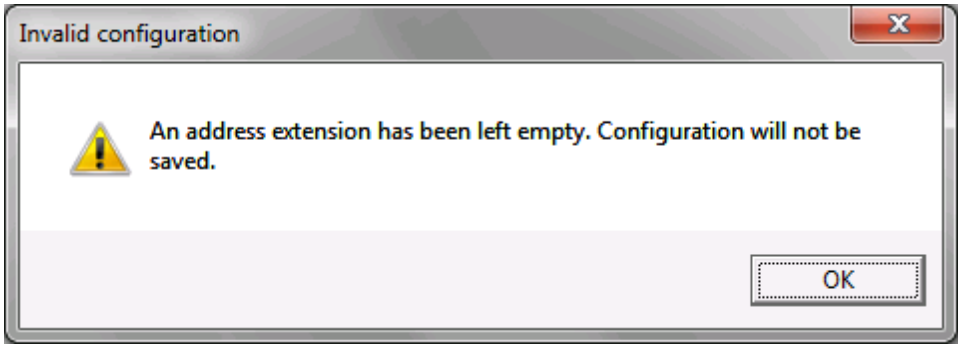
The publicly accessible IP address or URL your HIPAAsuite RealTime Server will be hosted in.

Port:

The SSL port your HIPAAsuite RealTime Server will be hosted in. This port will be used to host an SSL-secured endpoint for HIPAA RealTime Server. The setup must be run as an Administrator if the port is to be changed, as namespace reservations needed to use these ports without Administrative rights can only be created as an Administrator.

Auto-Fill:

This button will fill in the Realtime and Batch endpoints with suggested address extensions; an address extension is necessary in all fields.



RealTime SOAP and Batch SOAP address extensions must be unique, as they use different envelope formats and need different endpoint addresses.

Allow Unsecured Connections:

If checked will present a textbox to fill in the port number for the unsecured endpoints. The unsecured endpoint will have the same address (exempting port number) and address extensions for services as the secured endpoint but will use HTTP instead of HTTPS. For more information see here.

Database Connection

The database connection details. Specify Database Management System, location, and user credentials.

Database Type:

Select from Microsoft SQL Server and Microsoft SQL Server Integrated Security.

Database Server Name:

Name/address of server the database is hosted in.

Database (Data Source Name):

Name of the Database.

Username:

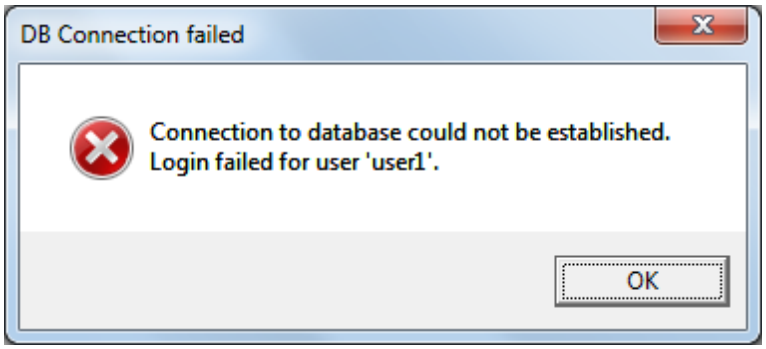
Username for Database connection. Only necessary is not using SQL Server Integrated Security.

Password:

Password for Database connection. Only necessary if not using SQL Server Integrated Security.

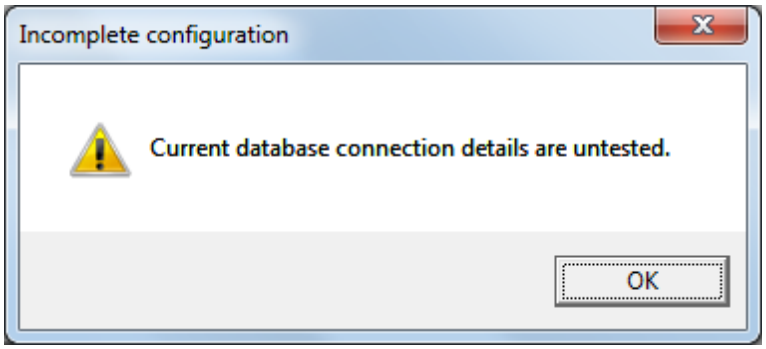
Test Connection:

The Test Connection button will test your database connection details using the connection details specified. If your connection details are correct, the checkmark to the right of the "Test Connection" button will be filled. If there are any problems connecting to the database using the connection details you provided, a popup window will be displayed specifying the reason.



Connection failure window. An incorrect password was supplied for the username.

Saving the configuration without having tested the database connection details will still save the configuration, but you will be warned beforehand.

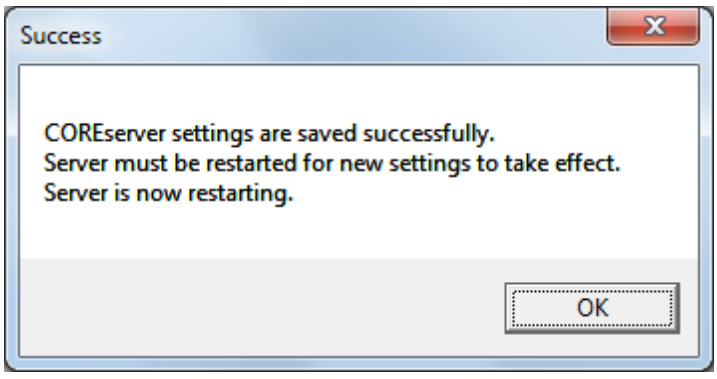


Database connection was not tested before saving configuration.

HIPAAsuite Applications

Executable paths for all the HIPAAsuite applications. The appropriate HIPAAsuite application will be called whenever an EDI file is received by HIPAAsuite RealTime Server.

After saving your preferences, it is always necessary to restart your HIPAAsuite RealTime Server for the changes to take effect. The RealTime Server will restart itself for you.

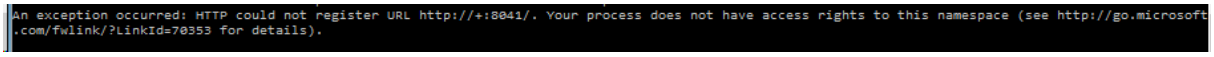


Server will restart after a successful configuration.

Troubleshooting

This section contains errors shown on the debug screen or log for quick reference.

An exception occurred: HTTP could not register URL <url>. Your process does not have access right to this namespace.



Namespace reservation error.

The RealTime Server reserves one or more HTTP namespaces for the kernel-level HTTP listener. Namespace reservations enable processes running under a non-administrator account to listen using the HTTP listener. Administrative users do not have to reserve a namespace. In order to run RealTime Server using a non-administrative account, you must first reserve the namespace using an administrative account. This applies to deletions and changes to namespace reservations, as well.

2.3 Server Certificates

As we mentioned earlier, the HIPAAsuite RealTime Server uses Secured Socket Layers for security reasons. This is also known as HTTP over SSL or in short HTTPS. This secure transport layer is based on an encryption certificate. These certificates have several capabilities:

- Encrypt your application layer data. (In your case, the application layer protocol is HTTP.)
- Authenticate the server to the client.
- Authenticate the client to the server.

When you connect to a site that is secured by a certificate, a behind the scenes process exchanges the public part of the server certificate to the client and the client browser has also a public key which it exchanges with the Server. Once the public keys are exchanged, all the traffic between client and server, back and forth, are securely encrypted.

Certificates are issued by Certificate Authorities. There are a handful of recognized

organizations in the world that generate and sell such server certificates. These certificates will also identify and verify the domain for which they are issued and guaranteed by the authority. This makes sure that nobody could for example fake a certificate for the New York Times because no Certificate Authority will do that for a rogue player. The most known certificate authorities are Verisign, GeoTrust, Thawte, DigiCert and Comodo. They sell several forms of server certificates. The purchase of the Certificate is left to the system administrator. Certificate vary in price from 50 to several thousand dollars.

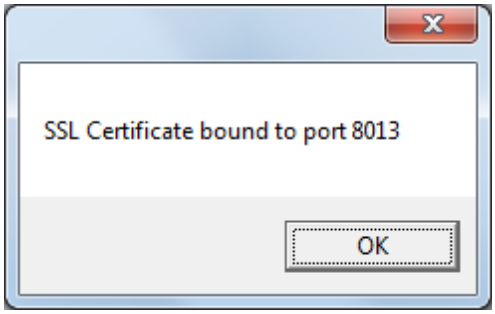
As mentioned, certificates are valid for a domain. You cannot deploy the HIPAAsuite RealTime Server without having an internet domain like www.MyHealthPlan.com and deploy the server using a fixed IP address. In addition you need a valid server certificate.

Binding a certificate to port:

To bind a certificate to a port using RealTime Server, the application must be run as an Administrator. You will need the certificate's full file path. If you used relative addressing to specify the certificate path when creating a self-signed certificate, assuming the default installation location for Windows SDK, your certificate path will be something like `C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin\<certificatepath.cer>`. Write the filepath of the certificate (including filename) into the textbox and click the "Bind" button to the right of the textbox.

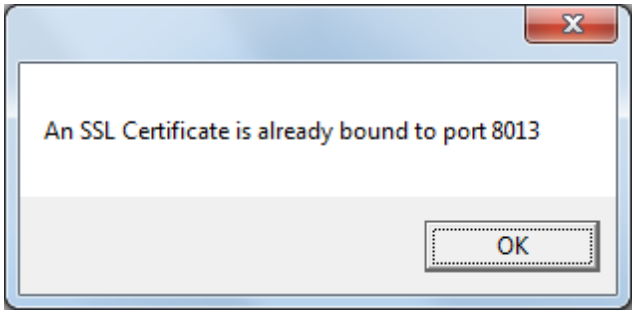
Server Configuration group

This will bind the specified certificate to the port specified in Server Endpoints -> Port in the same configuration window and a message will be shown:



Successful binding of selected SSL Certificate to secure port

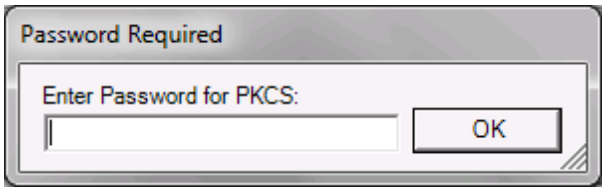
If a certificate has already been bound to the secure port, a message will be shown:



The port already has a certificate bound to it.

.pfx

A certificate export file generated from the key store can also be used. These files are generally password-protected, so you will be prompted for one upon clicking the "Bind" button.



Password prompt for SSL certificate

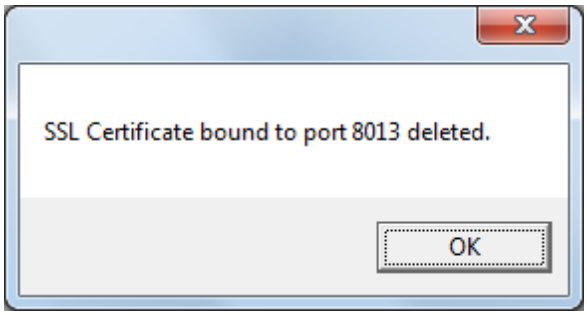
If the password is correct, the certificate represented in the .pfx file will be bound to your secure https port.

Removing a bound certificate from a port:

To remove a certificate bound to your Secure SSL port, RealTime Server must be run as an Administrator. This will remove any SSL Certificate bound to the port specified in Server Endpoints. Click the "Del" button.

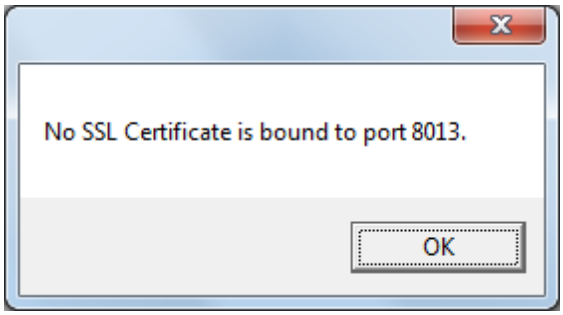
Button to delete a certificate bound to a port

This operation will free up the port specified in Server Endpoints. When performed correctly, the following message will appear:



Success in unbinding an SSL Certificate from a port

Trying to delete an SSL certificate from a port that has none will yield this message:

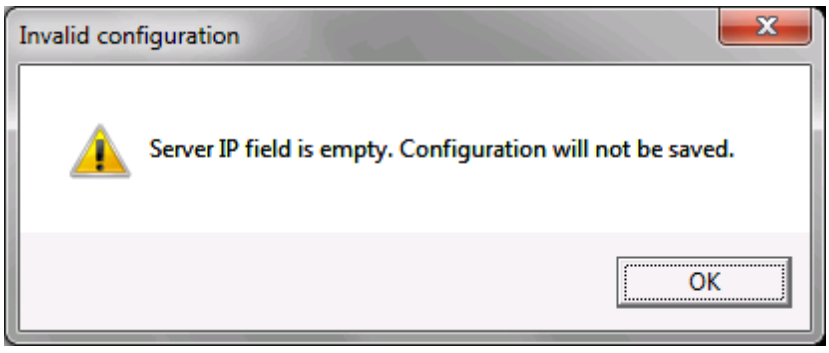


No SSL Certificate for the port

2.4 Setup Troubleshooting

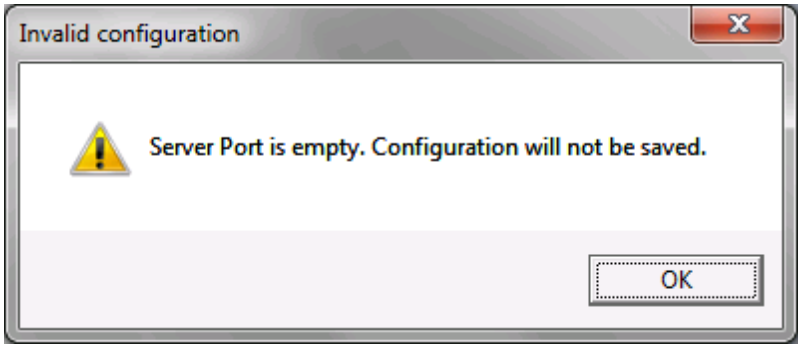
These are errors you may see during the server configuration and their solutions.

