HL7 hData RESTful Transport v0.19

Editor: Gerald Beuchelt

Contributors: Robert Dingwell, Andrew Gregorowicz, Marc Hadley, Mark Kramer, Samuel Sayer, Harry Sleeper

The MITRE Corporation
202 Burlington Rd.
Bedford, MA 01730
U.S.A.

1 Introduction

The hData RESTful API specification defines a network transport API for accessing components of a Health Record and sending messages to an EHR system.

A related specification, the hData Record Format (HRF) **Invalid source specified.**, describes the logical organization of the information in an electronic health record (EHR). Please refer to the HRF specification for more details on the HL7 hData Record Format and how it fits into the HL7 version 3 standards.

1.1 Namespaces

This document uses the following namespaces. This specification uses a number of namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant.

Namespace Prefix	Namespace URI	Description
hrf	http://projecthdata.org/hdata/schemas/2009/06/core	Namespace for elements in this document
hrf-md	http://projecthdata.org/hdata/schemas/2009/11/meta	SectionDocument metadata

1.2 Glossary (Non-Normative)

HL7 hData Record Format (HRF) - this specification specifies an abstract hierarchical organization, packaging, and metadata for individual documents (referred to as “Section Documents” within the HRF specification). Section Documents can be of any type, either XML documents (such as CDA documents,

H7v3 messages, or simplified XML wire formats, etc.) or of other media types (such as e.g. MS Word documents or DICOM files). Also contained in this specification is the format for specifying the content that goes into an hData record, which is called the hData Content Profile (HCP) format.

hData Record (HDR) - an single instantiation of the HRF.

HL7 hData Restful Transport - this specification defines how the abstract hierarchical organization defined within the HRF specification is access and modified through a RESTful approach, using HTTP as the access protocol. It creates a unique mapping to an URL structure, and defines how HTTP verbs such as GET, PUT, DELETE, etc. affect the underlying information.

hData Content Profile (HCP) - a profile of the content of an HDR. The HRF specification contains the definition of the HCP format. RLUS – fill in!!

1.3 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

When describing concrete XML schemas, this specification uses the following notation: each member of an element's [children] or [attributes] property is described using an XPath notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element wildcard. The use of @{any} indicates the presence of an attribute wildcard.

2 hData Record RESTful Transport

2.1 Overview

Any HDR can be represented as a set of Hypertext Transfer Protocol (HTTP 1.1, see [8]) resources in a canonical way. The entire HDR is referenced by a base URL that depends on the implementation. See IETF RFC 3986, section 5 for more details. This base URL will be denoted as *baseURL* throughout this document.

2.1.1 Out of Scope

While this specification does not dictate the format of the base URL, the base URL SHOULD NOT contain a query component. All content within an HDR MUST be expressible as a HTTP resource. In the following, the minimum version for HTTP is 1.1. This specification does not define any access controls to the web resources. It is RECOMMENDED that a comprehensive access control management system is always deployed with any hData installation (see Section 4)

2.1.2 General Conventions

Any HTTP GET, PUT, POST, DELETE, or OPTIONS operation (see [8], section 9) on a given resource that are either (i) unspecified or (ii) not implemented MUST return an HTTP response with a status code of

405 that includes an Allow header that specifies the allowed methods. All operations may return HTTP status codes in the 5xx range if there is a server problem.

It is RECOMMENDED that all section document responses include a "Last-Modified" header. It is RECOMMENDED that all document resources support the “If-ModifiedSince” and “If-Unmodified-Since” headers to support conditional GET and optimistic concurrency.

2.2 Operations on the Base URL

2.2.1 GET

If there is no HRF at the base URL, the server SHOULD return a 404 - Not found status code.

The server MUST offer an Atom 1.0 compliant feed of all child sections, as identified in the corresponding sections node in the root document. Each entry MUST contain a link to the resource for each child section.

It is RECOMMENDED that the server also offers a web user interface that allows users to access and manipulate the content of the HDR, as permitted by the policies of the system. Selecting between the two can be achieved using standard content negotiation (HTTP Accept header). This is not necessary for systems that are used by non-person entities only.

Status Code: 200, 404

2.2.2 POST – Parameters: extensionId, path, name

The request body is of type “application/x-www-form-urlencoded” and MUST contain the extensionId, path, and name parameters. The extensionId parameter MUST be a string that is equal to the extensionId attribute of one of the registered <extension> nodes of the root document of the HDR identified by *baseURL*. The path MUST be a string that can be used as a URL path segment. If any parameters are incorrect or not existent, the server MUST return a status code of 400.

