hData RESTful API Specification v0.8

Editor: Gerald Beuchelt

Contributors: Robert Dingwell, Andrew Gregorowicz, Marc Hadley, Harry Sleeper

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

1 Introduction

The hData Record Format (HRF) [1] describes an XML representation of the information in an electronic health record (EHR). It also contains a glossary of terms used in this specification. This specification defines a network transport API for accessing components of an HRF and sending messages to an EHR system.

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 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-like 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 API

Any HDR can be represented as HTTP resources in a canonical way. The entire HDR is referenced by a base URL which depends on the implementation. See IETF RFC 2396, section 5 for more details. This base URL will be denoted as *baseURL* throughout this document. This specification does not dictate the format of the base URL. All content within an HDR MUST be expressible as a HTTP resource. In the following, the minimum version for HTTP is 1.1, unless explicitly specified otherwise.

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.

Any GET, PUT, POST, or DELETE operations on a given resource that are either (i) unspecified or (ii) not implemented MUST return an HTTP status code of 405, and include the allowed methods. All operations may return HTTP status codes in the 5xx range if there is a server problem.

2.1 Operations on the Base URL

2.1.1 GET

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

Status Code: 404

2.1.1.1 Default

There is no defined default behavior for an HTTP GET to the *baseURL*. It is RECOMMENDED that a GET returns a human-friendly user interface that represents the content of the HDR.

Status Code: 200

2.1.1.2 Parameter: type

The parameter "type" MUST have value "sections". The server MUST return 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.

If type has any other value, the server MUST return a 404 status code.

Status Code: 200, 404

2.1.2 POST

2.1.2.1 Parameters: type, contentType, path, name

The type parameter MUST equal "section". The contentType MUST be a string that is registered in the extensions node 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 contentType is not registered as a valid extension, the server MUST verify that it can support this extension. It is RECOMMENDED that this contentType is a URL. 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. 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

2.1.3 PUT

This operation MUST NOT be implemented.

Status Code: 405

2.1.4 DELETE

This operation MAY be implemented, but special precaution should be taken: if a DELETE is sent to the *baseURL*, the **entire** HDR represented by the *baseURL* is completely deleted. Future requests to the *baseURL* SHOULD return a status code of 410, unless the record is restored. Any DELETE operation MUST by logged by the underlying system.

Status Code: 204, 410

2.2 *baseURL*/root.xml

2.2.1 GET

2.2.1.1 Default

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

Status Code: 200

2.2.2 POST, PUT, DELETE

These operations MUST NOT be implemented.

Status Code: 405

2.3 *baseURL/sectionpath*

2.3.1 GET

2.3.1.1 Default

This operation MUST return an Atom 1.0 [3] compliant feed of all section documents contained in this section. Each entry MUST contain a link to a resource that uniquely identifies the section document. If the section document type defines a creation time, is RECOMMENDED to set the Created node to that datetime. The Content element MUST contain the XML representation of each section document meta data (see [1], Section 2.4.1).

Status Code: 200

2.3.1.2 Parameter: type

The parameter "type" has two allowed values: "documents" and "sections". If "type" is of value "documents", the server MUST return the same content as for a Default GET to the section resource.

If "type" is "sections", the server MUST return an Atom 1.0 compliant feed of all immediate 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.

If type has any other value, the server MUST return a 404 status code.

Status Code: 200, 404

2.3.2 POST

POST operations MUST provide a type parameter which MUST be either of value "section" or "document". For "section", three additional parameters are required, and the POST will create a new child section within this section. For "document" a document MUST be sent that conforms to the business rules expressed by the extension that the section has registered.

2.3.2.1 Parameters: contentType, path, name (Add new section)

If the content type equals “application/x-www-form-urlencoded” this POST method MUST create a new sub section. If the contentType is not registered as a valid extension, the server MUST verify that it can support this extension. It is RECOMMENDED that this contentType is a URL. 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. 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. 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.

Status Code: 201, 400, 409

2.3.2.2 Parameters: content (Add new document)

When adding a new section document, the request Media Type MUST be multipart/mixed if including meta data. In this case, the content part MUST contain the section document. The metadata part MUST contain the meta data for this section document. It is to be treated as informational, since the service MUST compute the valid new meta data 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 meta data of the section document. For the application/xml media type, the document MUST also conform to the XML schema identified by the content type in for the section or the document meta data. If the parameter is incorrect or 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.

Status Code: 201, 400.

2.3.3 PUT

This operation MUST NOT be implemented.

Status Code: 405

2.3.4 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 MAY return a status code of 410, unless the record is restored. Any DELETE operation MUST by logged by the underlying system.

Status Code: 204, 410

2.4 *baseURL/sectionpath/documentname*

2.4.1 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 stable 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

2.4.2 PUT

This operation is used to update a document. The content MUST conform to the media type identified by the document meta data 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 meta data 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.