Missing Server IP field. The Server IP is the public-facing IP for your server.

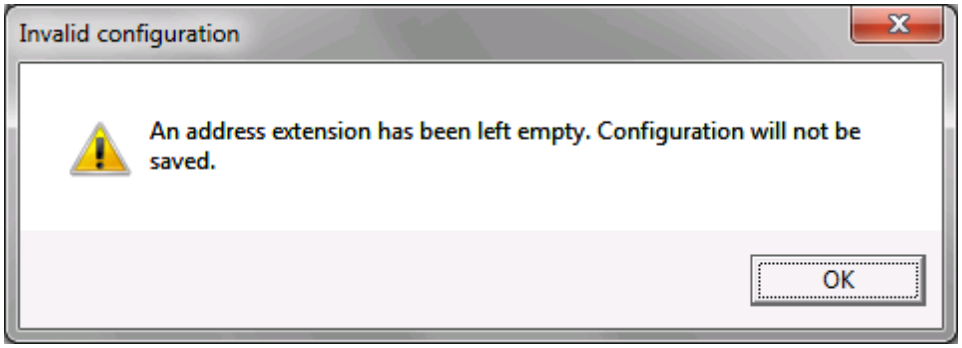


Missing Server Port. The Server Port is the port your RealTime Server will listen on for

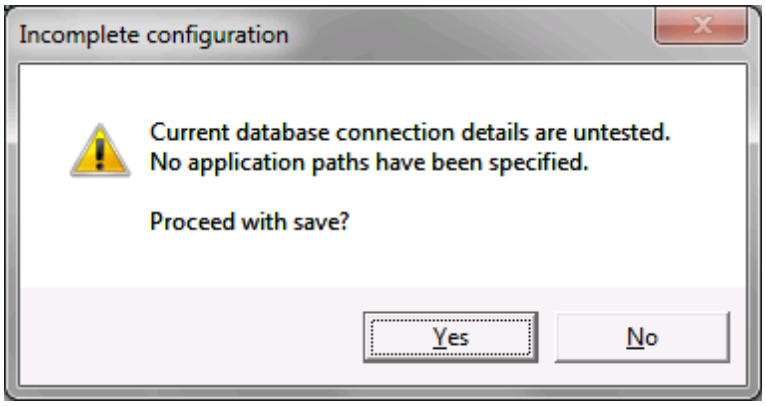
incoming transactions. Ports must be set or changed by an administrator.



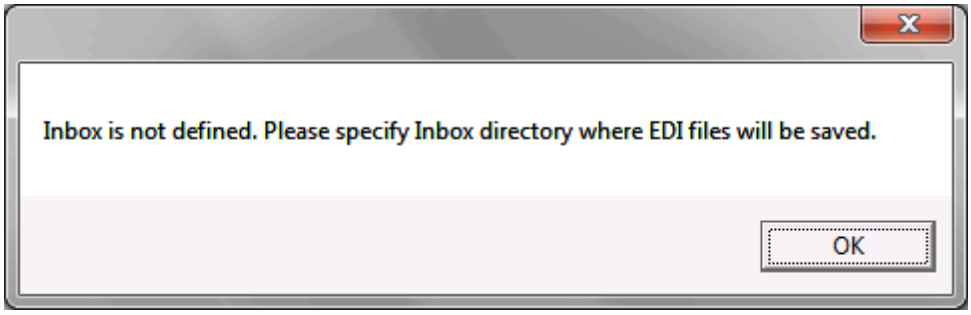
Missing address extension. All four are necessary. They determine where each service is hosted. MIME uses the same envelope format in both Real-Time and Batch modes and do not need to be different. However, SOAP Real-Time and Batch modes use SOAP and MTOM message envelopes, respectively, making different addresses for each a necessity.



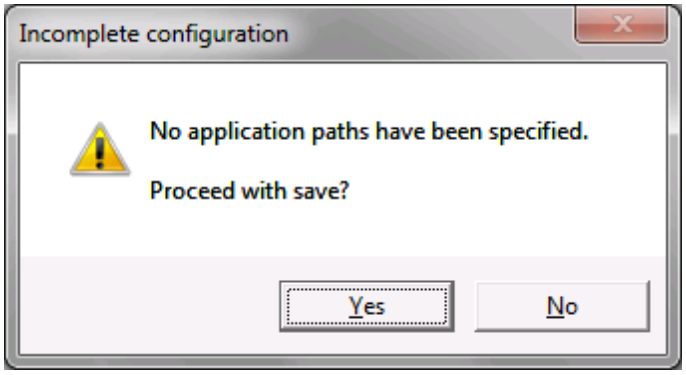
Server connection options have not been tested. If you are sure the connection details are correct, this warning can be ignored and the Real Time Server's settings can be saved.



Missing inbox address. The inbox address is necessary to determine where HIPAA RealTime Server will deposit incoming EDI files. Use your existing HIPAAsuite EDI Exchange EDI Root folder.

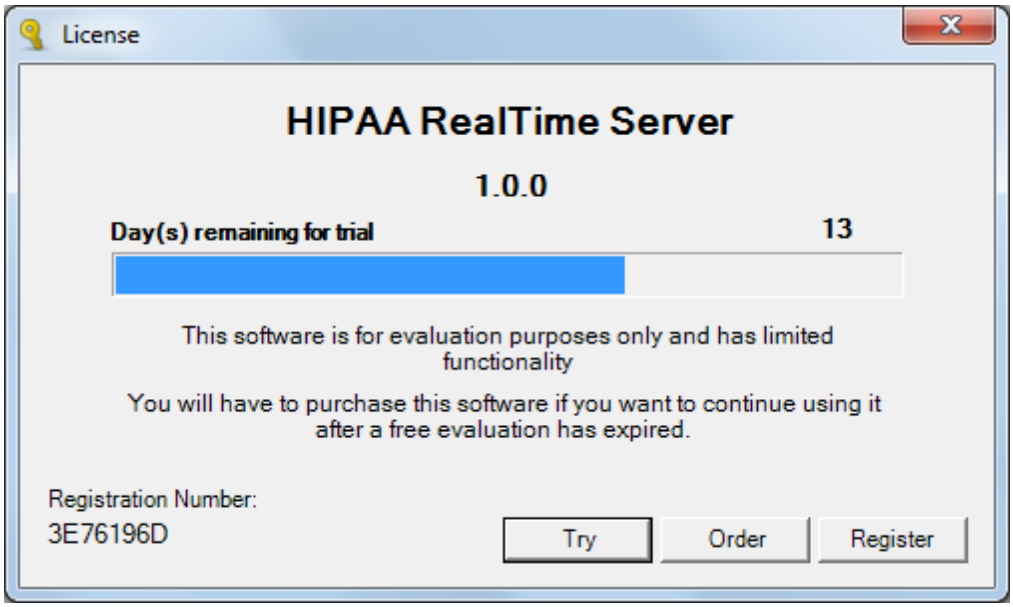


Missing HIPAAsuite application path(s). At least one HIPAAsuite application compatible with HIPAA RealTime Server (HIPAA Eligibility Responder, HIPAA Claim Status Responder, etc) is necessary to make use of the HIPAA RealTime Server. The .exe file for any of these is found in their respective install folders.



3 Software Registration

The HIPAA RealTime Server can be tested free of charge. Starting the server during the trial period will present the following trial screen before starting up. This screen shows how many days remain in the free trial:

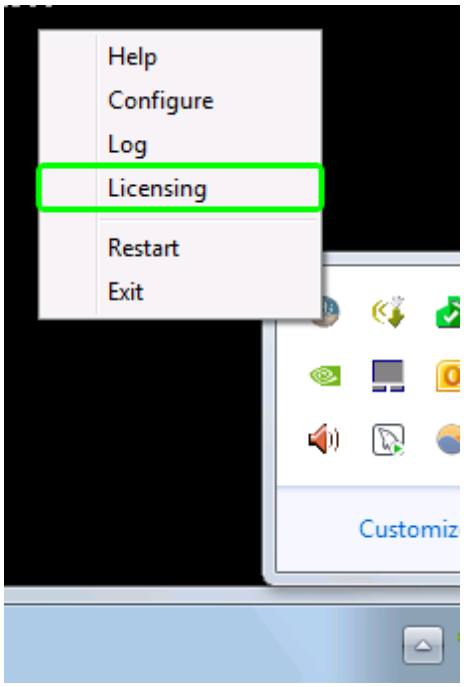


The Trial Screen

During the testing period, the Server will not open an SSL host. All hosted addresses will be unsecured http. During this time, the log will make note of this every time the server is started. A typical server startup log entry with a trial license will read as follows:

```
The service is ready in debug mode.
16:19:48.7403136 Starting Server...
       Loading Config...
       Config Loaded.
Trial license. Secure host cannot be opened. Insecure test host will be opened.
Test host open.
The service is ready in debug mode.
```

Whether or not the current license is a trial license will be displayed when starting up the server or by checking the "Features and Licenses" window by clicking the "Licensing" option in the right-click menu.

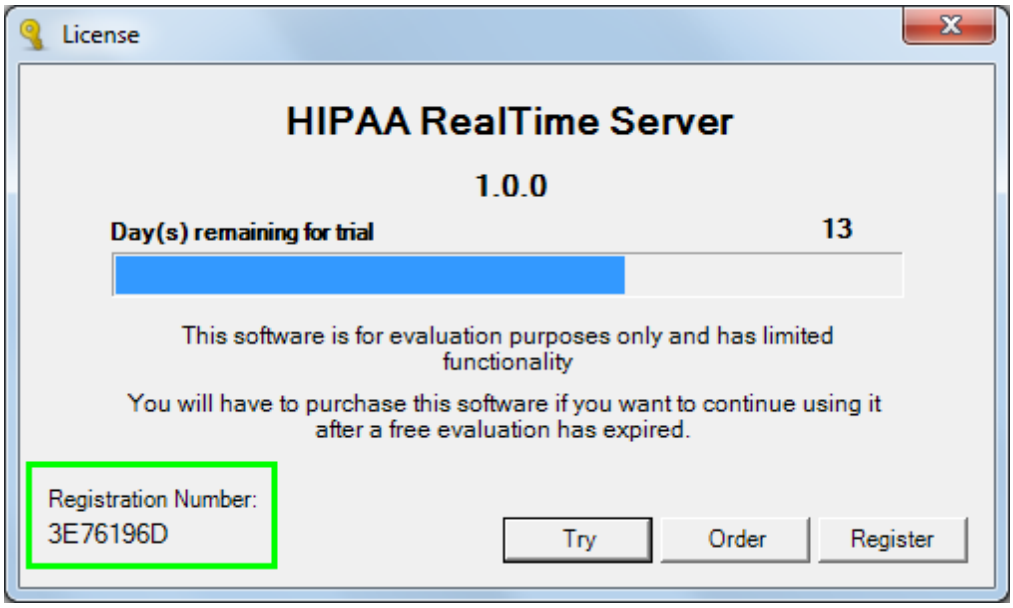


Right-click menu. "Licensing" option is highlighted.

Licensed window displaying an Expired Trial Key.

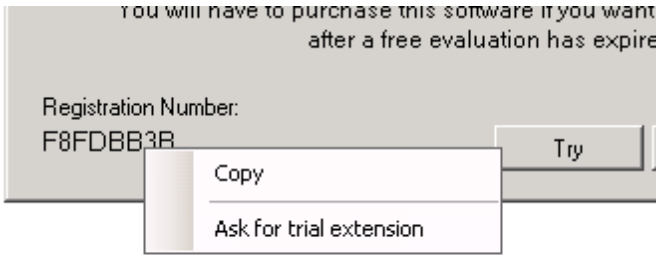
Should your trial time expire and you wish to continue your testing of the software, please send an email to info@HIPAAsuite.com with the Registration number and we will give you a trial extension. Clicking the "Request Trial Button" from the "Features and Licenses" window will open your default mailing application.

The registration number is shown on the lower left hand corner of the license trial window.



