CCR Exchange Protocol (CXP/POST and CXP/SOAP)

Version 0.9.2

Version	Date	Comments
0.8	8/01/2005	Initial version posted to CCR Acceleration group
0.9	9/26/05	Changes based on feedback and experience building client application.
0.9.1	10/03/2005	Moved some MedCommons transaction id behavior to Appendix C.
0.9.2	10/07/2005	Added SOAP back in, added WSDL

CXP is a two party peer to peer protocol that moves CCR-family data and other XML based structures across the Internet between co-operating Sending and Receiving systems from multiple vendors. CCR-related data includes PDF, DICOM, and other documents referenced from within a CCR, and a CCR itself.

A Transfer is the movement of a CCR and Referenced documents or other XML based data. In CXP, we refer to whichever party is holding and moving the CCR as the Sender, and the party that is accepting and potentially storing the CCR as the Receiver.

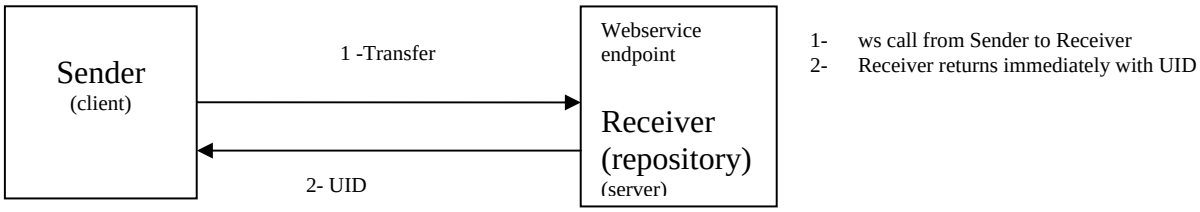
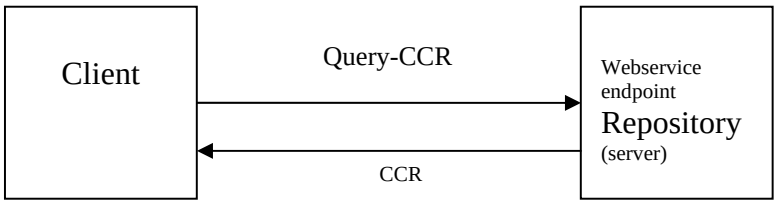


Figure 1

A Transfer is initiated by a Transfer Command calling a pre-registered webservice endpoint URL. As shown in figure 1, a basic Transfer flows from Sender to Receiver, is safely stored by the Receiver, and a document key known as the UID flows back from the Receiver to the Sender.

The UID is defined by the CXP protocol and must be precisely implemented by both Sender and Receiver[1].



[1] In MedCommons the UID of a document is the SHA-1 hash of its contents. See appendix for open source javascript

As shown in figure 2, a specific CCR can be located via the Query-CCR Command by passing in the UID. To retrieve attachments, a similar Query-XML Command is utilized.

Indexing and Retrievals

CCRs will be stored and indexed in different ways by different Vendors, but each CCR is accessible by UID, which is the SHA-1 hash of the CCR itself. There are no other prescribed or mandatory keys, tags, or any identifying characteristics other than data present in the CCR itself and some vendors may accept CCRs alone, with no additional metadata. Vendors can index whatever elements of the CCR they choose and may accept an optional XML Data Block[2] to support the client authorization and query requirements when those elements are not accessible in the CCR.

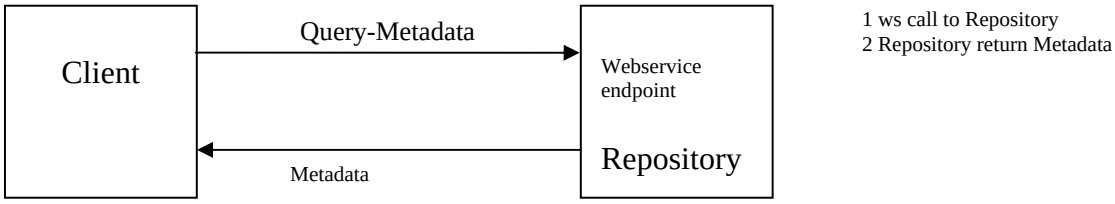


Figure 2

A Query-MetaData Command can be issued to retrieve metadata about CCRs. Different Vendors will support different Query options depending upon, security, and flexibility preferences.

