Commons eXchange Protocol

Version 1.0

Commons eXchange Protocol 1
Overview 2
Protocol 3
Possible Architectures 4
Architecture A: System registry and repository 4
Architecture B: Simple End-to-End communication 4
Architecture C: System with multiple registries 5
API 6
Data Definitions 6
CXP Messages 8
Security 9
Data Encryption 9
Identity 9
HIPAA 9
Trademarks and Intellectual Property 10
Appendix A: Changes history of CXP Protocol 11
Changes since 0.8 11
Changes since 0.9 11
Changes since 0.9.2 11
Changes since 0.9.3 11
Changes since 0.9.9 11
Appendix B – CXP SOAP WDSL 12
Appendix C: Example code 20
Example C# PUT call 20
Example C# GET Call 21
Appendix D: Normative Security/Identity Recommendations 22
Appendix E: Normative RegistryParameters Data Dictionary 23
Appendix F: Future directions 23

Overview

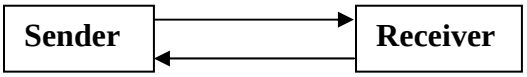
CXP is a TCP (SOAP) based end to end protocol allowing two parties to structure a spontaneous and reliable constrained exchange of standards-based, patient-focused healthcare information. It can be utilized over plain http or SSL, or preferably over locked-down certificate based TLS connections, with increasing levels of trust.

CXP may be used as a low-complexity, simple mechanism for document storage/retrieval with other types of registries and repositories. The use of multiple registry parameter blocks permits opt-in on the part of the call initiator to other repository models such as XDS.

The simplicity of the base protocol is intentional. Other services and protocols will be built on this foundation.

Protocol

CXP is a two party point to point protocol that moves CCRs and other documents across the Internet between co-operating Sending and Receiving systems from multiple vendors. It is designed to be a very lightweight protocol while being able to support different vendors and different system architectures.



All messages in CXP are synchronous and stateless. The Sender initiates the session; the Receiver responds. The protocol that the messages use is SOAP 1.1.

CXP contains three commands PUT, GET, and DELETE; while all of them need to be supported in the WDSL not all need to be supported by any particular implementation. The mechanism for this is described below.

PUT places a CCR into a Receiving system. There are no guarantees for what a system does with the document; it might be stored, digested, indexed depending on the policies and configuration of the Receiving server.

GET retrieves a document based on its identifier (which was returned by PUT) or optionally by metadata (subject to the configurations and policies of the Receiving system)

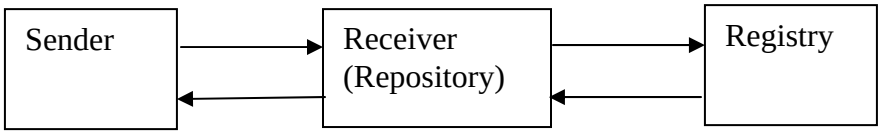
DELETE deletes a document based on its identifier (which was returned by PUT) or optionally by metadata (subject to the configurations and policies of the Receiving system).

A CXP service might support PUT but not GET or DELETE. For example – if the service was a desktop application that accepted the CCRs might parse them, use the contents, and then discard the CCR. In this case – the status code from any GET or DELETE messages would be 501 – Not Implemented (identical to HTTP).

Possible Architectures

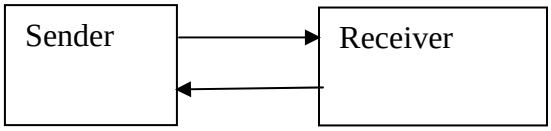
Although CXP does not mandate any type of architecture for the service endpoints, this section will describe three possibilities that could be implemented to illustrate how the protocol can be used.

Architecture A: System registry and repository



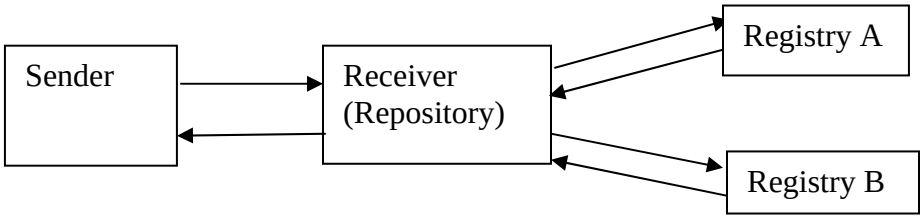
This model assumes that the sender wishes to store data on a repository with separate registry.