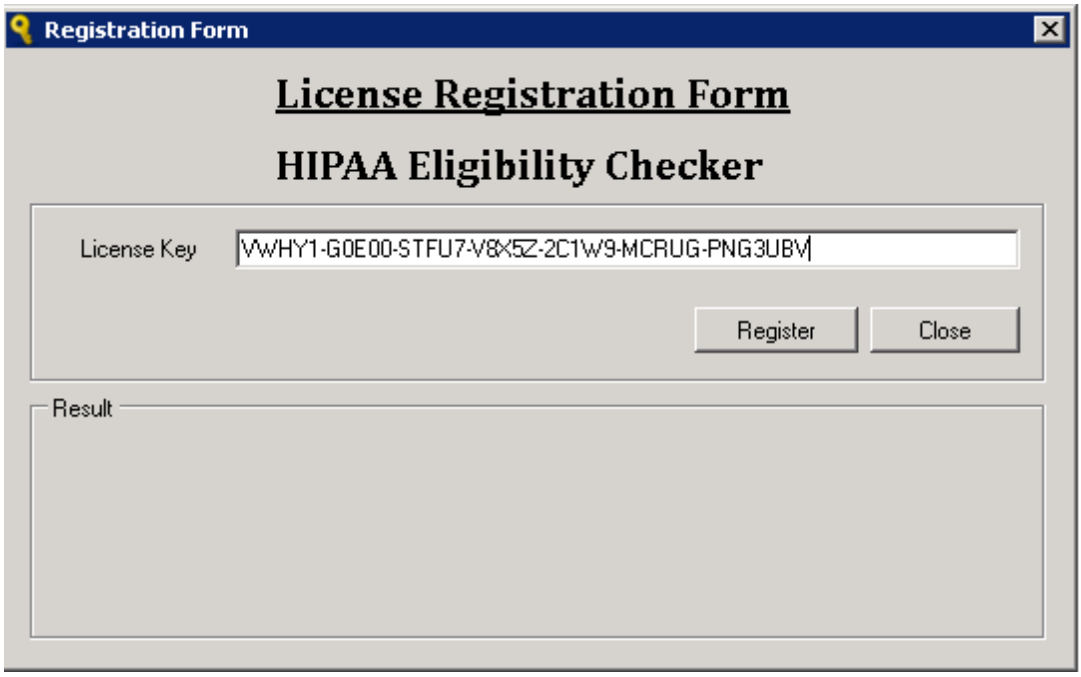
The Registration number

This number is needed for the registration as well as trial extension. It is unique to your computer and hardware. **You can copy the registration number to the clip board, just hover with the mouse over it**, so that you can easily paste it into an email. This avoids human error. (There are no letter 'O's, the letter in the number, only zeros!)



Copy the registration number to the Windows clip board by hovering over the it with the mouse

Once you have received the license key from us you click on the 'Register' button and the license registration from comes.



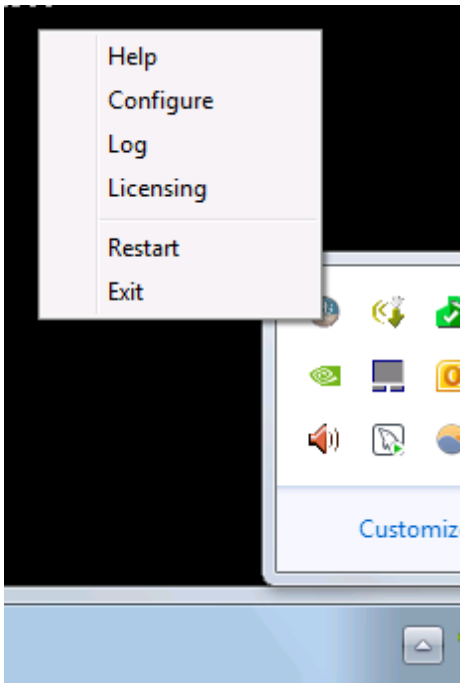
Entering the registration information

Enter the License Key as supplied in our email. The best is to copy and paste the information from our email.

Once the product is registered with a permanent unlock code, the SSL-secured https host will be available for use and all future upgrades will find this key and install without further action necessary.

4 Context Menu

Being a mostly formless windows application, the HIPAAsuite RealTime server doesn't present the user much in the way of a Graphical User Interface. Right-clicking the application's icon will present you with a menu from which you can select various options.

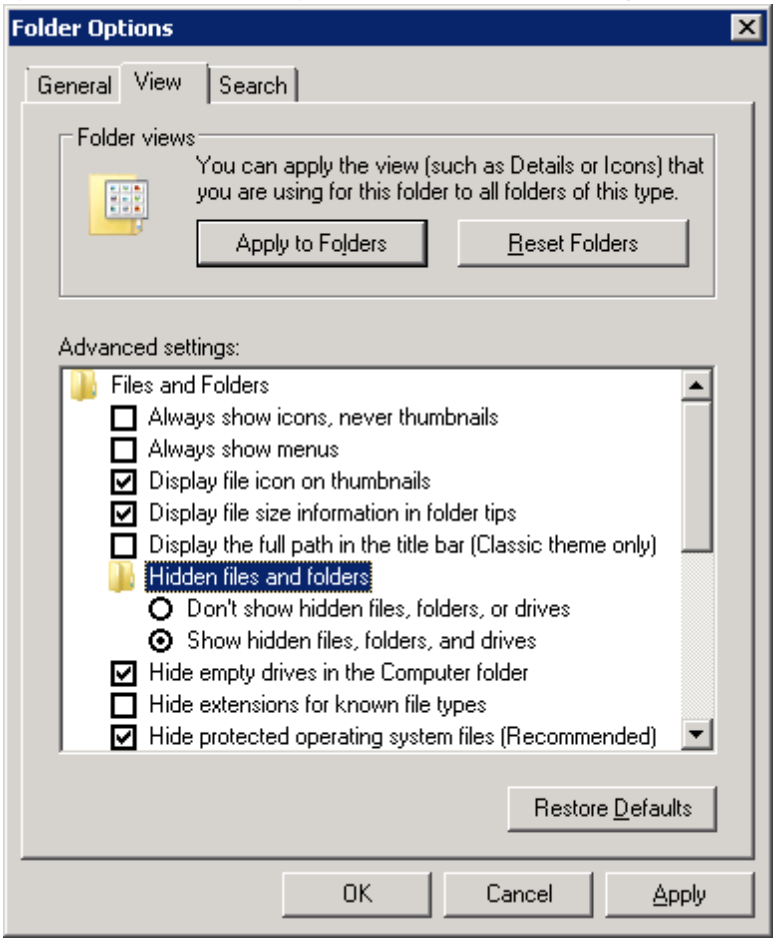


Icon right-click menu.

- Help - Brings up this helpfile.
- Configure - Presents you with the configuration window. More details on this here.
- Log - Presents you with the program log. Log files are stored in C:\ProgramData \HIPAAsuite\HIPAA RealTimeServer. This is a quick way of accessing them.
- Licensing - Presents all licensing information, whether the registered license is a trial or full license, and if it is a trial license how many days left before expiration.
- Restart - Restarts the RealTime Server.
- Exit - Exits the RealTime Server and closes any open hosts and the debug console window.

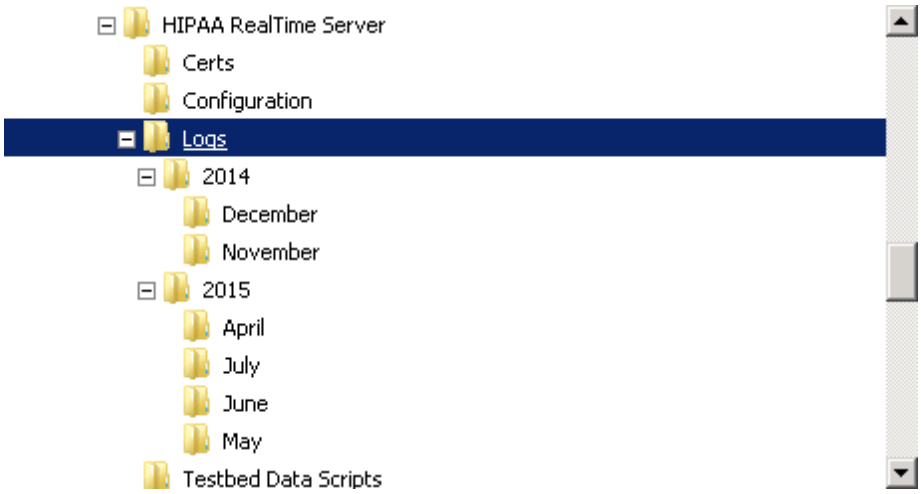
5 Logging

HIPAA RealTime Server logs all transaction information in text files that are stored in C:\ProgramData\HIPAAsuite\HIPAA RealTime Server\Logs. If you cannot see this folder, it is because it is hidden in Windows by default and can only be seen in Windows Explorer if you change the default settings.



Folder options in WIndows Explorer

Logs are stored in directories divided by year and subdivided by month. A single log file will be created per day.



Log tree.

6 MIME

Definition of the MIME-Multipart communication protocol and how it relates to EDI transfer.

6.1 What is MIME

MIME (Multi-purpose Internet Mail Extensions) is an extension of the original Internet e-mail protocol that specifies how messages must be formatted so that they can be exchanged between different email systems. This standardization allows people exchange different kinds of data files on the Internet: audio, video, images, application programs, etc., as well as the ASCII text handled in the original protocol, the Simple Mail Transfer Protocol(SMTP). All manually composed and automated emails are transmitted through SMTP in MIME format. The association of Internet email with SMTP and MIME standards is such that the emails are sometimes referred to as SMTP/MIME email.

In 1991, Nathan Borenstein of Bellcore proposed to the Internet Engineering Task Force (IETF) that SMTP be extended so that Internet, but mainly Web, clients and servers could recognize and handle other kinds of data than ASCII text. As a result, new file types were added to "mail" as a supported Internet Protocol file type. This meant that email was no longer limited to a single ASCII-character text attachment per email message. MIME is extensible because it defines a method to register new content types. New MIME data types are registered with the Internet Assigned Numbers Authority (IANA).

Servers insert the MIME header at the beginning of any Web transmission. Clients use this header to select and appropriate "player" application for the type of data the header indicates. Some of these players are built into the Web client or browser (for example, all browsers come with GIF and JPEG image players as well as the ability to handle HTML files); other players may need to be downloaded. The MIME header describes the data contained in the message so that it can be represented correctly by the recipient:

MIME Version: The presence of MIME Version generally indicates whether the message is MIME formatted. The value of the header is 1.0 and it is shown as `MIME-Version: 1.0`. The idea behind this was to create more advanced versions of MIME like 2.0 and so on, but this has had the opposite outcome. A way of handling a future MIME version was never adequately specified.

Content-Type: This describes the data's Internet media type and its subtype. It may consist of a 'charset' parameter separated by a semicolon specifying the character set to be used. For example: `Content-Type: Text/Plain.`

Content-Transfer-Encoding: It specifies the encoding used in the message body (originally limited to the ASCII character set). This header indicates whether a binary-to-text encoding scheme has been used on top of the original encoding as specified within the Content-Type header and which one or, if no encoding scheme was used, a descriptive label for the format of the content.

Content-Description: Provides additional information about the content of the message.

Content-Disposition: Specifies the presentation style, the name of the file (if any), and creation and modification dates. which can be used by the reader's mail client to store the attachment. For example: `Content-Disposition: attachment; filename=genome.jpeg; modification-date="Wed, 12 Feb 1997 16:29:51 -0500";`. Originally, MIME specifications only described the structure of mail messages without addressing presentation styles.

The following is an example of a MIME message as it would arrive in your inbox:

```
From: Jerry Peek <jpeek@jpeek.com>
To: carlos@entelfam.cl
Subject: Un =?iso-8859-1?Q?d=EDa_dif=EDcil?=
MIME-Version: 1.0
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: quoted-printable

[...message content goes here...]
```

6.2 The Multipart Content-Type

Through the use of the `multipart` content type, MIME allows mail messages to have parts arranged in a tree structure where the leaf nodes are any non-multipart content type and the non-leaf nodes are any of a variety of multipart types. This supports any number of subtypes, such as simple text messages (`text/plain`), text with attachments, and replies with original attached; it also supports alternative content, such as a message in both plain text and HTML, images (`image/jpeg`), audio (`audio/mp3`), video (`video/mp4`),

applications (`application/msword`); and so on.

A MIME multipart message contains a boundary in the `Content-Type` header. This boundary must not occur in any of the parts and is usually a randomly generated string; it is up to the client to choose a boundary string that doesn't clash with the body text. It is placed at the beginning, between message parts, and at the end of the Multipart message. The last boundary is always followed by two hyphens identifying it as such. Each part bookmarked by the boundary contains its own header and body. An example:

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=frontier

This is a message with multiple parts in MIME format.
--frontier
Content-Type: text/plain

This is the body of the message.
--frontier
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB0aGUg
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--frontier--
```

6.3 MIME in CORE

This section contains examples of MIME Multipart messages depicting the request and response required for patient eligibility checks.

The MIME EDI messaging envelope contains metadata related to its payload and may sometimes not contain the payload it describes. Fields common to all the messaging envelopes are:

- `PayloadType`: Identifies the kind of transaction the message is transmitting. Not all payload types are tied to a specific EDI transaction, some may be for confirmations of message arrival or error notifications.
- `ProcessingMode`: Identifies whether the message is to be processed in real time or batch

modes.

- `PayloadID`: Unique identifier for the related payload. The payload does not need to be in the message; the message may be an acknowledgement of a received request.
- `TimeStamp`: Time in which the message was sent.
- `SenderID`: Identifies the sender of the message.
- `ReceiverID`: Identifies the recipient of the message.
- `CORERuleVersion`: Idenfities the CORE rules in effect. Currently 2.2.0.

Request messages will always contain a header through which the sender is verified as an authorized user of the receiving service. This verification can be done through username/password tokens or the use of an x.509 certificate. The examples provided use the former method, as does HIPAAsuite RealTime Server.

The response message will not contain a header, but does have two additional fields for error processing: `ErrorCode` and `ErrorMessage`.

The MIME message envelopes for real time transactions are almost identical. Their only difference is the aforementioned distinction between request and response messages. The response in real time is meant to answer the request in full. The following are examples of a real time request and a real time response.

Real time Eligibility Request message:

```
POST /core/eligibility HTTP/1.1
Host: server_host:server_port
Content-Length: 2408
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_270_Request_005010X279A1
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
RealTime
--XbCY
Content-Disposition: form-data; name="PayloadID"
e51d4fae-7dec-11d0-a765-00a0c91e6da6
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
```