Apart from the UID, which can be computed by either the client or the repository, the identifiers used in the queries are determined by the Receiver. The scope of the identifier is determined by the repository – some identifiers may be permanent; others are time limited; still others may be one-time access keys. Defining these further is outside the scope of this proposal.

[2] We may want to enter a constraint here that the XML block should not replicate data that is in the CCR. This is attractive because it means that there isn't the possibility of inconsistencies between the CCR and the XML block. It's unattractive because for some vendors the XML block may be condensed version of the CCR that parses more quickly or it may be some other standard XML format which requires (say) patient demographics. This should be nailed down by discussions with the accelerator group.

Interaction with Higher Level Protocols

There will be higher level protocols constructed on top of CXP. For example, there may be protocols for registration of Patients and other administrative tasks as well as for moving CCRs and attachments. In all cases these are layered on top of CXP as follows:

- CCRs and an optional XML-Data block are passed back and forth in standard form
- Administrative commands encode their functions and return status in an XML-Data block
- CCR Referenced documents may be transferred along with the CCR as XML-Data or separately by non-CXP methods.
- For the http POST protocol variant, CCR References moving with the CCR are base-64 encoded and passed as part of XML-Data. For the http SOAP protocol variant CCR references are encoded according to the WSDL specification for the CXP interface as shown in appendix F.
- CCR References not moving with the CCR may need additional standards and protocols.
- SSL or TLS may be utilized and is encouraged between parties, but it is not required.
- Encrypted CCRs are permitted in CXP.

Thus the XML-Data block is where all non-CCR data that needs to be passed between systems is expressed. This is application specific. Standardizing these elements may be warranted as more vendors participate.

Security

A CXP recipient vendor shall accept CCRs encrypted per the ASTM CCR spec. Status responses indicate if insufficient information was provided in order to issue a confirmation to the sender. In some cases, transfer of encrypted CCRs may require pre-registration of the patient or other Actors. These pre-registration steps are considered higher level protocols beyond CXP.

The CXP sender must accept responsibility for release of information to the recipient. Patient consent to information release is the responsibility of the sender.[3]. Consent and the key management are outside of CXP. Another example might be implied consent between two CXP systems that have a HIPAA provider 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.

[3] 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

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 XML Data Block to pass additional authentication data such as a shared secret in order to provide additional validation to the transaction.

Implementation

CXP is based on SOAP over HTTP(S) or alternatively POST over HTTP(S). A Sender or Receiver may choose to implement either or both alternatives. This is a static choice made by the Vendor during installation. If implementing both, a separate set of URL endpoints will be utilized.

For a Sender to reliably connect and transfer a CCR to a Receiver, the Receiver must have a fixed public IP address and must publish the URL(s) (via email, word of mouth, etc) prior to accepting CXP connections.

For those operations such as Queries that do not imply a Sender and Receiver, but rather a Client and a Repository, the Repository is required to have a fixed IP address.

Commands

All commands map directly to SOAP or POST over HTTP. For compatibility, only a single output argument is utilized:

The general form is:

XML-Return-Data = CXP-Command([CCR],[XML-Data]);

- ALL invocations of CXP-command return an XML data structure. The formats of this data structure are discussed below, and in the Errors section.

- Either the XML-Data block, OR the CCR must be present. Otherwise an encoded 404 error is returned to the caller.

- If the XML-Data block is absent, and operation-code of ‘Transfer’ is implicitly assumed. Otherwise the operation-code in the XML-Data block is utilized and other data values are defaulted as described elsewhere in this spec.

- Some operation codes return a real CCR as the XML-Return-Data, others will generate other XML return structures. This will evolve other time.

The same form of command is used in both directions. When a Notification of an incoming CCR is delivered via SOAP or POST, the argument list is exactly as described above.

XML-DATA

The XML-Data block passed between Sender and Receiver has different fields of interest depending on the specific operation request. The general structure, which is always unencrypted, is

```
<CXP>
  <OperationCode>opcode</OperationCode>
  <InformationSystem>
    <Name>Vendor</Name>
    <Type>Type of application</Type>
    <Version>Vendor's version number</Version>
  </InformationSystem>
  <CXPVersion>Version of CXP</CXPVersion>
  <SenderID>1234</SenderID>
.. other operation-code specific elements
</CXP>
```

The only mandatory field is OperationCode. The other fields are present or not, depending on the needs of the OperationCode. The Version and InformationSystem fields are currently not required. The xml data block is naturally extensible by adding additional tags which must be ignored if not understood by the Receiver.