Architecture B: Simple End-to-End communication



The simplest architecture is a simple end-to-end message. This is especially appropriate for two devices sharing information without any need of a server or special registry.

Architecture C: System with multiple registries



Using CXP with multiple registries can be very useful for simple devices to interact with more complex repository models. For example – if the sender (the issuer of a PUT command) specified two registries with different structures (Registry A might use only opaque or voluntary identifiers with no patient metadata federation, Registry B might be an XDS registry which required ample metadata and federation across multiple MPIs) the sender can effectively 'opt-in' to both registries. This permits the sender to use a very simple protocol to interact with several registries with different policies.

The only differences in architectures A, B, and C is in the registry parameter blocks and the authentication layer; the core CXP commands remain the same.

API

All of the services have optional registry parameters. These registry parameters are always specified within an array.

Data Definitions

RegistryParameters object contains the following fields:

- String RegistryIdentifier
- String RegistryName
- String version – this is the version of CXP.
- List of name/value pairs.

RegistryParameters are used by several commands on input and output. Different implementations may have different sets of name/value pairs. The version number is the version of CXP; if there are versions of other software components they can be included in the name/value pairs.

The PutResponse object is defined to be

- String guid[1]
- Integer status
- String reason
- (optional) RegistryParameters parameters

The GetResponse object is defined to be

- String guid
- Integer status
- String reason
- String content-type
- String content
- (optional) RegistryParameters parameters

The Attachment object is defined to be

- String guid
- String content-type
- String content

[1] A guid is a global document identifier. In CXP it is modeled as a string.

Status code

The status code is an integer. This table defines what the values mean; a future version of this specification will define specific codes. The meaning of any particular code should be obvious from the <Reason> text. The meanings of the codes should echo the W3C recommendations[2] when possible. A short overview follows.

Code range	Meaning
200-299	Success. 200 means completely successful; other values in this range may have warning or informational messages in the reason field.
400-499	Bad request (client error)
500-599	Server error. Two special cases 500 Internal Server Errror 501 Not supported

Any CXP implementation may define additional status codes within this broad framework.

Content Types

Table: Defined Content Types

Content type	**Document type**
application/pdf	PDF document
application/dicom	DICOM Series
application/x-ccr+xml	CCR

[2] see http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

CXP Messages

PUT

Put places a CCR on the Receiving server. It returns success only after the document has been successfully received by the Receiver. There are three versions of PUT:
Response put(String ccr)
Response put(String ccr, RegistryParameters[] parameters)
Response put(String ccr, RegistryParameters[] parameters, Attachments[] attachments[3])

GET

GetResponse get(String guid)
GetResponse get(RegistryParameters [] parameters)
GetResponse get(String guid, RegistryParameters[] parameters)

DELETE

DeleteResponse DELETE(String guid)
DeleteResponse DELETE(RegistryParameters [] parameters)
DeleteResponse DELETE(String guid, RegistryParameters [] parameters)

Note that there are no general query mechanisms in CXP for (say) all of the documents belonging to a patient or all of the documents belonging to a doctor. A query of this type could be supported by a registry – but that happens outside the scope of the CCR. By placing registry parameters in a second (optional) block we permit multiple business models.

[3] This version of CXP places file attachments as base64 encoded entities; the next version will use MTOM.

Security

Data Encryption

There are two types of data encryption supported here. The first is session encryption. Session encryption takes place via SSL or TLS. It is recommended that all clinical data be transmitted over encrypted channels such as these.

The second type of encryption is data object encryption – such as a CCR with encrypted sections per the upcoming ASTM specification. This data can be handled via CXP but the storage and transmission of keys is outside of the CXP specification. Implementations may refuse encrypted CCRs or they may require additional registry parameter entries to process encrypted content.

Identity

Authentication happens outside of CXP. There are potentially two separate notions of identity that may be modeled in the future:

- Identity of the information system
- Identity of the sender (the human)

HIPAA

The CXP sender must accept responsibility for release of information to the recipient. Patient consent to information release is the responsibility of the sender.[4]. Consent and the key management are outside of CXP. Another example might be implied consent between two CXP systems that have a HIPAA covered entity or business associate relationship and that authenticate each other via TLS. The HIPAA relationship is beyond CXP. Other consent mechanisms may exist, all outside CXP.