```
--XbCY
Content-Disposition: form-data; name="UserName"
hospa
--XbCY
Content-Disposition: form-data; name="Password"
8y6dt3dd2
--XbCY
Content-Disposition: form-data; name="SenderID"
HospitalA
--XbCY
Content-Disposition: form-data; name="ReceiverID"
PayerB
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY
Content-Disposition: form-data; name="Payload"
<contents of file go here -- 1674 bytes long as specified above>
--XbCY--
```

Real Time Eligibility Response Message:

```
HTTP/1.1 200 OK
Content-Length: 2408
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_271_Response_005010X279A1
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
RealTime
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0c91e6da6
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="SenderID"
PayerB
--XbCY
Content-Disposition: form-data; name="ReceiverID"
HospitalA
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY
Content-Disposition: form-data; name="ErrorCode"
Success
--XbCY
Content-Disposition: form-data; name="ErrorMessage"
None
--XbCY
Content-Disposition: form-data; name="Payload"
<contents of file go here -- 1674 bytes long as specified above>
--XbCY--
```

Batch messages, unlike the real time messages, won't necessarily contain a payload. Some messages (Batch Retrieval Requests, Batch Submission Acknowledgement Retrieval Requests, Batch Submission Responses, and Batch Results Acknowledgement Submission Responses) exist to confirm the arrival of a payload at the receiving end of the transaction.

In addition to the fields in the real time MIME messages, batch MIME messages have the following fields whenever a payload is included in the message:

- `PayloadLength`: Defines the length of the actual payload in bytes.
- `Checksum`: An element used to allow the receiver to verify the integrity of the included payload.

Batch Submission message:

```
POST /core/eligibility HTTP/1.1
Host: server_host:server_port
Content-Length: 244508
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_270_Request_005010X279A1
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
Batch
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0d91e6fa6
--XbCY
Content-Disposition: form-data; name="PayloadLength"
10240
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="UserName"
hospa
--XbCY
Content-Disposition: form-data; name="Password"
8y6dt3dd2
--XbCY
Content-Disposition: form-data; name="SenderID"
HospitalA
```

```
--XbCY
Content-Disposition: form-data; name="ReceiverID"
PayerB
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY
Content-Disposition: form-data; name="Checksum"
6A3FE55946
--XbCY
Content-Disposition: form-data; name="Payload"
<contents of batch file go here>
--XbCY--
```

Batch Submission Response message:

```
HTTP/1.1 200 OK
Content-Length: 2408
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_BatchReceiptConfirmation
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
Batch
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0c91e6da6
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="SenderID"
PayerB
--XbCY
Content-Disposition: form-data; name="ReceiverID"
HospitalA
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY
Content-Disposition: form-data; name="ErrorCode"
Success
--XbCY
Content-Disposition: form-data; name="ErrorMessage"
None
--XbCY--
```

Batch Submission Acknowledgement Retrieval Request message:

```
POST /core/eligibility HTTP/1.1
Host: server_host:server_port
```

```
Content-Length: 244508
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_999_RetrievalRequest_005010X231A1
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
Batch
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0d91e6fa6
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="UserName"
hospa
--XbCY
Content-Disposition: form-data; name="Password"
8y6dt3dd2
--XbCY
Content-Disposition: form-data; name="SenderID"
HospitalA
--XbCY
Content-Disposition: form-data; name="ReceiverID"
PayerB
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY--
```

Batch Submission Acknowledgement Retrieval Response message:

```
HTTP/1.1 200 OK
Content-Length: 12648
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_999_Response_005010X231A1
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
Batch
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0c91e6da6
--XbCY
Content-Disposition: form-data; name="PayloadLength"
10240
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="SenderID"
PayerB
--XbCY
```

```
Content-Disposition: form-data; name="ReceiverID"
HospitalA
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY
Content-Disposition: form-data; name="Checksum"
6A3FE55946
--XbCY
Content-Disposition: form-data; name="Payload"
<contents of batch file go here>
--XbCY
Content-Disposition: form-data; name="ErrorCode"
Success
--XbCY
Content-Disposition: form-data; name="ErrorMessage"
None
--XbCY--
```

Batch Results Retrieval Request message:

```
POST /core/eligibility HTTP/1.1
Host: server_host:server_port
Content-Length: 244508
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_005010_Request_Batch_Results_271
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
Batch
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0d91e6fa6
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="UserName"
hospa
--XbCY
Content-Disposition: form-data; name="Password"
8y6dt3dd2
--XbCY
Content-Disposition: form-data; name="SenderID"
HospitalA
--XbCY
Content-Disposition: form-data; name="ReceiverID"
PayerB
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY--
```

Batch Retrieval Response message:

```
HTTP/1.1 200 OK
Content-Length: 12648
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_271_Response_005010X279A1
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
Batch
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0c91e6da6
--XbCY
Content-Disposition: form-data; name="PayloadLength"
10240
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="SenderID"
PayerB
--XbCY
Content-Disposition: form-data; name="ReceiverID"
HospitalA
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
None
--XbCY
Content-Disposition: form-data; name="Checksum"
6A3FE55946
--XbCY
Content-Disposition: form-data; name="Payload"
<contents of batch file go here>
--XbCY
Content-Disposition: form-data; name="ErrorCode"
Success
--XbCY
Content-Disposition: form-data; name="ErrorMessage"
None
--XbCY--
```

Batch Results Acknowledgement Submission message:

```
POST /core/eligibility HTTP/1.1
Host: server_host:server_port
Content-Length: 244508
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
```

```
X12_999_SubmissionRequest_005010X231A1
12
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
Batch
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0d91e6fa6
--XbCY
Content-Disposition: form-data; name="PayloadLength"
10240
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="UserName"
hospa
--XbCY
Content-Disposition: form-data; name="Password"
8y6dt3dd2
--XbCY
Content-Disposition: form-data; name="SenderID"
HospitalA
--XbCY
Content-Disposition: form-data; name="ReceiverID"
PayerB
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY
Content-Disposition: form-data; name="Checksum"
6A3FE55946
--XbCY
Content-Disposition: form-data; name="Payload"
<contents of batch file go here>
--XbCY--
```

Batch Results Acknowledgement Submission Response message:

```
HTTP/1.1 200 OK
Content-Length: 2408
Content-Type: multipart/form-data; boundary=XbCY

--XbCY
Content-Disposition: form-data; name="PayloadType"
X12_Response_ConfirmReceiptReceived
--XbCY
Content-Disposition: form-data; name="ProcessingMode"
Batch
--XbCY
Content-Disposition: form-data; name="PayloadID"
f81d4fae-7dec-11d0-a765-00a0d91e6fa6
--XbCY
Content-Disposition: form-data; name="TimeStamp"
2007-08-30T10:20:34Z
--XbCY
Content-Disposition: form-data; name="SenderID"
```

```
PayerB
--XbCY
Content-Disposition: form-data; name="ReceiverID"
HospitalA
--XbCY
Content-Disposition: form-data; name="CORERuleVersion"
2.2.0
--XbCY
Content-Disposition: form-data; name="ErrorCode"
Success
--XbCY
Content-Disposition: form-data; name="ErrorMessage"
None
--XbCY--
```

7 SOAP

7.1 What is SOAP

SOAP is an XML-based messaging protocol that allows programs that run on disparate operating systems to communicate via HyperText Transfer Protocol (HTTP) and its Extensible Markup Language (XML). SOAP originally stood for Simple Object Access Protocol, but version 1.2 of the standard dropped this acronym.

HTTP is the underlying application layer protocol used by the World Wide Web. It defines how messages are formatted and transmitted and what actions servers and browsers should take in response to various commands, such as fetching a Web page from a Web server in response to entering a URL into your browser (the client). This independent request-response interaction makes HTTP stateless; each command is executed without knowledge of the commands that came before it, making it ideal as a carrier for IDE real time transactions.

SOAP was designed as an object-access protocol in 1998 by Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein for Microsoft, where Atkinson and Al-Ghosein were working at the time. Because it was championed by Microsoft it was initially met with skepticism by those who believed it was the software giant's attempt to hijack a piece of the internet. That view has since changed and SOAP is now recognized as a common standard; its development is guided by the XML Protocol Working Group of the World Wide Web Consortium (W3C), the organization that guides most important Web standards.

Since Web protocols are installed and available for use by all major operating system platforms, HTTP and XML provide an at-hand solution that allows programs running under different operating systems in a network to communicate with each other. SOAP specifies exactly how to encode an HTTP header and an XML file so that a program in one computer can call a program in another and pass along information. SOAP also specifies how the called program will return a response. Despite its frequent pairing with HTTP, SOAP supports other transport protocols as well, such as SMTP.

SOAP defines the XML-based message format that Web service-enabled applications use to communicate and inter-operate with each other over the Web. The heterogeneous environment and of the Web demands that applications support a common data encoding

protocol and message format. SOAP is a standard for encoding messages in XML that invoke functions in other applications. It is analogous to Remote Procedure Calls (RCP), used in many technologies such as DCOM and CORBA, but eliminates some of the complexities of using these interfaces. SOAP enables applications to call functions of other applications running on any hardware platform, regardless of their host operating systems or programming languages.

SOAP messages are more likely to get through firewall servers, since HTTP is typically Port 80 compliant, where other calls may be blocked for security reasons. Since HTTP requests are usually allowed to go through firewalls, no additional firewall or proxy configurations or exceptions are needed and programs using SOAP to communicate can be sure that the program can communicate with programs anywhere.

Anatomy

A message is sent via SOAP is in XML format, and is made up of four parts - and envelope, a header, a body, and a fault segment. The envelope encapsulates the message header and body and contains a variety of information required for processing the message, including a description of the kind of data to be found inside the envelope, and information about how that data should be processed. It also contains information about the sender and the recipient of the message.

Soap doesn't require that a message contain a header or fault segment, although as a practical matter messages will include the header when used in Web services. Information found in the header can perform a variety of functions, such as providing authentication. The data found in headers is organized into header blocks; there can be multiple blocks in a single header.

The body of the message is what contains the data. This might be a request for information or a response to a request for information. The data in the SOAP body is organized into sub-elements, of which there can be one or more.

The optional fault segment of the message contains any errors that occurred while processing the message.

The following is a simple example of a SOAP message requesting the stock price (`GetStockPrice`) of parameter `StockName` IBM:

Example message (simple):

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

The following is a response to the previous SOAP `GetStockPrice` request where the Stock `Price` of IBM is returned in the `GetStockPriceResponse` message:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

The following is a more complex message: a request for a travel reservation being sent

via SOAP. As you can see, the entire message itself is in XML format, as are all SOAP messages. Even if you are not able to understand all of what you see, if you take a few minutes to examine it, you'll be able to get a better sense of what each of the elements of a SOAP message do.

Example message (detailed):

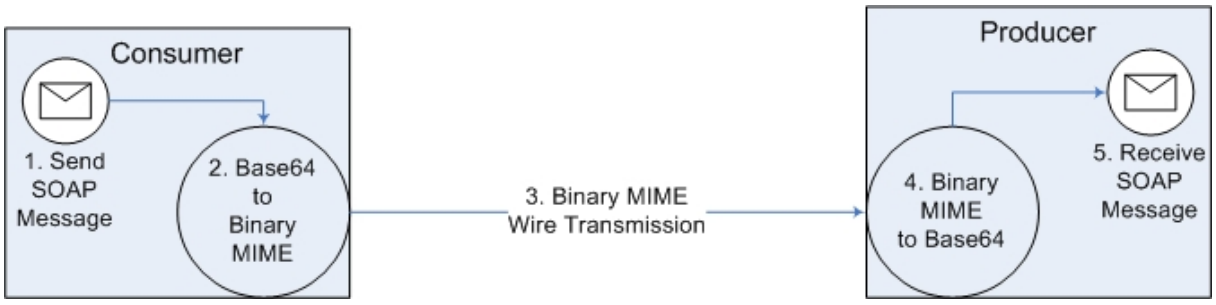
```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
    <env:Header>
        <m:reservation xmlns:m="http://travelcompany.example.org/reservation" env:actor="http://ww
            <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
            <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
        </m:reservation>
        <n:passenger xmlns:n="http://mycompany.example.com/employees" env:actor="http://www.w3.org
            <n:name>John Q. Public</n:name>
        </n:passenger>
    </env:Header>
    <env:Body>
        <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
            <p:departure>
                <p:departing>New York</p:departing>
                <p:arriving>Los Angeles</p:arriving>
                <p:departureDate>2001-12-14</p:departureDate>
                <p:departureTime>late afternoon</p:departureTime>
                <p:seatPreference>aisle</p:seatPreference>
            </p:departure>
            <p:return>
                <p:departing>Los Angeles</p:departing>
                <p:arriving>New York</p:arriving>
                <p:departureDate>2001-12-20</p:departureDate>
                <p:departureTime>mid morning</p:departureTime>
                <p:seatPreference/>
            </p:return>
        </p:itinerary>
        <q:lodging xmlns:q="http://travelcompany.example.org/reservation/hotels">
            <q:preference>none</q:preference>
        </q:lodging>
    </env:Body>
</env:Envelope>
```

7.2 What is MTOM

MTOM is a W3C (World Wide Web Consortium) Recommendation designed for optimizing the electronic transfer of attachments to and from web services. With web services-based SOA (Service Oriented Architecture) now being deployed across the world, transmitting attachments such as MRI Scans, X-Rays, Design Documents, and Business Contracts using SOAP messages has become a common practice. Through electronic transmission of documents, corporations can realize significant cost savings and better service levels by eliminating the use of postal mail. Paper-based manual tasks can be replaced with simple and efficient electronic processes where binary data can be

transmitted between organizations through standards such as MTOM.

MTOM provides an elegant mechanism of efficiently transmitting binary data, such as images, PDF files, MS Word documents, between systems. The Figure below shows the steps involved in transmitting data between a Consumer and Producer using MTOM.



SOAP transmissions encode binary data as BASE64. Sending it as BASE64 increases the size of the binary data. With MTOM, the SOAP messages are sent as MIME messages with the BASE64 encoding being replaced with a placeholder. The binary data is then placed between delimiters (which happens for each piece of binary data), and then placed at the end of the SOAP request, a reference . The binary data is then sent unencoded. MTOM also determines whether sending it as a MIME message will increase the size of the SOAP call and if doesn't provide a saving, it will send it as a normal SOAP message.

MTOM Process

The Consumer or Client application begins by sending a SOAP Message that contains complex data in Base64Binary encoded format. Base64Binary data type represents arbitrary data (e.g., Images, PDF files, Word Docs) in 65 textual characters that can be displayed as part of a SOAP Message element.

A sample SOAP Body with Base64Binary encoded element