Operation code TRANSFER specific elements

These elements are required for transfer of attached files. There is one <File> per attached file. The CCR must refer to these attachments via a <Reference>.
See Appendix C for the details of how CCR <Reference> fields are defined. See Appendix D for an example of transfer.

```
<Files>
  <File>
    <FileName>filename</FileName>
    <FileType>content type</FileType>
    <SHA1>hash of filename's contents</SHA1>
     <FileContents>
     </FileContents>
  </File>
  … additional files
</Files>
```

Operation Code QUERYUID specific elements

To query for a specific document (as identified by the UID) create a QUERYUID request using the UID specified the the <UID> element:

<UID>*SHA-1 hash identifier of document*</UID>

See Appendix D for example

Table: Required Operation Codes[4]

Operation Code	CCR	XML-Data Block	XML-Returned-Data-Block
TRANSFER	Yes	Optional. If present includes attachments	Returns status and reason; if status is successful (values 200->299) then a UID is returned.
QUERYUID	No	Required: Includes UID query parameter	Returns success or failure for query along with either CCR or XML data

Additional OperationCodes are defined in Appendix C.

Table: Defined Content Types

Content type	Document type
application/pdf	PDF document
application/dicom	DICOM Series

Returned parameters blocks and error handling

Any invocation of CXP-command can return all O/S level TCP and HTTP errors. Beyond that, everything is returned as valid XML. All errors come back as valid XML structures.

The standard returned XML is of the following form:

<CXP>
<OperationCode>*Operation code* </OperationCode>
<Status> *Status code (integer)* </Status>
<Reason> *Human readable status* </Reason>
<UID>
Returned by TRANSFERS – this is the SHA-1 hash of the transferred document </UID>

[4] Another operation code QUERYTXID is defined in Appendix C. We hope that discussions on the CCR Accelerator list help define what is in the base protocol and which extensions are necessary for certain classes of applications.

```
  <CXPVersion>Version of CXP</CXPVersion>
  <InformationSystem>
    <Name>Vendor Name</Name>
    <Type>Application type</Type>
    <Version>Software version</Version>
</InformationSystem>
</CXP>
```

The operation code is the same one that is specified in the original request.

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.

Code range	Meaning
200-299	Success
400-499	Bad request (client error)
500-599	Server error

For transfers one parameter must be returned: the UID (global identifier) of the document. Other parameters may be returned depending on the application but only the UID is required.

A successful query will return the document. An unsuccessful one will return a parameter block with a 4xx or 5xx value status code.

See appendix D for examples of returned parameter blocks.

Trademarks and Intellectual Property

CXP (or whatever the CCR Accelerator Group chooses to call this) is an alternative to XML File Import or Export from a Vendor System when the file is a CCR. CXP provides the alternative of a Web Service endpoint for the Import or Export target and, beyond that, offers the minimum required set of tracking and status protocols to enable reliable transfer of responsibility between two cooperating vendors. It will be useful for either ASTM or AAFP or the CCR Accelerator Group to take ownership and control of the trademark for Web transport via CXP in order to improve the rate of adoption in the marketplace.