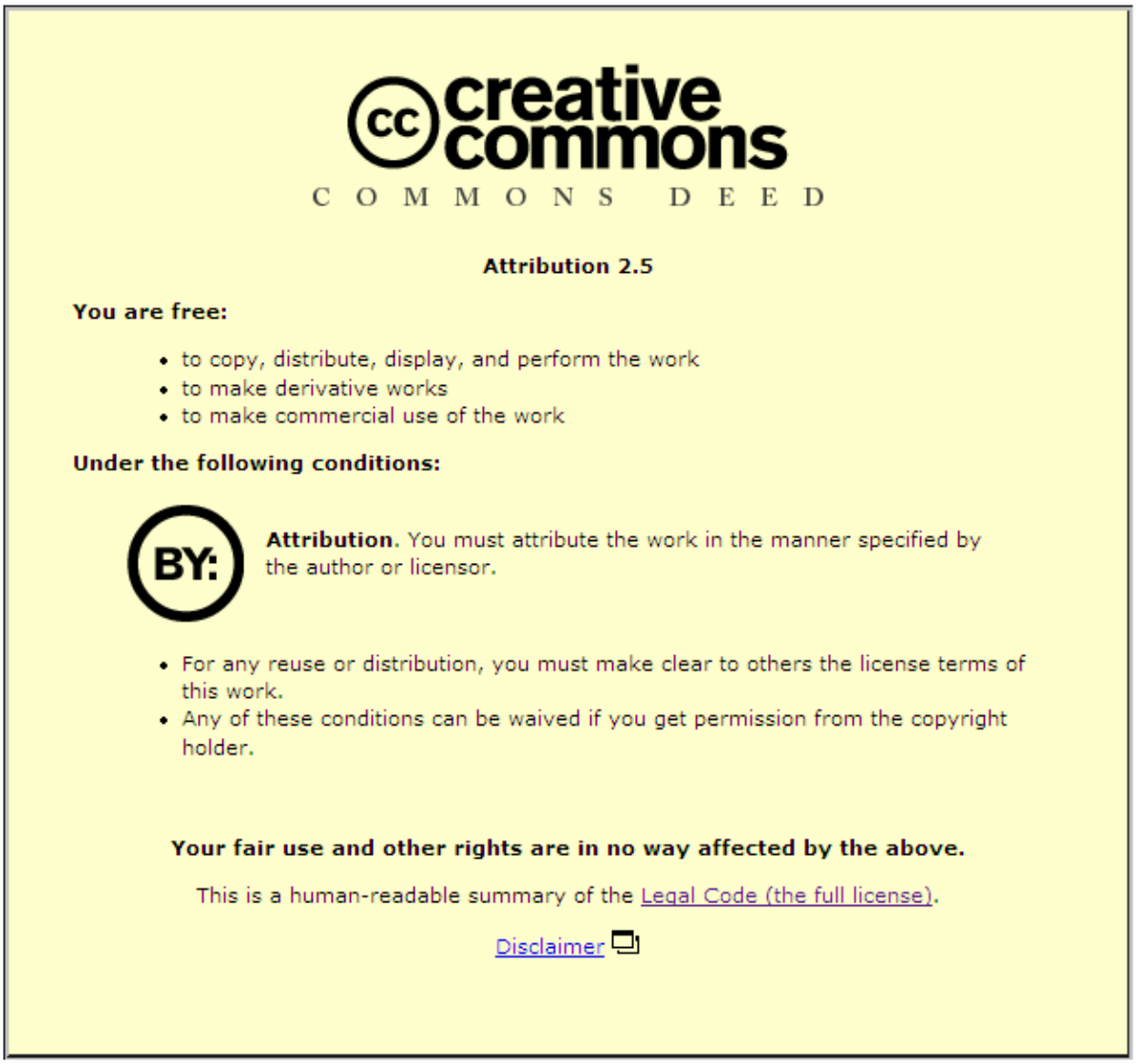
A CXP receiver is expected to manage its own security independent of the sender system or vendor. There is no defined responsibility for a recipient to publish general policies or to handle security for a particular transfer in a particular way. If the sender needs specific security information to manage its own policy, then the negotiation of these assertions is beyond the scope of CXP.