```
<tns:data> is as follows:
<soap:Body>
 <tns:ByteEcho>
 <tns:data>JVBERi0xLjYNJeLjz9MNCjE+DQpzdGFyNCjEx0YNCg==</tns:data>
</tns:ByteEcho>
</soap:Body>
```

An MTOM-aware web services engine detects the presence of Base64Binary encoded data

types, <tns:data> in our example, and makes a decision – typically based on data size – to convert the Base64Binary data to MIME data with an XML-binary Optimization Package (xop) content type. The data conversion results in replacing the Base64Binary data with an `<xop:Include>` element that references the original raw bytes of the document being transmitted. The raw bytes are appended to the SOAP Message and are separated by a MIME boundary as shown below:

```
<soap:Envelope>
 <soap:Body>
  <tns:ByteEcho>
  <tns:data><xop:Include href="cid:1.63333845875937500@example.org"/></tns:data>
  </tns:ByteEcho>
 </soap:Body>
</soap:Envelope>

--MIMEBoundary000000
content-id: <1.63333845875937500@example.org>
content-type: application/octet-stream
content-transfer-encoding: binary
```

The raw binary data along with the SOAP Message and the MIME Boundary is transmitted over the wire to the Producer. The Producer then changes the raw binary data back to Base64Binary encoding for further processing. With this conversion between Base64Binary and raw binary MIME types, MTOM provides two significant advantages:

Efficient Transmission: Base64Binary encoded data is ~33% larger than raw byte transmission using MIME. MTOM therefore reduces data bloat by converting Base64Binary encoding to raw bytes for transmission.

Processing Simplicity: Base64Binary encoded data is composed of 65 textual characters. The data is represented within an element of a SOAP message. Security standards such as WS-Signatures and WS-Encryption can directly be applied to the SOAP Message. Once such operations are performed, the Base64Binary data can be converted to raw bytes for efficient transmission. Securing document transmission via SOAP, therefore, does not require additional standards for securing MIME-based attachments.

7.3 SOAP in CORE

This section contains examples of SOAP messages depicting the request and response required for patient eligibility checks.

The SOAP EDI messaging envelope contains metadata related to its payload and may sometimes not contain the payload it describes. Fields common to all the messaging

envelopes are:

- `PayloadType`: Identifies the kind of transaction the message is transmitting. Not all payload types are tied to a specific EDI transaction, some may be for confirmations of message arrival or error notifications.
- `ProcessingMode`: Identifies whether the message is to be processed in real time or batch modes.
- `PayloadID`: Unique identifier for the related payload. The payload does not need to be in the message; the message may be an acknowledgement of a received request.
- `TimeStamp`: Time in which the message was sent.
- `SenderID`: Identifies the sender of the message.
- `ReceiverID`: Identifies the recipient of the message.
- `CORERuleVersion`: Idenfities the CORE rules in effect. Currently 2.2.0.

Request messages will always contain a header through which the sender is verified as an authorized user of the receiving service. This verification can be done through username/password tokens or the use of an x.509 certificate. The examples provided use the former method, as does HIPAAsuite RealTime Server.

The response message will not contain a header, but does have two additional fields for error processing: `ErrorCode` and `ErrorMessage`.

The SOAP message envelopes for real time transactions are almost identical. Their only difference is the aforementioned distinction between request and response messages. The response in real time is meant to answer the request in full. The following are examples of a real time request and a real time response.

Real time Eligibility Request message:

```
POST /core/eligibility HTTP/1.1
Host: server_host:server_port
Content-Type: application/soap+xml; charset=UTF-8; action="RealTimeTransaction"

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Header>
```

```
        <wsse:Security
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext1.0.x
            <wsse:UsernameToken xmlns:wsu=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wssw
                <wsse:Username>bob</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-userr
            </wsse:UsernameToken>
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:COREEnvelopeRealTimeRequest
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType> X12_270_Request_005010X279A1</PayloadType>
            <ProcessingMode>RealTime</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>HospitalA</SenderID>
            <ReceiverID>PayerB</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
            <Payload><![CDATA[ISA*00* *00* *ZZ*NEHEN780 *ZZ*NEHEN003 ...IEA*1*000000031]]></Payloa
        </ns1:COREEnvelopeRealTimeRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

Real time Eligibility Response message:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; action="http://www.caqh.org/SOAP/WSDL/CORETransactions/RealTim

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Body>
        <ns1:COREEnvelopeRealTimeResponse
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0xsd">
            <PayloadType>X12_271_Response_005010X279A1</PayloadType>
            <ProcessingMode>RealTime</ProcessingMode>
            <PayloadID>a81d44ae-7dec-11d0-a765-00a0c91e6ba0</PayloadID>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>PayerB</SenderID>
            <ReceiverID>HospitalA</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
            <Payload><![CDATA[ISA*00* *00* *ZZ*NEHEN780 *ZZ*NEHEN003...IEA*1*000000031]]></Payload
            <ErrorCode>Success</ErrorCode>
            <ErrorMessage></ErrorMessage>
        </ns1:COREEnvelopeRealTimeResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP Batch messages also use MTOM to send the Payload file. The use of MTOM in Batch mode requests/responses creates multipart MIME messages in order to separate the payload from the other message elements even if there is no payload (which is the case for Batch Retrieval Requests, Batch Submission Acknowledgement Retrieval Requests, Batch Submission Responses, and Batch Results Acknowledgement Submission Responses)

or if the payload size is too small to justify the use of MTOM.

If WCF determines that the size of the overhead of a multipart header in addition to the encoded payload is less than the payload without MTOM encoding, the payload is instead referenced from the SOAP message and appended as an attachment of the multipart message.

In addition to the fields in the real time SOAP messages, batch SOAP messages, or MTOM-encoded SOAP messages, have the following fields whenever a payload is included in the message:

- **`PayloadLength`**: Defines the length of the actual payload in bytes.
- **`Checksum`**: An element used to allow the receiver to verify the integrity of the included payload.

Batch Submission message:

```
POST /core/eligibilityBatch HTTP/1.1
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614;
type="application/xop+xml"; start="0.urn:uuid:5117AAE1116EA8B87A1200060184615@apache.org"; startin

--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614
Content-Type: application/xop+xml; charset=UTF-8; type="application/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:5117AAE1116EA8B87A1200060184615@apache.org>

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Header>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
            <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
                <wsse:Username>bob</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-userr
            </wsse:UsernameToken>
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:COREEnvelopeBatchSubmission
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType>X12_270_Request_005010X279A1</PayloadType>
            <ProcessingMode>Batch</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <PayloadLength>1551254</PayloadLength>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>HospitalA</SenderID>
```

```
            <ReceiverID>PayerB</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
            <CheckSum>43B8485AB5</CheckSum>
            <Payload>
                <xop:Include href="cid:1.urn:uuid:5117AAE1116EA8B87A1200060184692@apache.org"
                xmlns:xop="http://www.w3.org/2004/08/xop/include" />
            </Payload>
        </ns1:COREEnvelopeBatchSubmission>
    </soapenv:Body>
</soapenv:Envelope>

--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <1.urn:uuid:5117AAE1116EA8B87A1200060184692@apache.org>

<Mixed batch file>

--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614--
```

The Generic Batch Submission Request uses the same request message as the Batch Submission Request message structure depicted above, with `PayloadType` values based on what is being submitted.

Batch Submission Response message:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339;
type="application/xop+xml"; start="0.urn:uuid:0B72121B1FEFA9BDD31200060195340@apache.org"; startir
action="http://www.caqh.org/SOAP/WSDL/CORETransactions/BatchSubmitTransactionResponse"

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339
Content-Type: application/xop+xml; charset=UTF-8; type="application/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:0B72121B1FEFA9BDD31200060195340@apache.org>

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Body>
        <ns1:COREEnvelopeBatchSubmissionResponse
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType>X12_BatchReceiptConfirmation</PayloadType>
            <ProcessingMode>Batch</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>PayerB</SenderID>
            <ReceiverID>HospitalA</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
            <ErrorCode>Success</ErrorCode>
            <ErrorMessage></ErrorMessage>
        </ns1:COREEnvelopeBatchSubmissionResponse>
    </soapenv:Body>
</soapenv:Envelope>
--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339--
```

Batch Submission Acknowledgement Retrieval Request message:

```
POST /core/eligibilityBatch HTTP/1.1
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614;
type="application/xop+xml"; start="0.urn:uuid:5117AAE1116EA8B87A1200060184615@apache.org"; startin

--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614
Content-Type: application/xop+xml; charset=UTF-8; type="application/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:5117AAE1116EA8B87A1200060184615@apache.org>

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Header>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
            <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
                <wsse:Username>bob</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-userr
            </wsse:UsernameToken>
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:COREEnvelopeBatchSubmissionAckRetrievalRequest
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType>X12_999_RetrievalRequest_005010X231A1</PayloadType>
            <ProcessingMode>Batch</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>HospitalA</SenderID>
            <ReceiverID>PayerB</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
        </ns1:COREEnvelopeBatchSubmissionAckRetrievalRequest>
    </soapenv:Body>
</soapenv:Envelope>
--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614--
```

Batch Submission Acknowledgement Retrieval Response message:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339;
type="application/xop+xml"; start="0.urn:uuid:0B72121B1FEFA9BDD31200060195340@apache.org"; startin

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339
Content-Type: application/xop+xml; charset=UTF-8; type="application/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:0B72121B1FEFA9BDD31200060195340@apache.org>

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Body>
        <ns1:COREEnvelopeBatchSubmissionAckRetrievalResponse
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType>X12_999_Response_005010X231A1</PayloadType>
            <ProcessingMode>Batch</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <PayloadLength>1551254</PayloadLength>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
```

```
            <SenderID>PayerB</SenderID>
            <ReceiverID>HospitalA</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
            <CheckSum>43B8485AB5</CheckSum>
            <Payload>
                <xop:Include href="cid:1.urn:uuid:5117AAE1116EA8B87A1200060184692@apache.org"
                xmlns:xop="http://www.w3.org/2004/08/xop/include" />
            </Payload>
            <ErrorCode>Success</ErrorCode>
            <ErrorMessage></ErrorMessage>
        </ns1:COREEnvelopeBatchSubmissionAckRetrievalResponse>
    </soapenv:Body>
</soapenv:Envelope>

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <1.urn:uuid:5117AAE1116EA8B87A1200060184692@apache.org>

<999 file>

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339--
```

Batch Results Retrieval Request message:

```
POST /core/eligibilityBatch HTTP/1.1
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614;
type="application/xop+xml"; start="0.urn:uuid:5117AAE1116EA8B87A1200060184615@apache.org"; startin

--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614
Content-Type: application/xop+xml; charset=UTF-8; type="application/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:5117AAE1116EA8B87A1200060184615@apache.org>

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Header>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
            <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
                <wsse:Username>bob</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-userr
            </wsse:UsernameToken>
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:COREEnvelopeBatchResultsRetrievalRequest
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType>X12_005010_Request_Batch_Results_271</PayloadType>
            <ProcessingMode>Batch</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>HospitalA</SenderID>
            <ReceiverID>PayerB</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
        </ns1:COREEnvelopeBatchResultsRetrievalRequest>
    </soapenv:Body>
</soapenv:Envelope>
--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614--
```

Batch Results Retrieval Response message:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339;
type="application/xop+xml"; start="0.urn:uuid:0B72121B1FEFA9BDD31200060195340@apache.org"; startir

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339
Content-Type: application/xop+xml; charset=UTF-8; type="application/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:0B72121B1FEFA9BDD31200060195340@apache.org>

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Body>
        <ns1:COREEnvelopeBatchResultsRetrievalResponse
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType>X12_271_Response_005010X279A1</PayloadType>
            <ProcessingMode>Batch</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <PayloadLength>1551254</PayloadLength>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>PayerB</SenderID>
            <ReceiverID>HospitalA</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
            <CheckSum>43B8485AB5</CheckSum>
            <Payload>
                <xop:Include href="cid:1.urn:uuid:5117AAE1116EA8B87A1200060184692@apache.org"
                xmlns:xop="http://www.w3.org/2004/08/xop/include" />
            </Payload>
            <ErrorCode>Success</ErrorCode>
            <ErrorMessage></ErrorMessage>
        </ns1:COREEnvelopeBatchResultsRetrievalResponse>
    </soapenv:Body>
</soapenv:Envelope>

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <1.urn:uuid:5117AAE1116EA8B87A1200060184692@apache.org>

<Response batch file>

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339--
```

Batch Results Acknowledgement Submission message:

```
POST /core/eligibilityBatch HTTP/1.1
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614;
type="application/xop+xml"; start="0.urn:uuid:5117AAE1116EA8B87A1200060184615@apache.org"; startir