The system MUST confirm that there is no other section registered as a child node that uses the same path name. If there is a collision, the server MUST return a status code of 409.

If the extensionId is not registered as a valid extension, the server MUST verify that it can support this extension. If it cannot support the extension it MUST return a status code of 406. It MAY provide additional entity information. If it can support that extension, it MUST register it with the root.xml of this record.

When creating the section resource, the server MUST update the root document: in the node of the parent section a new child node must be inserted. If successful, the server MUST return a 201 status code and SHOULD include the location of the new section. The name parameter MUST be used as the user-friendly name for the new section.

Status Code: 201, 400, 406, 409

2.2.3 PUT

This operation is undefined by this specification.

Status Code: 405, unless an implementer defines this operation.

2.2.4 DELETE

This operation is undefined by this specification.

Status Code: 405, unless an implementer defines this operation.

2.2.5 OPTIONS

The OPTIONS operation on the *baseURL* is per [8], section 9.2, intended to return communications options to the clients. Within the context of this specification, OPTIONS is used to indicate which security mechanisms are available for a given *baseURL* and a list of hData content profiles supported by this implementation. All implementations MUST support OPTIONS on the *baseURL* of each HDR and return a status code of 200, along with:

- The X-hdata-security HTTP header defined in section 4 of this specification The security mechanisms defined at the baseURL are applicable to all child resources, i.e. to the entire HDR.
- An X-hdata-hcp HTTP header that contains a comma separated list of the identifiers of the hData Content Profiles supported by this implementation

The server MAY include additional HTTP headers. The response SHOULD not include an HTTP body. The client MUST NOT use the Max-Forward header when requesting the security mechanisms for a given HDR.

Status Code: 200

2.3 *baseURL*/root.xml

2.3.1 GET

This operation returns an XML representation of the current root document, as defined by the HRF specification.

Status Code: 200

2.3.2 POST, PUT, DELETE

These operations MUST NOT be implemented.

Status Code: 405

2.4 *baseURL*/*sectionpath*

2.4.1 GET

This operation MUST return an Atom 1.0 **Invalid source specified.** compliant feed of all section documents and child sections contained in this section. Each entry MUST contain a link to a resource

that uniquely identifies the section document or child section. If the section document type defines a creation time, is RECOMMENDED to set the Created node to that datetime.

For section documents, the Atom Content element MUST contain the XML representation of its metadata (see **Invalid source specified.**, Section 2.4.1).

Status Code: 200

2.4.2 POST

For creating a new sub section, three additional parameters are used, and the POST will create a new child section within this section. For new documents a document MUST be sent that conforms to the business rules expressed by the extension that the section has registered.

2.4.2.1 Add new section – Parameters: extensionId, path, name

The content type MUST equal "application/x-www-form-urlencoded" for the POST method to create a new sub section. The extensionId parameter is the URI in the root.xml document that identifies the Extension element. If the extensionId is not registered as a valid extension, the server MUST verify that it can support this extension. If it cannot support the extension it MUST return a status code of 406 and MAY provide additional information in the entity body. If it can support that extension, it MUST register it with the root.xml of this record. The path MUST be a string that can be used as a URL path segment. The name parameter MUST be used as the user-friendly name for the new section. If any parameters are incorrect, the server MUST return a status code of 400.

The system MUST confirm that there is no other section registered as a child node that uses the same path name and that it can create a new subsection identified by the path parameter. If there is a collision, the server MUST return a status code of 409.

When creating the section resource, the server MUST update the root document: in the node of the parent section a new child node must be inserted. The server MUST return a 201 status code. The extensionId and path parameters are REQUIRED, the name parameter is OPTIONAL.

Status Code: 201, 400, 406, 409

2.4.2.2 Add new document

When adding a new section document, the request Content Type MUST be "multipart/form-data" if including metadata. In this case, the content part MUST contain the section document. The content part MUST include a Content-Disposition header with a disposition of "form-data" and a name of "content". The metadata part MUST contain the metadata for this section document. The metadata part MUST include a Content-Disposition header with a disposition of "form-data" and a name of "metadata". It is to be treated as informational, since the service MUST compute the valid new metadata based on the requirements found in the HRF specification. The content media type MUST conform to the media type of either the section or the media type identified by metadata of the section document. For XML media types, the document MUST also conform to the XML schema identified by the extensionId for the

section or the document metadata. If the content cannot be validated against the media type and the XML schema identified by the content type of this section, the server MUST return a status code of 400.