Apart from the use of SSL and TLS there is no authentication of the machine at the other end of the CXP connection. A Sender and Receiver can agree to use extra fields in the Registry Parameters to pass additional authentication data such as a shared secret in order to provide additional validation to the transaction.

[4] For example, use of a recipient's public key as derived from a consent form explicitly signed by the patient is one way to satisfy this legal mandate

Trademarks and Intellectual Property

CXP is licensed under the Creative Commons Attribution license. We request attribution to both MedCommons and the CCR Accelerator Group.



This is a human-readable summary of the Legal Code (the full license).

Disclaimer

Appendix A: Changes history of CXP Protocol

This appendix describes the changes in the different draft versions of the CXP protocol.

Changes since 0.8

- Status no longer a scalar value. There are status codes plus human-readable reasons that can be displayed.
- XML elements now consistently use the same convention for XML element naming as the CCR specification: upper case for abbreviations and upper CamelCase for other element names. So, <CXP> all capitals (like <URL>) and <opcode> is now <OperationCode>.
- Added explicit CXPVersion element in XML blob. This is used by the client to specify which version of CXP is being used. Servers can reject messages in versions of CXP that they do not support.
- Added <InformationSystem> identifiers to parameter block. This describes the vendor's product name and version number. This is mostly useful for error reporting.
- QUERY operation code has been replaced by QUERYTXID and QUERYUID for the two different types of queries.

Changes since 0.9

- Made consistent the specification of transaction numbers and PINS. QUERYTXID has been moved to Appendix C.
- Documented SOAP interface

Changes since 0.9.2

- Changed SOAP WDSL to work with .NET environment. The earlier WDSL worked well for a Java client but the resulting WDSL generated by Axis was not compatible with the .NET tools. The main difference is that the CCR is now encoded as a string instead of as an embedded XML document.
- There is now a working C# client.

Changes since 0.9.3

- Removed REST documentation
- Factored out all registry information (which may vary by vendor or by architecture) to separate objects.
- Made naming more consistent.
- Added section on future plans

Changes since 0.9.9

- Added support for multiple registries
- Made CXP specification a bit more generic; moved some policy constraints to a separate document.

Appendix B – CXP SOAP WDSL

The CXP WDSL is available on every server that implements CXP in a form such as: https://hostname/router/services/CXP?wsdl

The current implementation of CXP uses SOAP 1.1.

The CXP 1.0 WDSL follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://hostname:9080/router/services/CXP" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://hostname:9080/router/services/CXP" xmlns:intf="http://hostname:9080/router/services/CXP"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="urn:BeanService"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.2.1on Jun 14, 2005 (09:15:57 EDT)-->
<wsdl:types>
 <schema targetNamespace="urn:BeanService" xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://hostname:9080/router/services/CXP"/>
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <complexType name="Parameter">
   <sequence>
    <element name="name" nillable="true" type="xsd:string"/>
    <element name="value" nillable="true" type="xsd:string"/>
   </sequence>
  </complexType>
  <complexType name="RegistryParameters">
   <sequence>
    <element name="parameters" nillable="true" type="impl:ArrayOf_tns1_Parameter"/>
    <element name="registryId" nillable="true" type="xsd:string"/>
    <element name="registryName" nillable="true" type="xsd:string"/>
   </sequence>
  </complexType>
  <complexType name="PutResponse">
   <sequence>
    <element name="cxpVersion" nillable="true" type="xsd:string"/>
    <element name="guid" nillable="true" type="xsd:string"/>
    <element name="parameters" nillable="true" type="impl:ArrayOf_tns1_RegistryParameters"/>
    <element name="reason" nillable="true" type="xsd:string"/>
    <element name="registryParameters" nillable="true" type="impl:ArrayOf_tns1_RegistryParameters"/>
    <element name="status" type="xsd:int"/>
   </sequence>
  </complexType>
  <complexType name="CXPAttachment">
   <sequence/>
  </complexType>
  <complexType name="GetResponse">
   <sequence>
    <element name="content" nillable="true" type="xsd:string"/>
    <element name="contentType" nillable="true" type="xsd:string"/>
    <element name="cxpVersion" nillable="true" type="xsd:string"/>
    <element name="guid" nillable="true" type="xsd:string"/>
    <element name="parameters" nillable="true" type="impl:ArrayOf_tns1_RegistryParameters"/>
    <element name="reason" nillable="true" type="xsd:string"/>
    <element name="registryParameters" nillable="true" type="impl:ArrayOf_tns1_RegistryParameters"/>
    <element name="status" type="xsd:int"/>
   </sequence>
  </complexType>
  <complexType name="DeleteResponse">
   <sequence>
    <element name="cxpVersion" nillable="true" type="xsd:string"/>
    <element name="guid" nillable="true" type="xsd:string"/>
    <element name="parameters" nillable="true" type="impl:ArrayOf_tns1_RegistryParameters"/>
    <element name="reason" nillable="true" type="xsd:string"/>
```