--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614
Content-Type: application/xop+xml; charset=UTF-8; type="application/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:5117AAE1116EA8B87A1200060184615@apache.org>

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
```

```
    <soapenv:Header>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
            <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
                <wsse:Username>bob</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-userr
            </wsse:UsernameToken>
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:COREEnvelopeBatchResultsAckSubmission
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType>X12_999_SubmissionRequest_005010X231A1</PayloadType>
            <ProcessingMode>Batch</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <PayloadLength>1551254</PayloadLength>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>HospitalA</SenderID>
            <ReceiverID>PayerB</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
            <CheckSum>43B8485AB5</CheckSum>
            <Payload>
                <xop:Include href="cid:1.urn:uuid:5117AAE1116EA8B87A1200060184692@apache.org"
                xmlns:xop="http://www.w3.org/2004/08/xop/include" />
            </Payload>
        </ns1:COREEnvelopeBatchResultsAckSubmission>
    </soapenv:Body>
</soapenv:Envelope>

--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <1.urn:uuid:5117AAE1116EA8B87A1200060184692@apache.org>

<999 file>

--MIMEBoundaryurn_uuid_5117AAE1116EA8B87A1200060184614--
```

Batch Results Acknowledgement Submission Response message:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339;
type="application/xop+xml"; start="0.urn:uuid:0B72121B1FEFA9BDD31200060195340@apache.org"; startin

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339
Content-Type: application/xop+xml; charset=UTF-8; type="application/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:0B72121B1FEFA9BDD31200060195340@apache.org>

<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Body>
        <ns1:COREEnvelopeBatchResultsAckSubmissionResponse
        xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
            <PayloadType>X12_Response_ConfirmReceiptReceived</PayloadType>
            <ProcessingMode>Batch</ProcessingMode>
            <PayloadID>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</PayloadID>
            <TimeStamp>2007-08-30T10:20:34Z</TimeStamp>
            <SenderID>PayerB</SenderID>
```

```
            <ReceiverID>HospitalA</ReceiverID>
            <CORERuleVersion>2.2.0</CORERuleVersion>
            <ErrorCode>Success</ErrorCode>
            <ErrorMessage></ErrorMessage>
        </ns1:COREEnvelopeBatchResultsAckSubmissionResponse>
    </soapenv:Body>
</soapenv:Envelope>

--MIMEBoundaryurn_uuid_0B72121B1FEFA9BDD31200060195339--
```

8 Testing

8.1 Testing SSL

Testing with the secure ports with a test SSL Certificate (one not signed by a certification authority) requires some setup.

Attempting to communicate via https with an unsecured RealTime server will result in the following error:

```
CommunicationException: An error occurred while making the HTTP request to https://
<serveraddress>:<port>/<addressextension>. This could be due to the fact that the
server certificate is not configured properly with HTTP.SYS in the HTTPS case. This
could also be caused by a mismatch of the security binding between the client and the
server.
```

This means the port must be configured with an SSL certificate. Without an SSL certificate signed by a major certification authority, we need to self-sign our own. Self-signed certificates are not intended for use in a production environment and should only be used for testing.

First, we need a self-signed certificate to bind to the port we will be using. Otherwise, we will run into the communication exception displayed above. Create the certificate using makecert.exe, which tends to be in the Windows SDK folder. The self-signed server certificate we will be binding first needs a root certificate to use for signing. Using makecert, use the following command to create a root certificate; `<machineName>` can be localhost:

```
makecert.exe -sk RootCA -sky signature -pe -n CN=<machineName> -r -sr LocalMachine -ss
Root MyCA.cer
```

This creates a certificate named MyCA.cer. You should see that file in the working directory from where you ran makecert.exe.

Next we will create the server certificate. `<certificate path>` should be something to use to identify the certificate.

```
makecert.exe -sk server -sky exchange -pe -n CN=<machineName> -ir LocalMachine -is
Root -ic MyCA.cer -sr LocalMachine -ss My <certificate path>
```

Again, you should see the resulting file in the working directory from where you ran

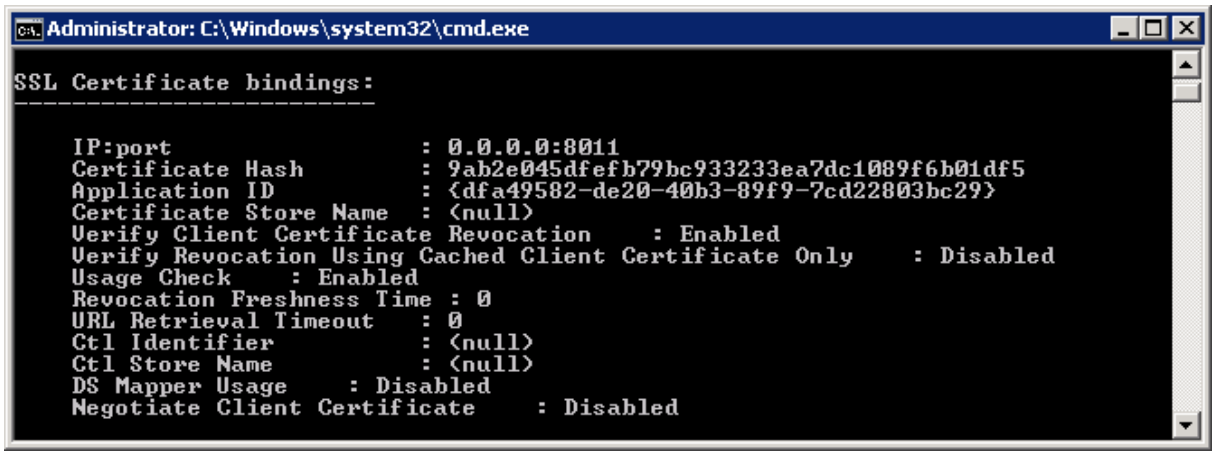
makecert.exe. This is the server certificate, so we are now ready to bind the certificate to the port. You can do this by command line or by using the RealTime Server.

Verifying certificates bound to a port:

Administrative privilege is not required to view certificates, any user can see them. To view a certificate bound to a specific `<port>`:

```
netsh http show sslcert ipport=0.0.0.0:<port>
```

This command will present any SSL certificates bound to the specified `<port>`:



Example of a self-signed certificate bound to port 8011.

8.1.1 MakeCert

MakeCert creates an X.509 certificate signed by the test root key or other specified key, that binds your name to the public part of the key pair. The certificate is saved to file, a system certificate store, or both. The tool is in the \Bin folder of the Microsoft SDK (Software Development Kit) installation path, typically something similar to \Program Files\Microsoft SDKs\Windows\v7.0A\Bin.

As part of the Microsoft SDK, MakeCert can be downloaded freely from http://www.microsoft.com/en-us/download/details.aspx?id=8279 and run from the command prompt.

Note that the commands used to bind the certificate to port do not use the certificate file itself. Instead the thumbprint is looked up in your certificate key store. The Testing SSL section sample commands store the key in your keystore through the use of the -sr (certificate store location) and -ss (certificate store name) flags.

8.2 Testing with a Trading Partner

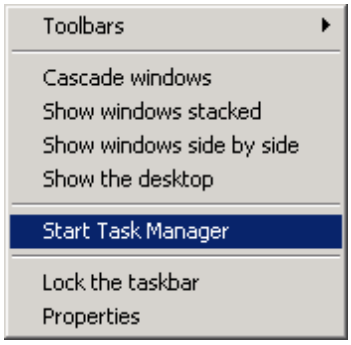
When you first deploy the HIPAAsuite RealTime Server, you will need to experiment with the application. You will learn to know how it works and you will see how the information flow is configured. In debug mode, you will see messages on all levels of the process helping you to trouble shoot issues.

Finally we explain the use of some helpful tools to work with the minute details of packets and SOAP testing applications.

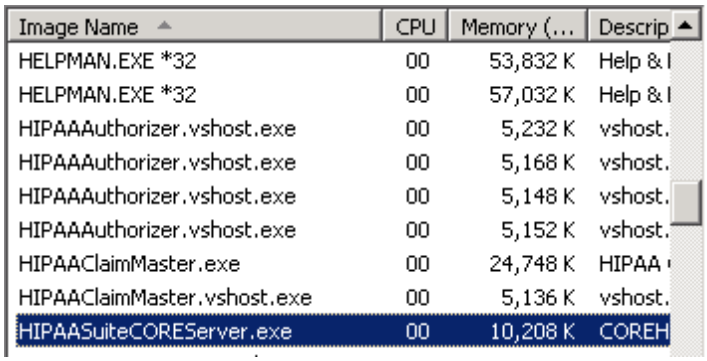
A system administrator should be able to trouble shoot and re-configure the server so that the outcome is predictable. By using the HIPAAsuite RealTime Server's testing procedures you will be able to do just that.

8.3 Debug Mode

HIPAAsuite RealTime Server is, as mentioned, a formless application. There is normally no user interface with feedback to the user. In fact for efficiency reasons this is not even desired. But this fact makes is also hard to see what is going on in this black box. Sure, one can see that the process is running in the task manager:

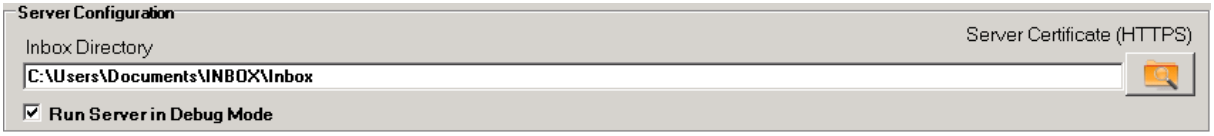


Right-clicking the taskbar is a quick way to access the Windows Task Manager.



HIPAAsuite RealTime Server listed in the Windows Task Manager.

The log also provides a history of activity. Otherwise, it is a riddle what is going on inside. Especially when setting up new trading partners or trouble shooting issues this is not enough. For this reason, the HIPAAsuite RealTime Server provides the option to show a debug console. The debug console will output to a dedicated console window incoming and outgoing messages. This includes their contents and user authentication. When the debug option is selected in the config menu,



Debug Mode checkbox.

a console window is opened the next time HIPAAsuite RealTime Server is run. And one line of console text informs us that the RealTime Server is now running in debug mode.

```
The service is ready in debug mode..
```

This is the line of text you should see in the console window.

When we designed the HIPAAsuite RealTime Server we had the goal to provide as much relevant information as possible about each step the server is taking while not overwhelming the observer. In the next paragraphs we will show you what the debug screen looks like and what information it contains.

If a new request comes in the process starts with a time stamp. The message parser does its work and the message parameters and payload are displayed. Use of depersonalized test data is recommended when using the debug console to test.

> Debug mode will put privileged medical information on the screen. Make sure that all the protocols of HIPAA Privacy and Security are followed and that staff is informed about the obligation under this law.

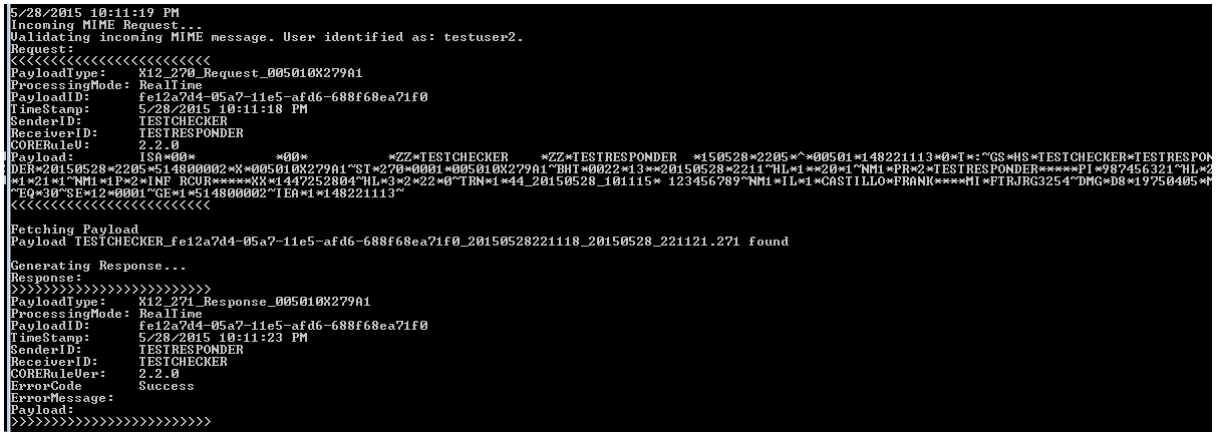
Upon receiving a request, the console will output the time it was received in and the username of the Trading Partner that initiated the transaction. Once the Trading Partner is authenticated, the request body is presented (excluding credentials). The HIPAA RealTime Server will then call the appropriate HIPAAsuite application to process the incoming

When the process is finished the response file will be read and turned into the payload. The message envelope is built and again displayed in the command window.