If the request is successful, the new section document MUST show up in the document feed for the section. The server returns a 201 with a Location header containing the URI of the new document.

Status Code: 201, 400.

2.4.3 PUT

This operation is not defined by this specification.

Status Code: 405, unless an implementer defines this operation.

2.4.4 DELETE

This operation SHOULD be implemented, but special precaution should be taken: if a DELETE is sent to the section URL, the **entire** section, its documents, and subsections are completely deleted. Future requests to the section URL MUST return a status code of 404, unless the record is restored. If successful the server MUST return a status code of 204.

Status Code: 204, 404

2.5 *baseURL/sectionpath/documentname*

2.5.1 GET

This operation returns a representation of the document that is identified by *documentname* within the section identified by *sectionpath*. The *documentname* is typically assigned by the underlying system and is not guaranteed to be identical across two different systems. Implementations MAY use identifiers contained within the infoset of the document as *documentnames*.

If no document of name *documentname* exists, the implementation MUST return a HTTP status code 404.

Status Codes: 200, 404

2.5.2 PUT

This operation is used to update a document. The content MUST conform to the media type identified by the document metadata or the section content type. For media type application/xml, the document MUST also conform to the XML schema that corresponds to the content type identified by the document metadata or the section. If the parameter is incorrect or the content cannot be validated against the correct media type or the XML schema identified by the content type of this section, the server MUST return a status code of 400.

If the request is successful, the new section document MUST show up in the document feed for the section. The server returns a 200.

Status Code: 200, 400.

2.5.3 POST

This operation is used to replace metadata on a section document. This operation SHOULD NOT be used unless necessary for replicating information within an organization. If a section document is copied from one system to another, a new document metadata instance MUST be constructed from the original metadata according to the rules in the HRF specification.

The request Media Type MUST be application/xml. The body MUST contain the document metadata. It MUST conform to the XML schema for the document metadata, defined in **Invalid source specified.**. If the metadata is not of media type application/xml or it cannot be validated against the document metadata XML schema, the server MUST return a status code of 400.

If the request is successful, the document metadata for the section document MUST be updated. The server returns a 201.

Status Code: 201, 400.

2.5.4 DELETE

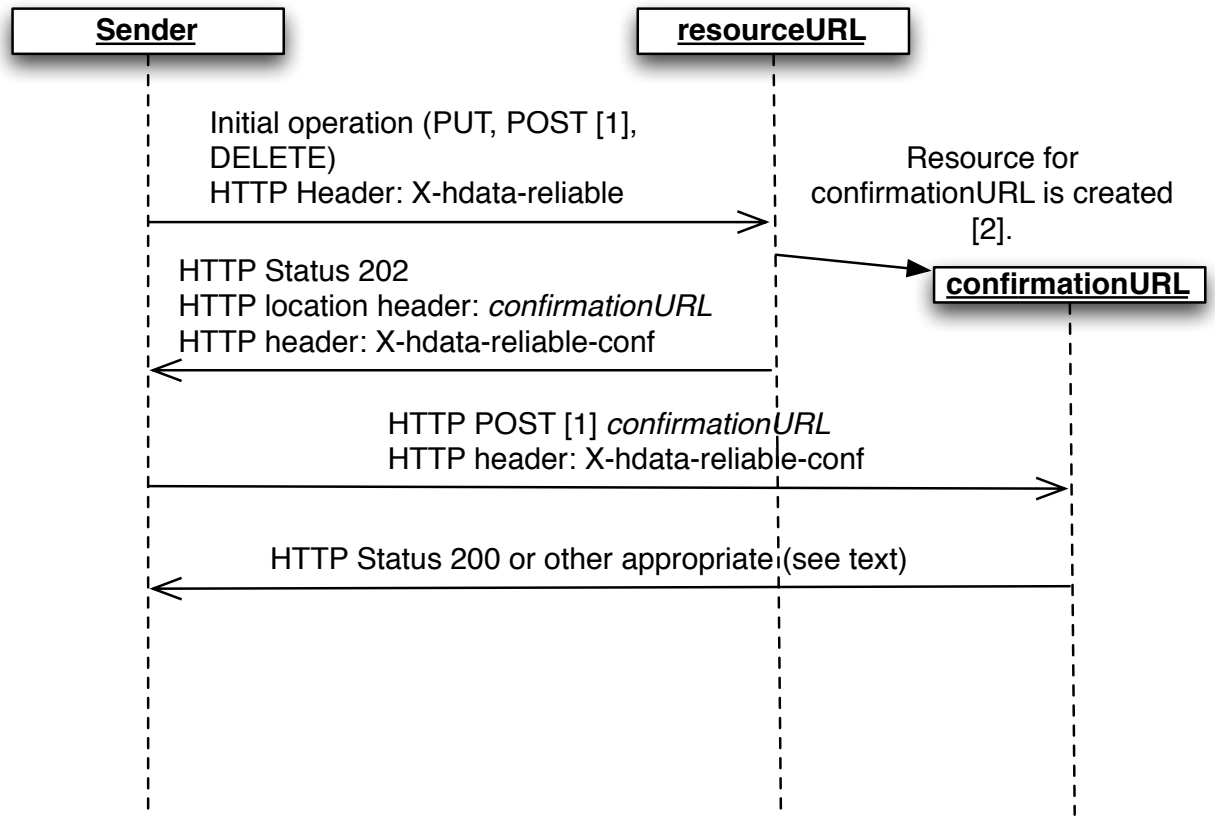
This operation SHOULD be implemented, but special precaution should be taken: if a DELETE is sent to the document URL, the document is completely deleted. Future requests to the section URL MAY return a status code of 410, unless the record is restored.