Appendix A: SHA-1 Implementation in Javascript

We have included the SHA-1 code that can be used by a CXP Javascript Clients. It's just an example – the calculated hash value here is identical with the one calculated by Sun's java.security.MessageDigest's SHA-1 implementation.

```
/*
 * A JavaScript implementation of the Secure Hash Algorithm, SHA-1, as defined
 * in FIPS PUB 180-1
 * Version 2.1 Copyright Paul Johnston 2000 - 2002.
 * Other contributors: Greg Holt, Andrew Kepert, Ydnar, Lostinet
 * Distributed under the BSD License
 * See http://pajhome.org.uk/crypt/md5 for details.
 */

/*
 * Configurable variables. You may need to tweak these to be compatible with
 * the server-side, but the defaults work in most cases.
 */
var hexcase = 0;  /* hex output format. 0 - lowercase; 1 - uppercase        */
var b64pad  = ""; /* base-64 pad character. "=" for strict RFC compliance   */
var chrsz   = 8;  /* bits per input character. 8 - ASCII; 16 - Unicode      */

/*
 * These are the functions you'll usually want to call
 * They take string arguments and return either hex or base-64 encoded strings
 */
function hex_sha1(s){return binb2hex(core_sha1(str2binb(s),s.length * chrsz));}
function b64_sha1(s){return binb2b64(core_sha1(str2binb(s),s.length * chrsz));}
function str_sha1(s){return binb2str(core_sha1(str2binb(s),s.length * chrsz));}
function hex_hmac_sha1(key, data){ return binb2hex(core_hmac_sha1(key, data));}
function b64_hmac_sha1(key, data){ return binb2b64(core_hmac_sha1(key, data));}
function str_hmac_sha1(key, data){ return binb2str(core_hmac_sha1(key, data));}

/*
 * Perform a simple self-test to see if the VM is working
 */
function sha1_vm_test()
{
  return hex_sha1("abc") == "a9993e364706816aba3e25717850c26c9cd0d89d";
}

/*
 * Calculate the SHA-1 of an array of big-endian words, and a bit length
 */
function core_sha1(x, len)
{
  /* append padding */
  x[len >> 5] |= 0x80 << (24 - len % 32);
  x[((len + 64 >> 9) << 4) + 15] = len;

  var w = Array(80);
  var a =  1732584193;
  var b = -271733879;
  var c = -1732584194;
  var d =  271733878;
  var e = -1009589776;

  for(var i = 0; i < x.length; i += 16)
  {
    var olda = a;
    var oldb = b;
    var oldc = c;
    var oldd = d;
    var olde = e;
```

```
    for(var j = 0; j < 80; j++)
    {
      if(j < 16) w[j] = x[i + j];
      else w[j] = rol(w[j-3] ^ w[j-8] ^ w[j-14] ^ w[j-16], 1);
      var t = safe_add(safe_add(rol(a, 5), sha1_ft(j, b, c, d)),
                       safe_add(safe_add(e, w[j]), sha1_kt(j)));
      e = d;
      d = c;
      c = rol(b, 30);
      b = a;
      a = t;
    }

    a = safe_add(a, olda);
    b = safe_add(b, oldb);
    c = safe_add(c, oldc);
    d = safe_add(d, oldd);
    e = safe_add(e, olde);
  }
  return Array(a, b, c, d, e);

}

/*
 * Perform the appropriate triplet combination function for the current
 * iteration
 */
function sha1_ft(t, b, c, d)
{
  if(t < 20) return (b & c) | ((~b) & d);
  if(t < 40) return b ^ c ^ d;
  if(t < 60) return (b & c) | (b & d) | (c & d);
  return b ^ c ^ d;
}

/*
 * Determine the appropriate additive constant for the current iteration
 */
function sha1_kt(t)
{
  return (t < 20) ?  1518500249 : (t < 40) ?  1859775393 :
         (t < 60) ? -1894007588 : -899497514;
}

/*
 * Calculate the HMAC-SHA1 of a key and some data
 */
function core_hmac_sha1(key, data)
{
  var bkey = str2binb(key);
  if(bkey.length > 16) bkey = core_sha1(bkey, key.length * chrsz);

  var ipad = Array(16), opad = Array(16);
  for(var i = 0; i < 16; i++)
  {
    ipad[i] = bkey[i] ^ 0x36363636;
    opad[i] = bkey[i] ^ 0x5C5C5C5C;
  }

  var hash = core_sha1(ipad.concat(str2binb(data)), 512 + data.length * chrsz);
  return core_sha1(opad.concat(hash), 512 + 160);
}

/*
 * Add integers, wrapping at 2^32. This uses 16-bit operations internally
 * to work around bugs in some JS interpreters.
 */
function safe_add(x, y)
{
  var lsw = (x & 0xFFFF) + (y & 0xFFFF);
  var msw = (x >> 16) + (y >> 16) + (lsw >> 16);
```

```
  return (msw << 16) | (lsw & 0xFFFF);
}

/*
 * Bitwise rotate a 32-bit number to the left.
 */
function rol(num, cnt)
{
  return (num << cnt) | (num >>> (32 - cnt));
}

/*
 * Convert an 8-bit or 16-bit string to an array of big-endian words
 * In 8-bit function, characters >255 have their hi-byte silently ignored.
 */
function str2binb(str)
{
  var bin = Array();
  var mask = (1 << chrsz) - 1;
  for(var i = 0; i < str.length * chrsz; i += chrsz)
    bin[i>>5] |= (str.charCodeAt(i / chrsz) & mask) << (24 - i%32);
  return bin;
}

/*
 * Convert an array of big-endian words to a string
 */
function binb2str(bin)
{
  var str = "";
  var mask = (1 << chrsz) - 1;
  for(var i = 0; i < bin.length * 32; i += chrsz)
    str += String.fromCharCode((bin[i>>5] >>> (24 - i%32)) & mask);
  return str;
}

/*
 * Convert an array of big-endian words to a hex string.
 */
function binb2hex(binarray)
{
  var hex_tab = hexcase ? "0123456789ABCDEF" : "0123456789abcdef";
  var str = "";
  for(var i = 0; i < binarray.length * 4; i++)
  {
    str += hex_tab.charAt((binarray[i>>2] >> ((3 - i%4)*8+4)) & 0xF) +
           hex_tab.charAt((binarray[i>>2] >> ((3 - i%4)*8  )) & 0xF);
  }
  return str;
}

/*
 * Convert an array of big-endian words to a base-64 string
 */
function binb2b64(binarray)
{
  var tab = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
  var str = "";
  for(var i = 0; i < binarray.length * 4; i += 3)
  {
    var triplet = (((binarray[i   >> 2] >> 8 * (3 -  i   %4)) & 0xFF) << 16)
                | (((binarray[i+1 >> 2] >> 8 * (3 - (i+1)%4)) & 0xFF) << 8 )
                |  ((binarray[i+2 >> 2] >> 8 * (3 - (i+2)%4)) & 0xFF);
    for(var j = 0; j < 4; j++)
    {
      if(i * 8 + j * 6 > binarray.length * 32) str += b64pad;
      else str += tab.charAt((triplet >> 6*(3-j)) & 0x3F);
    }
  }
  return str;
}
```