RealTime Example

The following is an example of what a real time SOAP-formatted general 270 Eligibility Request message and immediate 271 Eligibility Response message for testbed data

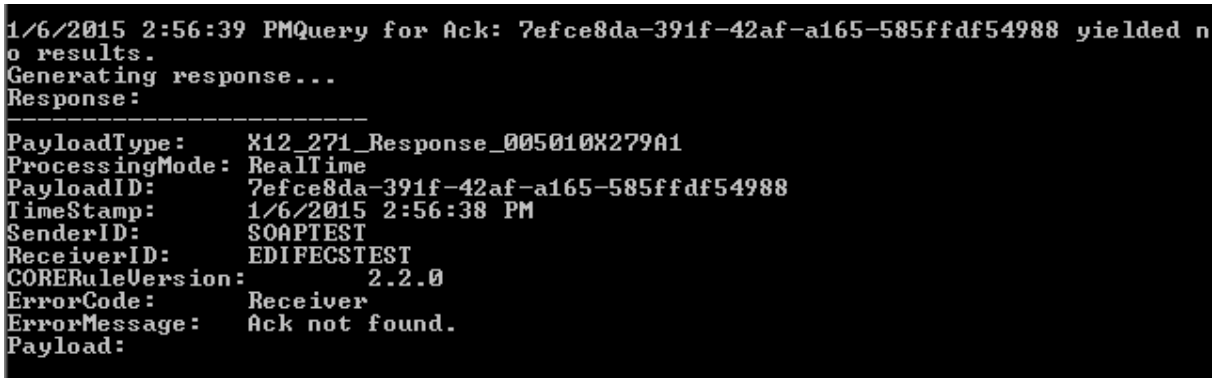
member Castillo, Frank looks like in debug mode with the "Show Payload" option enabled:



HIPAAsuite RealTime Server receives a test 270 SOAP real-time request

> We do not recommend running HIPAAsuite RealTime Server in Debug mode in production since this console interaction affects the performance of the server

Any error messages that the server throws are listed as well.



By using HIPAAsuite RealTime Server's debug mode, you can effectively observe the flow of data to the service. If there are problems you can pin point to what transaction, process or set-up needs troubleshooting and verify that everything is working as intended.

Batch Example

The following is a complete batch mode 270 Eligibility transaction as seen in debug mode with the "Show Payloads" option enabled for testbed data user Castillo, Frank starting with the initial request and its receipt confirmation.

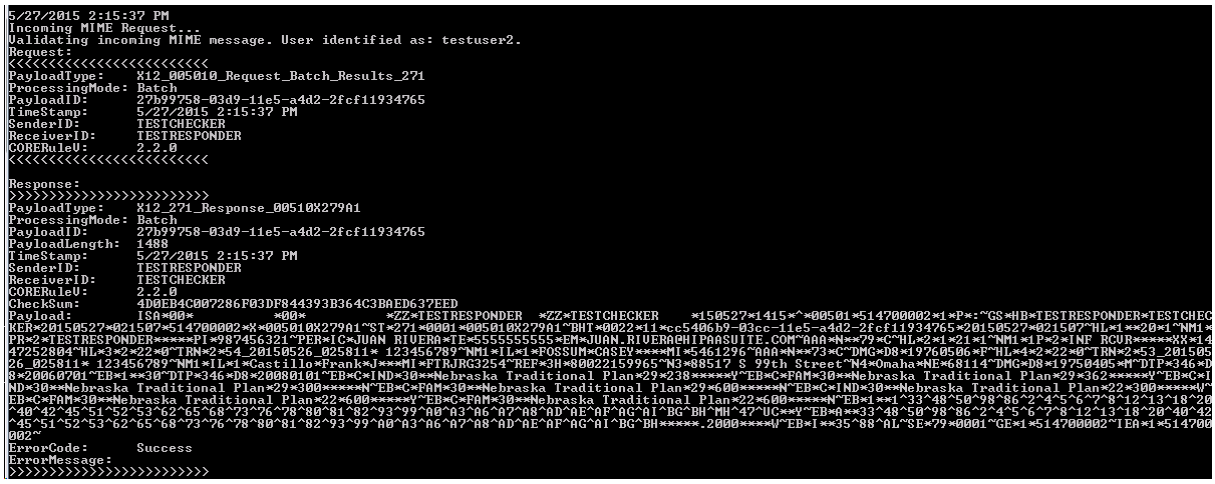
A batch submission request and subsequent receipt confirmation response.

The client is always initiating the transaction and always needs to validate itself using a user name token or X.509 certificate. (to be implemented later)

In batch mode the request is not fulfilled immediately as with real time, but is answered with a batch receipt confirmation. The following is a request for the 999 associated with the 270 Eligibility request submitted in the previous example and its subsequent 999 response:

A 999 request in SOAP

A 271 batch request for the previously sent 270 batch request and the subsequent 271 response:



A batch result request in debug mode

Finally, a 999 acknowledgment submission for the receipt of the above 271 batch results and subsequent acknowledgment receipt confirmation:

The final step in the batch process, the batch results receipt submission.

8.4 Log

Any action the server takes might run too quickly past the debug mode screen. The log file stores data on incoming and outgoing transactions as well as server startup notices and timestamps associated with the transactions. HIPAAsuite RealTime Server logs any action taken relating to the SOAP or MIME envelope of an incoming or outgoing EDI file and the database connections used to find and link them to their response or request as appropriate. Any action on the contents of the EDI files is logged by the approprate HIPAAsuite responder application.

The log files are stored in your HIPAAsuite RealTime Server directory under \log. The format used to name the log files is MM-dd-yyyy.txt, providing a daily log file for quick perusal.

Contents of a HIPAAsuite RealTime Server log file after startup and a received EDI transaction request:

```
       -----------------
03:12:55 PM: Starting Server
       Loading settings
       Settings loaded.
       Secure host open.
       Test host open.
       The service is ready in debug mode.
03:25:43 PM: Validating incoming SOAP message. User identified as: USERNAME.
       Opening connection to DB.
       User validated.
       Processing TEST in RealTime.
       Payload not found. Fetching ACK.
       03:25:46 PM: Response sent.
```

8.5 Test Port

For security and privacy reasons it is absolutely necessary to use the secure socket layer transport protocol or HTTPS with the HIPAAsuite RealTime Server. This mechanism guarantees that the packets that are transported between provider and payer are uniquely encrypted and nobody and break-in and steal information. HTTPS is widely used and accepted.

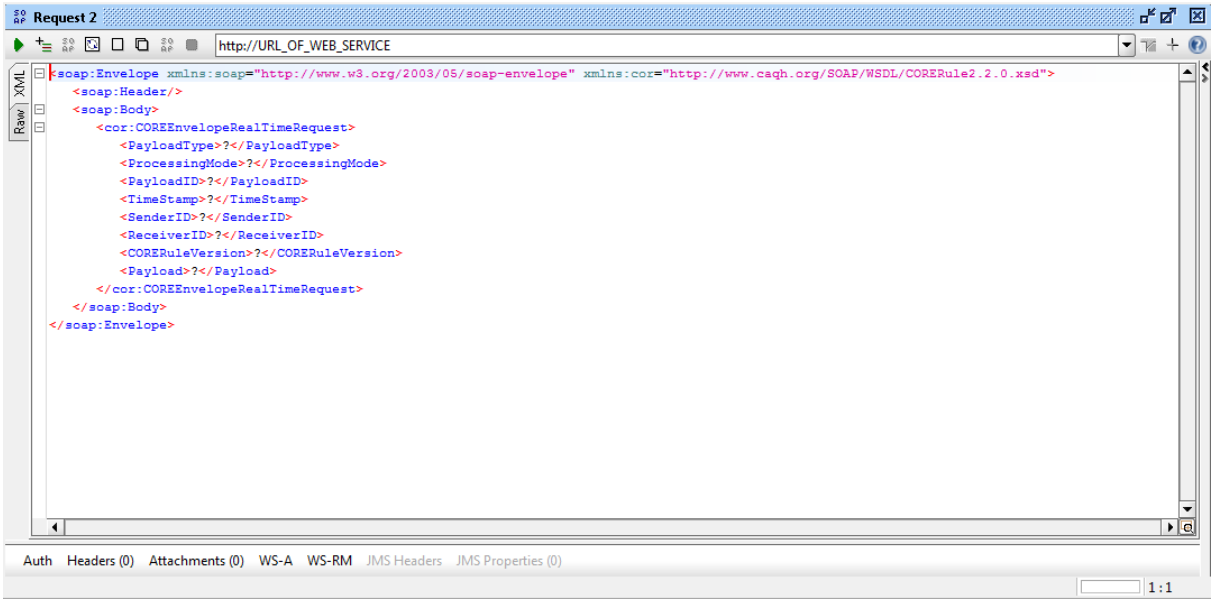
But sometimes for debugging reasons it is very inconvenient to have only garbled nonsense in the packets that travel between the server and client. The HIPAAsuite RealTime Server has for this reason an option to use an unsecured, straight HTTP port. This is an option and has to be turned on in the configuration window. The test port uses the same endpoints as its secure counterpart but is accessed with http:// instead of https://

8.6 Using SOAP UI

One of the essential tools for anyone working with SOAP transactions is SOAPUI. SoapUI is a free and open source cross-platform Functional Testing solution. With an easy-to-use graphical interface, and enterprise-class features, SoapUI allows you to easily and rapidly create and execute automated functional, regression, compliance, and load tests. In a single test environment, SoapUI provides complete test coverage and supports all the standard protocols and technologies. SoapUI is the world's most complete testing tool! You can download SOAPUI here

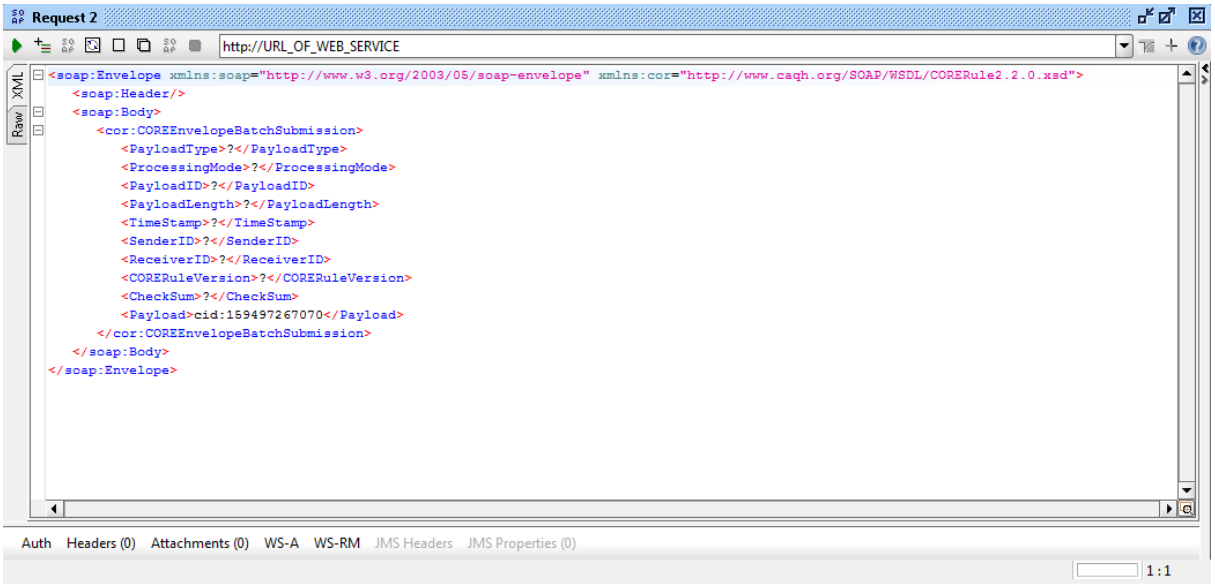
It would be too much for this documentation to explain SOAPUI, but in the following we want to share a typical test scenario.

The following are examples of SOAP request messages for real time and batch EDI transactions generated by SOAP \UI from the WSDL provided by CAQH here. All fields are blank and must be filled in by the user. The header authentication information (whether by username/password token or X.509 certificate) can be set up to be attached when sending the SOAP message or written directly to the request window within the `<soap:Header>` element. If the header authentication information is set up in SOAP UI's WS-Security Configuration, the header will not be displayed in the XML view of the request window, but will be present in Raw message view and in the SOAP message sent to the address specified in the address bar.



Sample real time SOAP request message generated by SOAPUI.

The following is a sample batch EDI transaction request generated by SOAPUI. As the above example, the metadata and payload fields are left to the user to fill in. Header information may be added directly to the request window within the <soap:Header> element or configured using WS-Security Configuration in SOAP UI project settings. If using the latter option, the header authentication information will only be displayed in the Raw view of the message request window.



Sample batch SOAP request message generated by SOAP UI.

The message will be serialized and sent via HTTP as a SOAP message to the URL in the address bar. SOAPUI will wait for a SOAP response and display one if received.