Status Code: 204, 410

3 Reliable Operations

This pattern is a complex multi step exchange, which is applicable to situations where a multi-phase commit is required. This pattern MAY be combined when interacting with an hData Record or with other message patterns, as long as there is no overloading of HTTP methods.

The use of the reliable operations pattern will be governed by the business requirements of the business domain.



[1] All POST methods must be implemented to support idempotency, e.g. through mechanisms like "Post Once Exactly" (POE).

[2] The confirmationURL may be identical to the resourceURL for document transactions.

Please see the text for more details on the interactions.

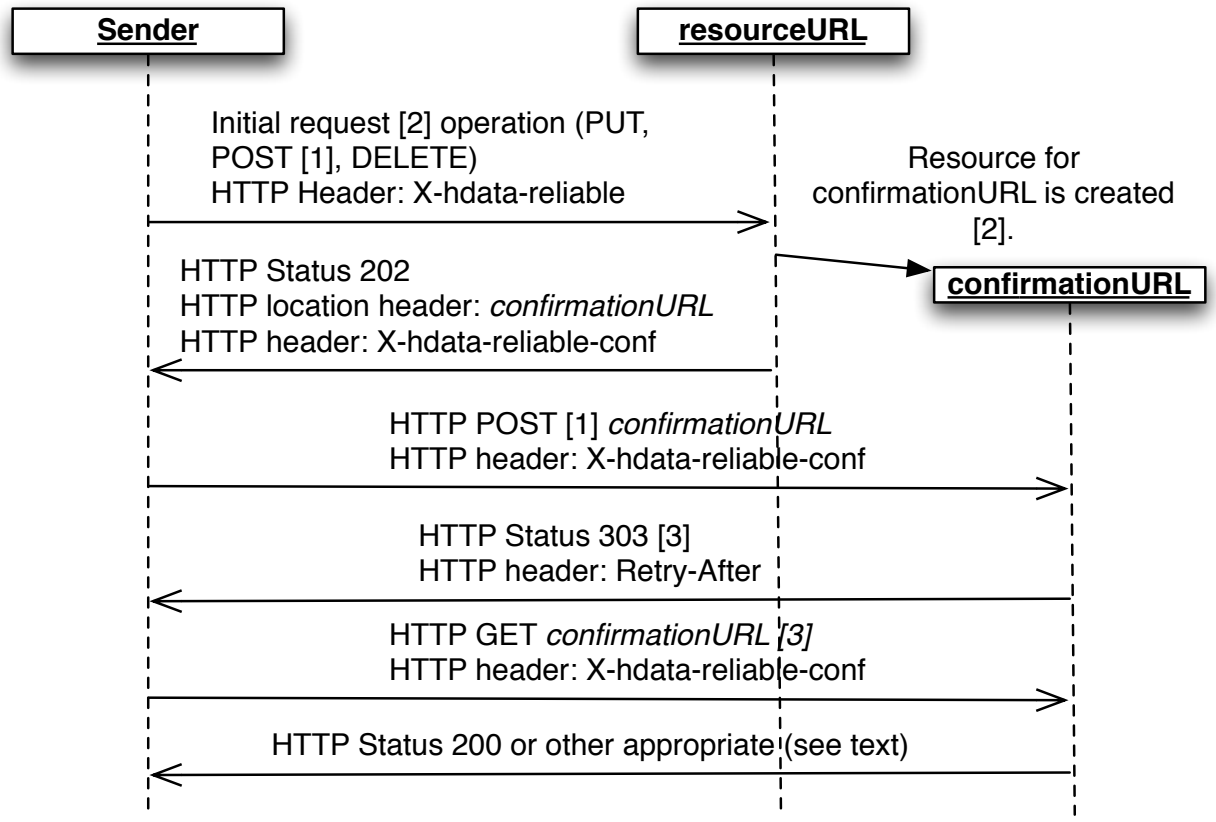
The flow of the patterns is as follows:

1. The sender accesses the *resourceURL* resource using PUT, POST, or DELETE. To indicate that it wants to use the reliable operations pattern, it sets the HTTP message header "X-hdata-reliable".
2. If the *resourceURL* is capable of performing the reliable operations pattern, it will create a new resource for a message at *confirmationURL*, and return an HTTP status code of 202. The HTTP result MUST contain the *confirmationURL* in the HTTP location header and a confirmation secret in the "X-hdata-reliable-conf" header. This secret SHOULD be a simple string of sufficient length to prevent guessing. The service MUST NOT process the message at this stage.
   If the *resourceURL* does not implement the reliable operations pattern, it MUST return an HTTP status code of 405 and discard the message.

3. The sender MUST then POST an empty request body to the resource at *confirmationURL* and set the "X-hdata-reliable-conf" header to the value provided in step 2. Upon receipt, the service – listening at the *confirmationURL* – MUST validate the confirmation secret. Once the GET secret is validated, the service processor MUST process the message immediately.
4. If the validation is successful, the *confirmationURL* returns an HTTP result with the expected status code for the initial operation. If the validation is not successful, the service MUST return an HTTP status code of 409. The sender MUST retry the POST until it receives either a different HTTP status code.

Remarks:

1. Since POST is not idempotent, the service MUST implement a safe guard against duplicity of requests for all posts in this flow. It is RECOMMENDED that the service implements "POST Once Exactly" (POE), as described in http://www.mnot.net/drafts/draft-nottingham-http-poe-00.txt.
2. The *confirmationURL* resource MAY be destroyed after the reliable message pattern flow is complete. The service MAY maintain the *confirmationURL* after the pattern flow completes.
3. If the initial operation in step 1 above is an application-level request message or document, the *confirmationURL* MAY provide an application-level response in step 4. The response MAY be provided by returning the response body in the final step; the HTTP status code MUST NOT be 409. For asynchronous responses, the *confirmationURL* MAY return an HTTP status 303 with a "Retry-After" header, indicating when the response will be available through a GET operation at the *confirmationURL*.



[1] All POST methods must be implemented to support idempotency, e.g. through mechanisms like "Post Once Exactly" (POE).

[2] The request/response protocol is defined at the application level and not through this specification. The Sender and the service at the resourceURL will determine if the operation is a request.

[3] The 303/Retry-After step is optional. It MAY be used for asynchronous responses.

Please see the text for more details on the interactions.

This specification does not provide guidance to what constitutes an application-level request/response protocol. Implementers of this specification can decide if this mechanism is appropriate for their application.

4. There is no default for how long the *confirmationURL* resource is *available for* confirmation (step 3). The service MAY destroy the *confirmationURL* resource and discard the message if the sender does not complete step 3 of the pattern flow. It is strongly RECOMMENDED to advertise the maximum time for confirming the message to the developer of the sender in the documentation for the service. If the service discards the message after timing out *the confirmation* step, it MUST return a status code of 404 at the *confirmationURL* permanently. If

the service issued a "Retry-After" header in response (as indicated in Remark 3.), it MUST provide the confirmationURL until after the expiration of the time indicated by this header.