Appendix B: Changes in 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

Appendix C: MedCommons Repository Extensions

The following extensions to CXP are specific to MedCommons: they are not part of the CXP specification because different repositories may have different structures than MedCommons. Although these extensions are specific to MedCommons they could become part of CXP in future versions if the CCR Accelerator Group/AAFP/ASTM pushed CXP in that direction.

References for File Attachments

The current CCR description of <References> is very general. CXP will accept any form of <References> but it does impose constraints when <Reference> contents refer to an attachment within a CXP transfer. The following requirements guided the design:

- References to the file must follow the CCR schema and implementation guide.
- References must be unambiguous. A CCR may have a reference to URLs that are outside the MedCommons repository; there should not be any ambiguity about which URLs apply to the attachments.
- Must not interfere with any other <Reference> element schemes. If a third party application has inserted <Reference> data into the CCR these elements are simply passed through the system. CXP (and MedCommons) are neutral transfer agents.
- The scheme for mapping <References> to attachments must be compatible with signed CCR documents.

The requirements that the CCR might be signed (e.g., it is an immutable document) and that there may be embedded references to documents sent as attachments server mean that the reference identifiers must be determined before the CCR is signed.

Within MedCommons a document identifier is the SHA-1 hash of the contents of that document. This permits the CXP client to generate document references before signing the CCR. The URL of the document is of the following form:

mcid://<SHA-1>

for example:

mcid://f021a6f7b0607ac52c5d972572136ab3335894e1

We imagine that other vendors will have other URL types and that this will be an area for experimentation.

The type of the Reference must also be specified. The currently supported types in MedCommons are:

- application/pdf
- application/dicom

We imagine that other types will be added in the future as well.

There are two ObjectAttributes for each location: the URL of the object and its DisplayName. The DisplayName is the human-readable name of the object.