```
    <element name="registryParameters" nillable="true" type="impl:ArrayOf_tns1_RegistryParameters"/>
    <element name="status" type="xsd:int"/>
   </sequence>
  </complexType>
 </schema>
 <schema targetNamespace="http://hostname:9080/router/services/CXP" xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="urn:BeanService"/>
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <complexType name="ArrayOf_tns1_Parameter">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Parameter[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_RegistryParameters">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:RegistryParameters[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="CXPException">
   <sequence/>
  </complexType>
  <complexType name="ArrayOf_tns1_CXPAttachment">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:CXPAttachment[]"/>
    </restriction>
   </complexContent>
  </complexType>
 </schema>
</wsdl:types>

  <wsdl:message name="getRequest">

    <wsdl:part name="xmlData" type="xsd:string"/>

  </wsdl:message>

  <wsdl:message name="getVersionResponse">

    <wsdl:part name="getVersionReturn" type="xsd:string"/>

  </wsdl:message>

  <wsdl:message name="putRequest">

    <wsdl:part name="ccrXml" type="xsd:string"/>

  </wsdl:message>

  <wsdl:message name="CXPException">

    <wsdl:part name="fault" type="impl:CXPException"/>

  </wsdl:message>

  <wsdl:message name="deleteRequest">

    <wsdl:part name="guid" type="xsd:string"/>

  </wsdl:message>

  <wsdl:message name="putResponse2">

    <wsdl:part name="putReturn" type="tns1:PutResponse"/>
```

```
</wsdl:message>
<wsdl:message name="getVersionRequest">
</wsdl:message>
<wsdl:message name="deleteRequest1">
  <wsdl:part name="parameters" type="impl:ArrayOf_tns1_RegistryParameters"/>
</wsdl:message>
<wsdl:message name="getRequest1">
  <wsdl:part name="inputRegistryParameters" type="impl:ArrayOf_tns1_RegistryParameters"/>
</wsdl:message>
<wsdl:message name="getResponse">
  <wsdl:part name="getReturn" type="tns1:GetResponse"/>
</wsdl:message>
<wsdl:message name="deleteResponse">
  <wsdl:part name="deleteReturn" type="tns1:DeleteResponse"/>
</wsdl:message>
<wsdl:message name="deleteResponse2">
  <wsdl:part name="deleteReturn" type="tns1:DeleteResponse"/>
</wsdl:message>
<wsdl:message name="putResponse">
  <wsdl:part name="putReturn" type="tns1:PutResponse"/>
</wsdl:message>
<wsdl:message name="putRequest2">
  <wsdl:part name="ccrXml" type="xsd:string"/>
  <wsdl:part name="inputRegistryParameters" type="impl:ArrayOf_tns1_RegistryParameters"/>
  <wsdl:part name="attachments" type="impl:ArrayOf_tns1_CXPAttachment"/>
</wsdl:message>
<wsdl:message name="deleteResponse1">
  <wsdl:part name="deleteReturn" type="tns1:DeleteResponse"/>
</wsdl:message>
<wsdl:message name="putRequest1">
  <wsdl:part name="ccrXml" type="xsd:string"/>
  <wsdl:part name="inputRegistryParameters" type="impl:ArrayOf_tns1_RegistryParameters"/>
</wsdl:message>
<wsdl:message name="deleteRequest2">
  <wsdl:part name="guid" type="xsd:string"/>
```