5. For operations on hData Records (as described in section 2) special provision MUST be taken to prevent alteration of the resource once the reliable message pattern is initiated. The service MUST provide the old status of the resource until step 3 completes. It is RECOMMENDED to use the resource URL (which is different from the URL for the metadata for the resource URL) also as the *confirmationURL*.

4 Security Considerations

This transport and API specification can be used to transfer data in many different situations, for example, inside organizations, between organizations, or from medical devices. As such, the specification cannot provide a comprehensive security solution that addresses the needs of all possible applications. However, this section describes a number of basic security mechanisms that hData implementations MUST support. In addition, this section describes general web security considerations and how additional security mechanisms and systems can be added to implementations of this standard. Implementers of hData are advised to review their domain specific security requirements and select or create appropriate security mechanisms. The section concludes with a discussion of risk analysis, which is highly recommended prior to implementing and deploying any infrastructure for clinical systems.

4.1 Security Mechanism Specification

To allow the support of multiple security mechanisms at a single HRF resource, clients MUST be able to always access the *baseURL* through an HTTP OPTIONS request (see [8], section 9.2). If the resource employs any security mechanism with the exception of transport security (see 4.2.1), it MUST include the HTTP header X-hdata-security which MUST contain a comma separated list of URL-encoded URIs that identify the supported security mechanism. Section 4.2 includes the URIs for the baseline security mechanisms.

It is RECOMMENDED that hData Content Profiles include a detailed specification of any required custom security mechanisms. If the custom security mechanism The URIs for identifying these additional security mechanisms SHOULD be made unique by using the DNS domain name in the first part of the URI.

Any new security mechanism specification that is compliant with this standard needs to provide the following items. This SHOULD be done through a commonly readable text document, such as HTML. This package provides implementers with the necessary security protocol information to create the security mechanism for their system.

1. Common Name (REQUIRED) – free text, recommended to be less that 32 characters

2. Identifier (REQUIRED) – URI, recommended to include the originating organizations DNS domain name for uniqueness. NOT REQUIRED for transport security (see 4.2.1). It is RECOMMENDED to use a URL that resolves into the HTML representation of the security mechanism specification.
3. Exclusiveness (REQUIRED) – free text, describes if the mechanism can be combined with other mechanism
4. Description (REQUIRED) – free text, includes a comprehensive description of all allowed interaction patterns, parameters, and dependencies
5. State diagram (RECOMMENDED) – UML state diagram, identifies all actors and illustrates all allowed interaction patterns
6. Business rules (RECOMMENDED) – free text, describes the business/domain justification and rules for this security mechanism
7. Example (RECOMMENDED) – free text, recommended to include examples including packet content for all interaction patterns
8. Other Content (OPTIONAL)

4.2 Baseline Security

The mechanisms described in this section MUST be supported by all implementation of this specification. While transport security is always RECOMMENDED, there can be situations where transport security is not required.

The versions of IETF standards selected within this specification are the minimal REQUIRED versions. It is RECOMMENDED to use more modern versions, as long as these newer versions are backward compatible.

4.2.1 HTTP Transport Security

Transport security is implemented within the network stack below the HTTP transport layer.

1. Common Name: HTTP Transport Security
2. Identifier: none – Not required because the identifier is encoded in the *baseURL* URL through the https scheme.
3. Exclusiveness: This mechanism can be combined with all other security mechanism.
4. Description: Implementations MUST support TLS 1.1 or higher. This protocol is described in detail in IETF RFC 4345 [2]. TLS supports both anonymous clients, as well as client authentication. Implementations of this specification MUST support anonymous client, and MUST support client authentication through TLS. If TLS client authentication is supported, implementation MAY use the principal obtained from the exchange in their authentication and authorization process.

4.2.2 Message Security

1. Common Name: S/MIME Message security
2. Identifier: http://www.hl7.org/hdata/2011/03/security/smime-messages
3. Exclusiveness: This mechanism can be combined with all other security mechanisms.

4. Description: Implementations MUST support S/MIME 3.2 or higher which is an IETF internet standard described in IETF RFC 5751 [4]. S/MIME requires PKI certificates for sender and receiver, and also a way for the sender to discover the public key certificate for the receiver. The sender should include its own certificate in the S/MIME message. Implementations MUST use SHA-256 and RSA for signature and encryption, respectively. To achieve confidentiality, implementations MUST use the EnvelopedData content type [10], section 2.4.3. The hData SectionDocument that becomes the MIME payload of the S/MIME message MUST be prepared by the implementation according to the requirements of the S/MIME specifications, with special consideration for the MIME content type.