Example of PDF document reference

```
<Reference>
 <ReferenceObjectID>mcref-0</ReferenceObjectID>
   <Type>
    <Text>application/pdf</Text>
  </Type>
   <Source>
     <ActorID>AA87987298</ActorID>
  </Source>
  <Locations>
   <Location>
    <Description>
     <ObjectAttribute>
      <Attribute>URL</Attribute>
     <AttributeValue>
      <Value>mcid://a0e695f4c056ac21c04e90d1ca3a33a4434f0141</Value>
     </AttributeValue>
    </ObjectAttribute>
    <ObjectAttribute>
     <Attribute>DisplayName</Attribute>
     <AttributeValue>
       <Value>Health CareProxy.pdf</Value>
     </AttributeValue>
    </ObjectAttribute>
   </Description>
  </Location>
 </Locations>
</Reference>
```

Support for Transaction Numbers and PINS

The MedCommons repository generates a transaction number and PIN that can be used to access the data by a third party application. This means

- An OperationCode of TRANSFER generates the <TXID> and <PIN> values
- An OperationCode of QUERYTXID can be used to retrieve a CCR based on a <TXID> and <PIN>.

The transaction ID can be sent to a user separately from the PIN.

Return values from TRANSFER

The MedCommons repository generates three values that are returned by a successful TRANSFER:

- TXID – the transaction id or 'tracking number' for the transfer. This is a number that can be used (with the PIN) to retrieve the document. The MedCommons implementation uses a 12 digit TXID.
- PIN – the 'password' associated with the TXID. This is typically communicated to a user separately from the TXID.
- [This is required for all CXP transfers] UID – the global identifier of the document.

Thus – the XML returned from a transfer is of the form:
The standard returned XML is of the following form:

```
<CXP>
  <OperationCode>Operation code </OperationCode>
  <Status> Status code (integer) </Status>
  <Reason> Human readable status </Reason>
  <UID>
  Returned by TRANSFERS – this is the SHA-1 hash of the transferred document </UID>
  <TXID> transaction id</TXID>
  <PIN> pin </PIN>
  <CXPVersion>Version of CXP</CXPVersion>
  <InformationSystem>
    <Name>Vendor Name</Name>
    <Type>Application type</Type>
    <Version>Software version</Version>
</InformationSystem>
</CXP>
```

Query by Tracking Number Extension

Operation Code QUERYTXID specific elements

To obtain a document by tracking number both the tracking number and PIN must be specified:

```
  <TXID>Transaction Number</TXID>
  <PIN>PIN</PIN>
```

See Appendix D for example.

Additions to Operation Codes table

QUERYTXID	No[5]	Required: Includes TXID and PIN query parameters	Returns success or failure for query along with either CCR or XML data

[5] If a Query returns a CCR plus attachments, then the simple form of Query can not be used to get the attachments. Instead, a notification endpoint is specified in the XML Data Block supplied with the Query. The endpoint will be called with the CCR and attachments encoded as a CCR and an XML Data Block that can be decomposed into separate attachments

Appendix D: Example messages

Example Successful Transfer Response

```
<CXP>
  <OperationCode>TRANSFER</OperationCode>
  <Status>200</Status>
  <UID>6808c479626110c0920e6e3594addc70c9cc0a00</UID>
  <Reason>OK</Reason>
  <TXID>917814521864</TXID>
  <PIN>19340</PIN>
  <InformationSystem>
    <Name>MedCommons</Name>
    <Type>Repository</Type>
    <Version>1.0.0.5</Version>
    </InformationSystem>
    <CXPVersion>0.9</CXPVersion>
</CXP>
```

Example Query by UID

```
<CXP>
  <OperationCode>QUERYUID</OperationCode>
  <UID>6808c479626110c0920e6e3594addc70c9cc0a00</UID>
  <CXPVersion>0.9</CXPVersion>
  <InformationSystem>
    <Name>MedCommons</Name>
    <Type>CXP Client</Type>
    <Version>0.9.17</Version>
  </InformationSystem>
</CXP>
```

Example Query by Transaction ID

Note: this uses the MedCommons extensions from Appendix C.

```
<CXP>
  <OperationCode>QUERYTXID</OperationCode>
  <TXID>917814521864</TXID>
  <PIN>19340</PIN>
  <CXPVersion>0.9</CXPVersion>
  <InformationSystem>
      <Name>MedCommons</Name>
      <Type>CXP Client</Type>
      <Version>0.9.17</Version>
    </InformationSystem>
</CXP>
```

Successful Query Response

The requested document (e.g., the bytestream) is returned.