```
    <wsdl:part name="parameters" type="tns1:RegistryParameters"/>
  </wsdl:message>
  <wsdl:message name="putResponse1">
    <wsdl:part name="putReturn" type="tns1:PutResponse"/>
  </wsdl:message>
  <wsdl:message name="getResponse1">
    <wsdl:part name="getReturn" type="tns1:GetResponse"/>
  </wsdl:message>
  <wsdl:portType name="CXP_10">
    <wsdl:operation name="put" parameterOrder="ccrXml">
      <wsdl:input message="impl:putRequest" name="putRequest"/>
      <wsdl:output message="impl:putResponse" name="putResponse"/>
      <wsdl:fault message="impl:CXPException" name="CXPException"/>
    </wsdl:operation>
    <wsdl:operation name="put" parameterOrder="ccrXml inputRegistryParameters">
      <wsdl:input message="impl:putRequest1" name="putRequest1"/>
      <wsdl:output message="impl:putResponse1" name="putResponse1"/>
      <wsdl:fault message="impl:CXPException" name="CXPException"/>
    </wsdl:operation>
    <wsdl:operation name="put" parameterOrder="ccrXml inputRegistryParameters attachments">
      <wsdl:input message="impl:putRequest2" name="putRequest2"/>
      <wsdl:output message="impl:putResponse2" name="putResponse2"/>
      <wsdl:fault message="impl:CXPException" name="CXPException"/>
    </wsdl:operation>
    <wsdl:operation name="get" parameterOrder="xmlData">
      <wsdl:input message="impl:getRequest" name="getRequest"/>
      <wsdl:output message="impl:getResponse" name="getResponse"/>
      <wsdl:fault message="impl:CXPException" name="CXPException"/>
    </wsdl:operation>
    <wsdl:operation name="get" parameterOrder="inputRegistryParameters">
      <wsdl:input message="impl:getRequest1" name="getRequest1"/>
      <wsdl:output message="impl:getResponse1" name="getResponse1"/>
      <wsdl:fault message="impl:CXPException" name="CXPException"/>
    </wsdl:operation>
```

```
    <wsdl:operation name="delete" parameterOrder="guid">
      <wsdl:input message="impl:deleteRequest" name="deleteRequest"/>
      <wsdl:output message="impl:deleteResponse" name="deleteResponse"/>
    </wsdl:operation>
    <wsdl:operation name="delete" parameterOrder="parameters">
      <wsdl:input message="impl:deleteRequest1" name="deleteRequest1"/>
      <wsdl:output message="impl:deleteResponse1" name="deleteResponse1"/>
    </wsdl:operation>
    <wsdl:operation name="delete" parameterOrder="guid parameters">
      <wsdl:input message="impl:deleteRequest2" name="deleteRequest2"/>
      <wsdl:output message="impl:deleteResponse2" name="deleteResponse2"/>
    </wsdl:operation>
    <wsdl:operation name="getVersion">
      <wsdl:input message="impl:getVersionRequest" name="getVersionRequest"/>
      <wsdl:output message="impl:getVersionResponse" name="getVersionResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CXPSoapBinding" type="impl:CXP_10">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="put">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="putRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>
      </wsdl:input>
      <wsdl:output name="putResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>
      </wsdl:output>
      <wsdl:fault name="CXPException">
        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" name="CXPException"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="put">
      <wsdlsoap:operation soapAction=""/>
```

```
      <wsdl:input name="putRequest1">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>

      </wsdl:input>

      <wsdl:output name="putResponse1">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:output>

      <wsdl:fault name="CXPException">

        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" name="CXPException"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:fault>

    </wsdl:operation>

    <wsdl:operation name="put">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="putRequest2">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>

      </wsdl:input>

      <wsdl:output name="putResponse2">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:output>

      <wsdl:fault name="CXPException">

        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" name="CXPException"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:fault>

    </wsdl:operation>

    <wsdl:operation name="get">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="getRequest">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>

      </wsdl:input>

      <wsdl:output name="getResponse">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:output>

      <wsdl:fault name="CXPException">
```

```
        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" name="CXPException"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:fault>