   While out of scope for this specification, there are a number of ways to discover the certificates:

   - If the receiver offers any web resources through https, it is RECOMMENDED to use the server certificate.
   - If any discovery services are available, it is RECOMMENDED that the metadata for the endpoint includes the public key certificate.
   - If DNS CERT resource records (IETF 4398 [5]) are available, the sender MAY use the certificate published.

4.2.3 Authentication

Authentication can be achieved through all of the mechanisms described in this section. Implementations of this specification MUST support all described authentication mechanisms, but these mechanisms MAY be disabled at deploy or runtime.

4.2.3.1 HTTP Basic Authentication

1. Common Name: HTTP Basic Authentication
2. Identifier: http://www.hl7.org/hdata/2011/03/security/http-basic-auth
3. Exclusiveness: This mechanism can be combined with all other security mechanisms. When combining with other authentication mechanisms, it SHOULD use the other mechanism's security principal for authentication and authorization.
4. Description: Implementations MUST implement HTTP Basic Authentication as specified in IETF RFC 2617 [6], section 2.

4.2.3.2 HTTP TLS Authentication

1. Common Name: HTTP over TLS
2. Identifier: http://www.hl7.org/hdata/2011/03/security/http-tls-auth
3. Exclusiveness: This mechanism SHOULD NOT be combined with other authentication security mechanisms. If combined with other security mechanisms, the principal of the client certificate, as identified by the Common Name (CN) attribute of the certificate, SHOULD be used as the security principal in all subsequent authentication and authorization decisions.
4. Description: Implementations MUST implement HTTP TLS Client Certificates as specified in IETF RFC 2246 [7], section 7.4.6.

4.3 Specifying A Custom Security Mechanism

Additional security mechanisms that can be published through the X-hdata-security header can be created as needed by the behavioral model and the application domain. It is RECOMMENDED to include or reference security mechanisms necessary for a given hData Content Profile (HCP) within the HCP package. The security mechanism description MUST comply with the template specified in section 4.2.

4.4 General Web Security Considerations

Because hData is implemented using common web technology, it is subject to the same security considerations as other security-sensitive web applications and services. Because Internet threats and vulnerabilities are constantly evolving, hData implementations should apply current best practices to assure appropriate levels of security.

These security best practices should be considered not only at the software application layer, but also at lower layers such as the network layer and physical layer. For example, hData implementations MAY also support lower-level protection mechanisms, such as IPSEC or other bulk traffic encryption. Typically, such technologies have no direct impact on the application layer, and their use and implementation is determined by the networking infrastructure. Protection of critical infrastructure services such as DNS or DHCP MAY be necessary. Information security must be integrated with non-IT security as well:

- Any information processing systems must be protected from intentional and unintentional physical harm, both man-made as well as natural.
- Business processes and non-IT workflow must integrate with information security, and prevent circumvention of information security measures.
- System operators and end users must be cleared for access at the appropriate level.

The reader is advised to consult appropriate resources in this area for more information, such as NIST 800-12, NIST 800-14, ISA-99, and ISO 27002.

4.5 Risk Assessment Approach and Best Practices

It is highly RECOMMEDED to perform a comprehensive risk analysis prior to deploying any clinical application. Risk analysis is a systematic consideration of the threats, vulnerabilities, and consequences of gaps in security, as well as mitigation strategies for risks. Often, the threats and vulnerabilities are captured in terms of specific scenarios that can be re-used during security audits throughout the system's lifecycle. The reader is advised to consult appropriate resources for more information on cyber risk assessment, such as NIST 800-30, the IHE security cookbook [11], and ISO/TS 25238,

5 Realization of RLUS Profiles

5.1 Introduction

The Retrieve, Locate, Update Service (RLUS) Specification defines an HL7 framework for healthcare services. The hData RESTful Transport is a realization of RLUS Functional Profiles. The hData Content

Profile (HCP) [1], section 3, acts as such as a Semantic Profile in the sense of [5], section 6.1. Taken together, the two portions of the hData specification forms an RLUS Conformance profile. This section provides a mapping between the hData RESTful implementation and the RLUS framework.