Failed query response

```
<CXP>
  <OperationCode>QUERYTXID</OperationCode>
  <Status>500</Status>
  <Reason>
    Tracking number + 0917814521865 does not refer to a known  document
    </Reason>
  <InformationSystem>
    <Name>MedCommons</Name>
    <Type>Repository</Type>
    <Version>1.0.0.5</Version>
    </InformationSystem>
    <CXPVersion>0.9</CXPVersion>
</CXP>
```

Appendix E – CXP SOAP Specifics

All CXP operations are available via SOAP. The WSDL for the CXP SOAP interface is detailed below.

Once installed, CXP SOAP connectivity can be verified by direct URL such as

http://hostname/router/CXPServer.jws?method=getVersion

This will deliver a response similar to:

<soapenv:Envelope>
<soapenv:Body>
  <getVersionResponseapenv:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/">
  <getVersionReturn xsi:type="xsd:string">
    CXP Server Version 1.0.0.5 built on 2005-04-04 03:33:57
  </getVersionReturn>
  </getVersionResponse>
</soapenv:Body>
</soapenv:Envelope>

CXP WSDL

This is a WSDL file for the CXP interface as generated by Apache Axis. We expect this to change significantly via the efforts of the CCR Accelerator Group and other interested parties.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://hostname/router/CXPServer.jws"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://hostname/router/CXPServer.jws"
   xmlns:intf="http://hostname/router/CXPServer.jws"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://cxp.medcommons.net"
   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: 1.2.1
Built on Jun 14, 2005 (09:15:57 EDT)-->
   <wsdl:types>
      <schema targetNamespace="http://cxp.medcommons.net"
xmlns="http://www.w3.org/2001/XMLSchema">
         <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
         <complexType name="CXPException">
            <sequence/>
         </complexType>
```

```
      </schema>
   </wsdl:types>
   <wsdl:message name="CXPException">
      <wsdl:part name="fault" type="tns1:CXPException"/>
   </wsdl:message>
   <wsdl:message name="commandRequest">
      <wsdl:part name="ccrXml" type="xsd:string"/>
      <wsdl:part name="xmlData" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="commandResponse">
      <wsdl:part name="commandReturn" type="xsd:string"/>
   </wsdl:message>
   <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="getResponse">
      <wsdl:part name="getReturn" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="putResponse">
      <wsdl:part name="putReturn" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="getVersionRequest"> </wsdl:message>
   <wsdl:portType name="CXPServer">
      <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="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="command" parameterOrder="ccrXml xmlData">
         <wsdl:input message="impl:commandRequest" name="commandRequest"/>
         <wsdl:output message="impl:commandResponse" name="commandResponse"/>
         <wsdl:fault message="impl:CXPException" name="CXPException"/>
      </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="CXPServerSoapBinding" type="impl:CXPServer">
      <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://DefaultNamespace" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="putResponse">
```

```
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://hostname/router/CXPServer.jws" use="encoded"/>
         </wsdl:output>
         <wsdl:fault name="CXPException">
            <wsdlsoap:fault
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               name="CXPException"
               namespace="http://hostname/router/CXPServer.jws" 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://DefaultNamespace" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="getResponse">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://hostname/router/CXPServer.jws" use="encoded"/>
         </wsdl:output>
         <wsdl:fault name="CXPException">
            <wsdlsoap:fault
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               name="CXPException"
               namespace="http://hostname/router/CXPServer.jws" use="encoded"/>
         </wsdl:fault>
      </wsdl:operation>
      <wsdl:operation name="command">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="commandRequest">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="commandResponse">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://hostname/router/CXPServer.jws" use="encoded"/>
         </wsdl:output>
         <wsdl:fault name="CXPException">
            <wsdlsoap:fault
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               name="CXPException"
               namespace="http://hostname/router/CXPServer.jws" use="encoded"/>
         </wsdl:fault>
      </wsdl:operation>
      <wsdl:operation name="getVersion">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getVersionRequest">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="getVersionResponse">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://hostname/router/CXPServer.jws" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
```

```
   <wsdl:service name="CXPServerService">
      <wsdl:port binding="impl:CXPServerSoapBinding" name="CXPServer">
         <wsdlsoap:address location="http://hostname/router/CXPServer.jws"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```