    </wsdl:operation>

    <wsdl:operation name="get">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="getRequest1">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>

      </wsdl:input>

      <wsdl:output name="getResponse1">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:output>

      <wsdl:fault name="CXPException">

        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" name="CXPException"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:fault>

    </wsdl:operation>

    <wsdl:operation name="delete">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="deleteRequest">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>

      </wsdl:input>

      <wsdl:output name="deleteResponse">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="delete">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="deleteRequest1">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>

      </wsdl:input>

      <wsdl:output name="deleteResponse1">
```

```
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="delete">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="deleteRequest2">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>

      </wsdl:input>

      <wsdl:output name="deleteResponse2">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="getVersion">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="getVersionRequest">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cxp.medcommons.net"
use="encoded"/>

      </wsdl:input>

      <wsdl:output name="getVersionResponse">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:9080/router/services/CXP" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

  </wsdl:binding>

  <wsdl:service name="CXP_10Service">

    <wsdl:port binding="impl:CXPSoapBinding" name="CXP">

      <wsdlsoap:address location="http://hostname:9080/router/services/CXP"/>

    </wsdl:port>

  </wsdl:service>

</wsdl:definitions>
```

Appendix C: Example code

Example C# PUT call

An example of PUT into a MedCommons Registry shows how to fill in the registry parameter block.

```
cxpsoap2.CXP.RegistryParameters[] inputParameters =
   new cxpsoap2.CXP.RegistryParameters[1];
cxpsoap2.CXP.RegistryParameters inputParameter =
   new cxpsoap2.CXP.RegistryParameters();
inputParameters[0] = inputParameter;
inputParameter.registryName = "MedCommons";
inputParameter.registryId = "medcommons.net";


cxpsoap2.CXP.Parameter []parameterList = new cxpsoap2.CXP.Parameter[4];
parameterList[0] = new cxpsoap2.CXP.Parameter();
parameterList[0].name = "CommonsID";
parameterList[0].value = this.CommonsID.Text;

parameterList[1] = new cxpsoap2.CXP.Parameter();
parameterList[1].name = "SenderID";
parameterList[1].value = this.SenderID.Text;

parameterList[2] = new cxpsoap2.CXP.Parameter();
parameterList[2].name = "RegistrySecret";
parameterList[2].value = this.RegistrySecret.Text;

parameterList[3] = new cxpsoap2.CXP.Parameter();
parameterList[3].name = "NotificationSubject";
parameterList[3].value = this.NotificationSubject.Text;

inputParameter.parameters = parameterList;

cxpsoap2.CXP.PutResponse response = null;

// Send the PUT message
try
{
    response = cxpServer.put(ccrData, inputParameters);
}
catch (Exception ex)
{
    MessageBox.Show("Unknown error communicating with server " +this.CXPServerWDSL.Text
        + "\n" + ex.ToString());
    return;
}
if (response.status == 200)
{
    handleOutputParameters(response.registryParameters);
    this.GUID.Text = response.guid;
    MessageBox.Show(this, "CXP Transfer Success!\nUUID:" + response.guid +
         "\nConfirmationCode:" + this.ConfirmationCode.Text +
         "\nRegistrySecret:" + this.RegistrySecret.Text
         );
 }
else{
  // More code here
}
```

Example C# GET Call

```
cxpServer.Url = this.CXPServerWDSL.Text;
cxpsoap2.CXP.RegistryParameters []inputParameters = new
cxpsoap2.CXP.RegistryParameters[1];
cxpsoap2.CXP.RegistryParameters inputParameter = new cxpsoap2.CXP.RegistryParameters();
inputParameters[0] = inputParameter;
inputParameter.registryName = "MedCommons";
inputParameter.registryId = "medcommons.net";


cxpsoap2.CXP.Parameter []parameterList = new cxpsoap2.CXP.Parameter[5];
parameterList[0] = new cxpsoap2.CXP.Parameter();
parameterList[0].name = "CommonsId";
parameterList[0].value = this.CommonsID.Text;

parameterList[1] = new cxpsoap2.CXP.Parameter();
parameterList[1].name = "SenderId";
parameterList[1].value = this.SenderID.Text;