5.2 Implementation of RLUS Interfaces

The RLUS specification defines a number of interfaces in [5], Section 5 "Detailed Functional Model". These are mostly implemented by the hData specification, as detailed within the table below.

RLUS Interface	hData RESTful Implementation
Create RLUS Entry (5.1.1)	Section 2.4.2.2 describes how a new SectionDocument can be created.
Update RLUS Entry (5.1.2)	Section 2.5.2 describe how an existing SectionDocument can be updated
Delete RLUS Entry (5.1.3)	Section 2.5.4 describes how a SectionDocument can be deleted.
List Conformance Profiles (5.1.4)	Section 2.2.5 describes the X-hdata-hcp header which returns a list of hData content profiles
List Semantic Signifiers (5.2.1)	The root.xml at the baseUrl contains the list of supported elements within the Extensions node. The list of Extension elements represents the list of semantic signifiers, as required by [5] 5.2.1. (The HRF specification [1] recommends URLs as identifiers for each Extension, which should resolve into a RDDL document describing the given Extension. This is consistent with the recommendation of [5] section 5.2.1 to provide an explanation for each semantic signifier.)
List Semantic Signifiers for Resource (5.2.2)	A SectionDocument resource within an HDR MUST conform to a single semantic signifier only. By default, this is the semantic signifier (<Extension>) that is defined in the root.xml for the Section in which the SectionDocument is located. This MAY be overridden in the metadata, as explained in [1], section 2.5.1.
Retrieve Semantic Signifier (5.2.3)	For any <Extension> that is a URL and resolves into a RDDL document, the necessary description can be retrieved. Thus, if an hData implementation strives to be compliant to this interface, recommendation in [1] section 2.3 to use URLs and resolve into RDDLs becomes a requirement.
Locate Resources by Resource Parameter (5.3.1)	Parameter-specific query may be implemented either over a single HDR or a collection of HDR by another specification. This is out-of-scope for the HRF and this specification.
Retrieve Resource (5.3.2)	This is implemented using a HTTP GET operation on the resource identified by its URL.
Retrieve Resource by Resource Parameter (5.3.3)	Parameter-specific query may be implemented either over a single HDR or a collection of HDR by another specification. This is out-of-scope for the HRF and this specification.

Create Resource (5.3.4)	Section 2.4.2.2 describes how a new SectionDocument can be created.
Update Resource (5.3.5)	Section 2.5.2 describe how an existing SectionDocument can be updated
Delete Resource (5.3.6)	Section 2.5.4 describes how a SectiondDocument can be deleted.

406

407 5.3 Conformance to RLUS Functional Profiles

408 Section 6 of the RLUS specification [5] defines five functional profiles:

Functional Profile	hData Realization	Comments
Location		

409

410

411 6 Bibliography

412 [1] G. Beuchelt et al., "hData Record Format", The MITRE Corporation, 2011.

413 [2] IETF RFC 4345 “Transport Layer Security (TLS) 1.1”, online at http://tools.ietf.org/html/rfc4346

414 [3] IETF Network Working Group. (2005, Dec.) IETF. [Online]. http://www.ietf.org/rfc/rfc4287.txt

415 [4] IETF Network Working Group “S/MIME 3.2 Message Specification”, online at
416 http://tools.ietf.org/html/rfc5751

417 [5] IETF Network Working Group, “Storing Certificates in the Domain Name System (DNS)”, online at
418 http://tools.ietf.org/html/rfc4398

419 [6] IETF Network Working Group, “HTTP Authentication: Basic and Digest Access Authentication”, online
420 at http://tools.ietf.org/html/rfc2617

421 [7] IETF Network Working Group “The TLS Protocol”, online at http://tools.ietf.org/html/rfc2246

422 [8] IETF Network Working Group “Hypertext Transfer Protocol – HTTP 1.1”, online at
423 http://tools.ietf.org/html/rfc2616

424 [9] HL7 Resource Location and Updating Service (RLUS), DSTU Release 1, Health Level Seven, Inc.,
425 December 2006

426 [10] "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification ", RFC
427 3851, The Internet Society, July 2004, online at http://www.rfc-editor.org/rfc/rfc3851.txt

[11] “Cookbook:Preparing the IHE Profile Security Section”, IHE International, October 2008, online at http://www.ihe.net/Technical_Framework/upload/IHE_ITI_Whitepaper_Security_Cookbook_2008-11-10.pdf