Lacking the authorization header configuration, the header may be included in the request sent by SOAPUI and be sent correctly. The following is an example of one such case. The payload, a 270 Eligibility Request, is omitted for simplicity:

```
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
<soapenv:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="true">
<wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-164844">
<wsse:Username>USERNAME</wsse:Username>
<wsse:Password
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">
PASSWORD</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
</soapenv:Header>
<soapenv:Body>
<ns1:COREEnvelopeRealTimeRequest xmlns:ns1="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
<PayloadType>X12_270_Request_005010X279A1</PayloadType>
<ProcessingMode>RealTime</ProcessingMode>
<PayloadID>7efce8da-391f-42af-a165-585ffdf54988</PayloadID>
<TimeStamp>2014-05-28T04:21:41Z</TimeStamp>
<SenderID>EDIFECSTEST</SenderID>
<ReceiverID>SOAPTEST</ReceiverID>
<CORERuleVersion>2.2.0</CORERuleVersion>
<Payload>
contents of file go here...
</Payload>
</ns1:COREEnvelopeRealTimeRequest>
</soapenv:Body>
</soapenv:Envelope>
```

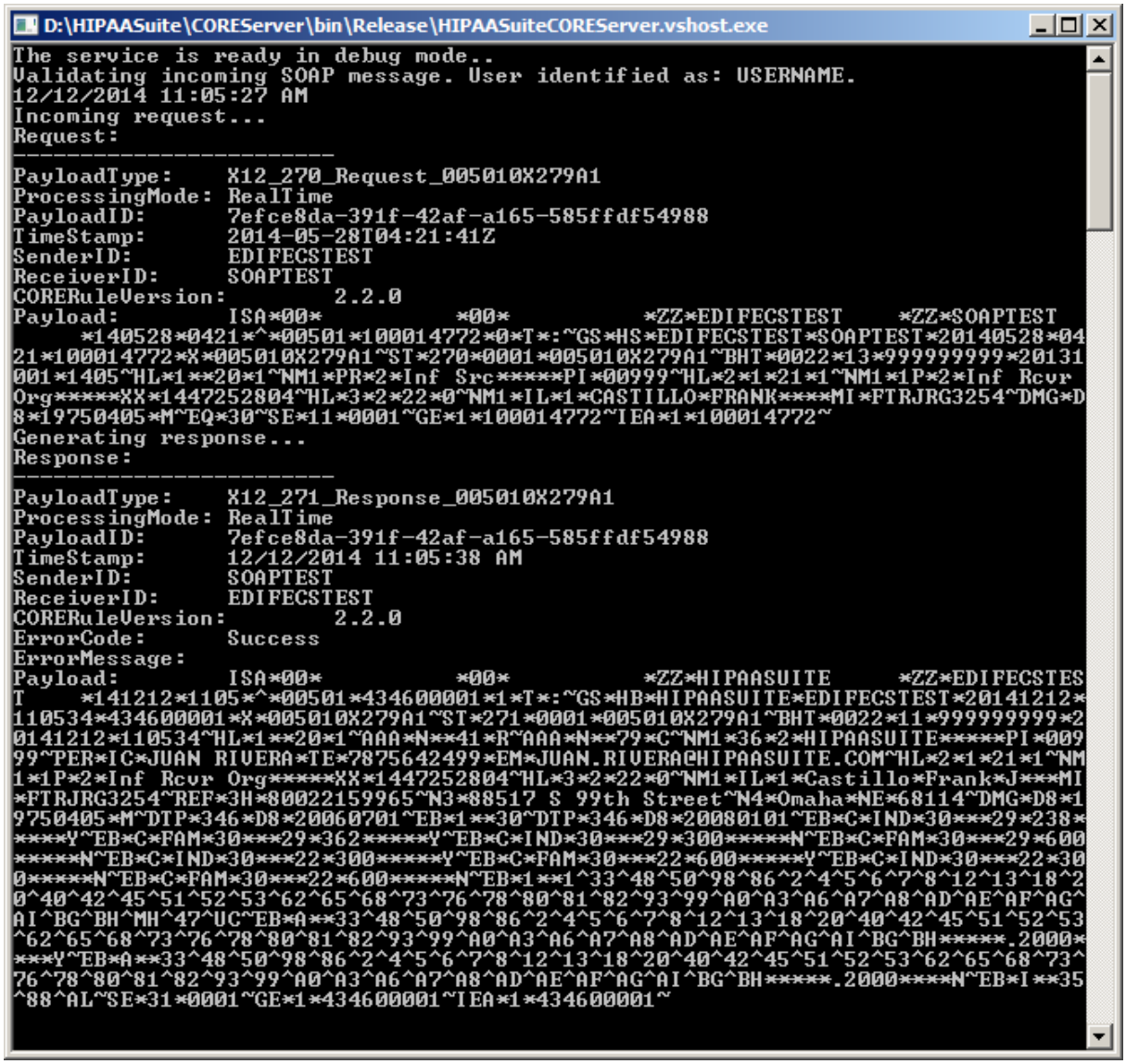
The 270 request, payload omitted for simplicity.

A response to the above 271 Eligibility Request message as displayed by SOAPUI:

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
   <s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <COREEnvelopeRealTimeResponse xmlns="http://www.caqh.org/SOAP/WSDL/CORERule2.2.0.xsd">
         <PayloadType xmlns="">X12_271_Response_005010X279A1</PayloadType>
         <ProcessingMode xmlns="">RealTime</ProcessingMode>
         <PayloadID xmlns="">7efce8da-391f-42af-a165-585ffdf54988</PayloadID>
         <TimeStamp xmlns="">12/12/2014 11:05:38 AM</TimeStamp>
         <SenderID xmlns="">SOAPTEST</SenderID>
         <ReceiverID xmlns="">EDIFECSTEST</ReceiverID>
         <CORERuleVersion xmlns="">2.2.0</CORERuleVersion>
         <Payload xmlns="">contents of file go here...</Payload>
         <ErrorCode xmlns="">Success</ErrorCode>
         <ErrorMessage xmlns=""/>
      </COREEnvelopeRealTimeResponse>
   </s:Body>
</s:Envelope>
```

The 271 response, payload omitted for simplicity.

HIPAAsuite RealTime Server's debug console allows you to see HIPAAsuite RealTime Server's actions in real time. The transactions are displayed as-is along with the metadata packaged in their messaging envelopes. The following is the transaction presented above as displayed in the debug console with the payload in full:



HIPAAsuite RealTime Server debug console displaying a 270 request received the 271 response sent back.

8.7 Using Packet sniffers

When testing web server traffic it is sometimes extremely helpful to see the raw data packages that are exchanged between the server and the client. But inspecting individual internet packages is not a trivial task. The good news is that there are free software products out there to do just that. We used Wireshark. Wireshark is the world's foremost network protocol analyzer. It lets you see what's happening on your network at a microscopic level. It is the de facto (and often de jure) standard across many industries and educational institutions.

Wireshark development thrives thanks to the contributions of networking experts across

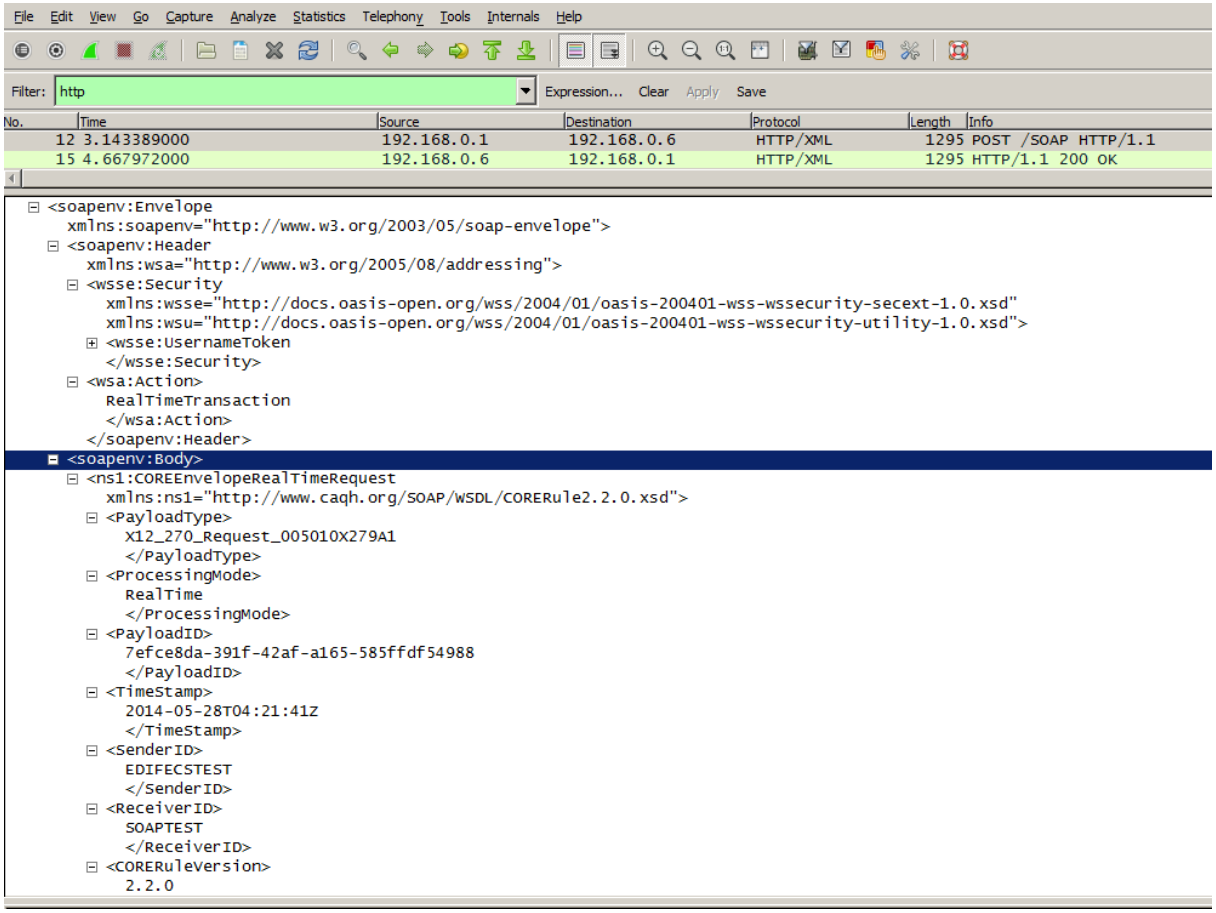
the globe. It is the continuation of a project that started in 1998 and it is free. You can download it here.

Again it would be too much for this documentation to explain Wireshark, but in the following we want to showcase a few steps that we took.

After selecting the appropriate interface in Wireshark, you will see all packets being transmitted through the selected interface. This includes HTTPS traffic, which can be decrypted provided you have the private key in a decrypted PKCS#8 PEM format. The binary DER format cannot be used with Wireshark. This is a useful feature, but to control the packet flow and avoid unnecessary setup we suggest you use the test port provided by HIPAAsuite RealTime Server.

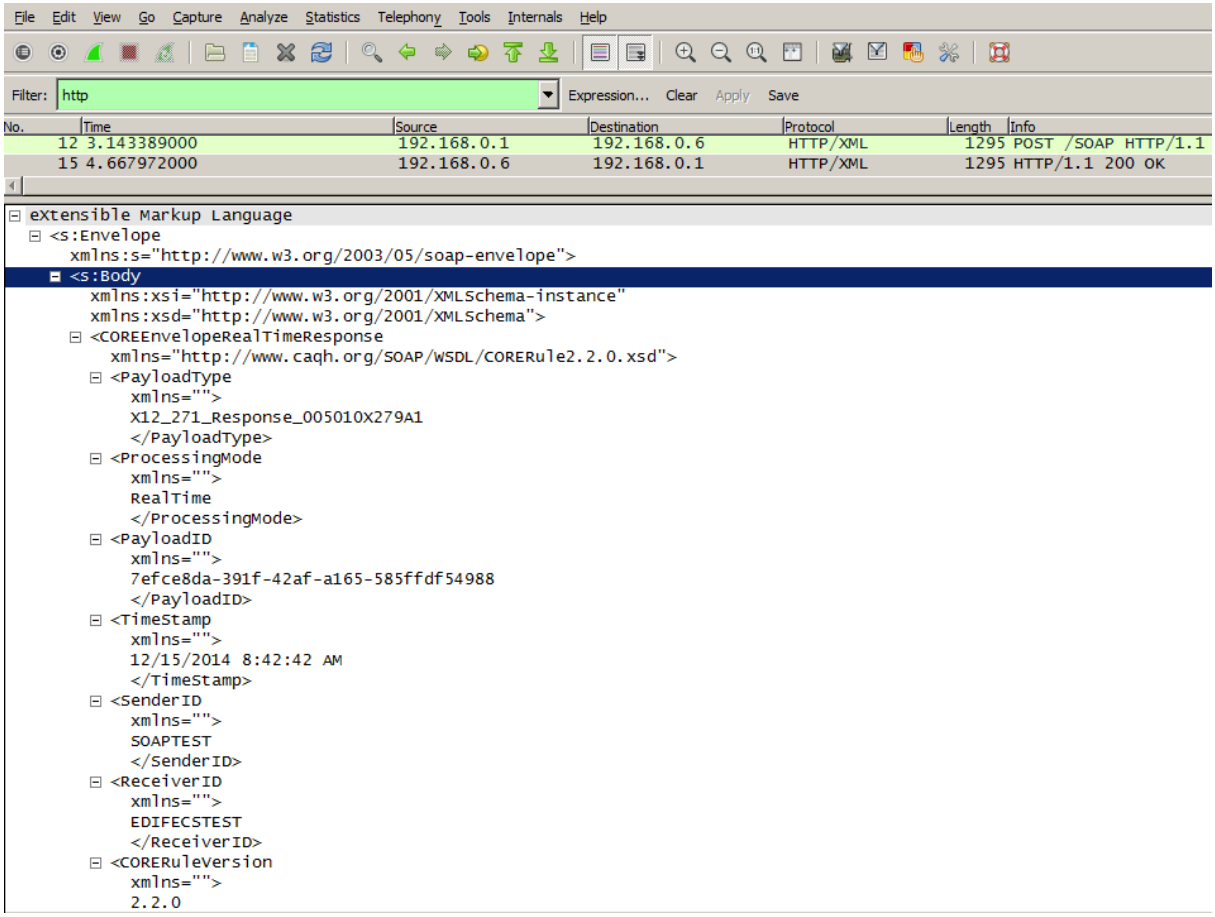
Wireshark will present you with all network traffic flowing over the selected interface. Filtering by HTTP or XML will allow you to quickly locate the SOAP packets. The following example packets were captured by Wireshark while monitoring HIPAAsuite RealTime Server's traffic over the HTTP test port.

270 Eligibility Request message:



Sample 270 request as captured by Wireshark. Note the Header containing username/password authentication data and the body containing EDI transfer metadata.

271 Eligibility Response message:



Sample 271 response as captured by Wireshark. Note the body containing EDI transfer metadata.

These examples show the messaging structure and all elements contained within as they are sent to and from HIPAAsuite RealTime Server. Wireshark is a useful tool to monitor the contents of the packets being transmitted.

Back Cover