parameterList[2] = new cxpsoap2.CXP.Parameter();
parameterList[2].name = "RegistrySecret";
if (this.RegistrySecret.Text.Length == 0)
    parameterList[2].value = null;
else if (this.RegistrySecret.Text.Length == 5)
{
    parameterList[2].value = this.RegistrySecret.Text;
}
else
{
    MessageBox.Show(this, "Invalid RegistrySecret: Must be a 5 integer string or blank,
not '" + this.RegistrySecret.Text + "'");
    return;
}

parameterList[3] = new cxpsoap2.CXP.Parameter();
parameterList[3].name = "NotificationSubject";
parameterList[3].value = this.NotificationSubject.Text;

parameterList[4] = new cxpsoap2.CXP.Parameter();
parameterList[4].name = "ConfirmationCode";
parameterList[4].value = this.ConfirmationCode.Text;

inputParameter.parameters = parameterList;



cxpsoap2.CXP.GetResponse response = cxpServer.get(inputParameters);
String output = "GET Response:";
if (response.status == 200)
{
    output += "\n First 200 characters of CCR\n";
    output += response.content.Substring(0,200);


}
else
{
    output += "\n Status = " + response.status;
    output += "\n Reason = " + response.reason;
}

MessageBox.Show(this, output);
```

Appendix D: Normative Security/Identity Recommendations

The security goals of CXP are:

1. Permit a private/secure mechanism for messages.
2. Only actors with authorization to obtain a document can obtain a document.

How these goals are obtained is beyond the scope of this specification. This appendix discusses mechanisms for how these may be specified using the existing structure of CXP.

CXP is a simple wire protocol. It knows nothing about user identity, security, object identity, or auditing. However, all of these issues are critical to how CXP will be used.

Transaction/Audit logs

A CXP implementation may require that each party initiating a transaction supply an opaque Sender identity token and that both parties maintain an audit log of each CXP transaction so that the logs of each party can be conjoined for forensic problem and dispute resolution. These tokens may be hardwired shared secrets, or otherwise obtained dynamically via methods outside CXP.

The format of these logs is not yet specified. See the SenderID description in the data dictionary appendix below for what types of identifiers are expected.

Trust relationships

CXP can be used in conjunction with a Federated Identity Service such as Liberty Alliance. In this case, the two parties establish a static trust relationship using Liberty, and dynamically use SAML2.0 and SOAP based WSF2.0 to communicate CXP payloads. The initiator of a CXP transaction obtains an opaque Sender identity token from the Liberty IDP and passes it along to his counterpart (the Liberty Service Provider) as above. But instead of hard coded secrets, or cumbersome certificates, the Liberty services permit dynamic interconnection of CXP participants who have signed bilateral business agreements and achieved technical compliance.

When the CXP Receiver is a Document Repository, and there is a separate Document Registry, the CXP transaction initiator (the document source) may choose to have an initial conversation (WSF2.0) with the Registry (e.g. a Liberty Service Provider) to determine which Repository to contact. In this case there is an implied transfer of trust from the Registry to the Repository and the CXP protocol is then utilized as documented herein. When using a Federated Identity Service this initial conversation is mandatory and supplies the same opaque Sender identity token.

Additionally, TLS may be used to define trust relationships between nodes.

Appendix E: Normative RegistryParameters Data Dictionary

The RegistryParameters data structure contains an array of name-value pairs. While these values are currently unconstrained by the CXP specification it is hoped that common elements can be identified in practice and placed in future versions of the standard.

Name	Value	Description
CommonsID	A string containing an identifier	This is the identifier of the account or user that 'owns' the data. This is a portion of a mechanism that provides the ability for (say) User A to put a document into the system for User B.
SenderID	A string identifying the sender of the document.	The SenderID format and semantics are still under development. A SAML assertion or artifact or a certificate are two candidates for inclusion. A SenderID can fall into three basic categories: 1. It's unique to an individual 2. It's unique to an organization (and the organization in turn may internally be able to connect this token to a particular user) 3. It's a transaction identifier. From the point of view of CXP this means that the user is pseudonymous but potentially traceable via audit logs on other sites.

Appendix F: Future directions

- MTOM for streaming large binary files.
- Security; details on how SAML identifiers can be passed through.
- Support for non-CCR data validation schemas Such as CCD. .
- Handling encrypted document content (awaits to some degree the ASTM committees)