2.4.3 POST

This operation is used to replace meta data 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 meta data instance SHOULD be constructed from the original meta data. The derived attribute on the DocumentMetaData/Source node SHOULD then be set to true, and a link to the original document SHOULD be included.

Status Code: 200, 400

2.4.3.1 Parameters: metadata

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 meta data, defined in [1]. If the metadata is not of media type application/xml or it cannot be validated against the document meta data XML schema, the server MUST return a status code of 400.

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

Status Code: 201, 400.

2.4.4 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. Any DELETE operation MUST by logged by the underlying system.

Status Code: 204, 410

3 RESTful Message API

It is RECOMMENDED to utilize the hData Record API for all transactions. If this is not possible, the Message API in this section can be used. It provides a generic approach to sending and receiving messages through RESTful services.

In the following sub-sections, xxx different patterns are described to allow (i) sending traditional messages (such as HL7 v2/v3 messages) and (ii) lightweight messages to EHR systems.

3.1 General Requirements

It is RECOMMENDED to use transport layer security for all transactions. It is RECOMMENDED to use server and client authentication in TLS. If a method is not supported by any resource, it MUST return a 405 status code, including a list of allowed methods.

3.2 Simple Patterns

3.2.1 Default Operations

All patterns in this section MUST support implement the following methods.

3.2.1.1 POST

All messages MUST be sent to a single endpoint through a POST, identified by a URL. The endpoint MUST to process the input and determine if it can process the message. If successful, the service MUST return a 201 status code. It MAY include a location header with a URL to identify a persistent record for the message.

If the message cannot be processed, the service should return a 400 status code.

Status Code: 200, 400

3.2.1.2 GET

The resource MAY allow the GET method to support reliable sending of messages – see section 3.3.1.

3.2.1.3 PUT, DELETE

These methods SHOULD NOT be allowed.

Status Code: 405

3.2.2 Single Endpoint Pattern

This pattern is intended for easy integration with existing message processors. It exposes a message endpoint directly as a resource. All message processing including validation are performed by the resource.

There is no guidance on the naming of the resource. It is RECOMMENDED to use a simple URL without any parameters.

Pattern: *baseURL*

Example (non-normative): https://example.com/input - All messages for Example Corp. go to a single endpoint and the service needs to triage all messages to the correct destination.

Example (non-normative): https://example-hie.com/example_com/input - In this case, all messages for Example Corp. would be managed by Example HIE, Inc. Everything else would be the same as above.

3.2.3 Standard Restricted Pattern

This pattern SHOULD be used if the message processor is specific to some message format standard. As long as the message POSTed to this resource is compliant with the standard identified through the standard identifier, the resource MUST return a 200 status code. If the message can not be processed because it is not implemented by the message processor, it is RECOMMENDED to have the message processor return a response in the HTTP body that indicates failure to process.

Pattern: *baseURL*/<standard>

The <standard> identifies the message format. The following table contains the REQUIRED <standard> identifiers for HL7 standards:

Standard	Identifier	Example (non-normative)
HL7 v2	hl7v2	https://example.com/hl7v2
HL7 v3	hl7v3	https://example.com:8080/base/hl7v3
muITS	muITS-<subidentifier>	https://example.com/muITS-hdata

The subidentifier for the muITS identifier MUST be specified by the muITS.

3.2.4 Standard and Content Class Restricted Pattern

This pattern SHOULD be used to simplify processing. The message processor at this resource MUST process messages compliant to the specified standard. It MUST only process messages that fall within the content class of the standard.

Patterns: *baseURL*/<standard>/<content class>

The content class identifier is out of the scope of this document.

For HL7 v2 and HL7 v3 messages the content class identifier MAY be identical to the message name. It is case sensitive. If the message name contains characters that are not allowed in a URI (per IETF RFC 3986), these characters MUST be percent-encoded.

For HL7 v2 messages, the content class identifier MAY be prepended with either “xml/” or “text/” to indicate XML or ASCII encoding.

Systems MAY allow URL parameters to provide processing guidance to the message processor. All resources SHOULD support the following guidance parameters:

URL Parameter	Use
version	Indicate a specific version of a standard or content class

Examples (non-normative):

https://example.com/hl7v2/xml/ADT%5EA01/?version=2.5 -- admit discharge transfer in xml

https://example.com/hl7v3/PRPA%5FRM201301UV02 -- patient activate

https://example.com/muITS-hdata/admitPatient -- admit discharge transfer

3.2.5 Content Class Only Restricted Pattern

This pattern is very similar to the one described in section 3.2.4, but the standard identifier is omitted. The message processor MUST process messages belonging to the identified content class, only.

Pattern: *baseURL*/<content class>

Example (non-normative):

https://example.com/ADT%5EA01/?version=2.5 -- admit discharge transfer

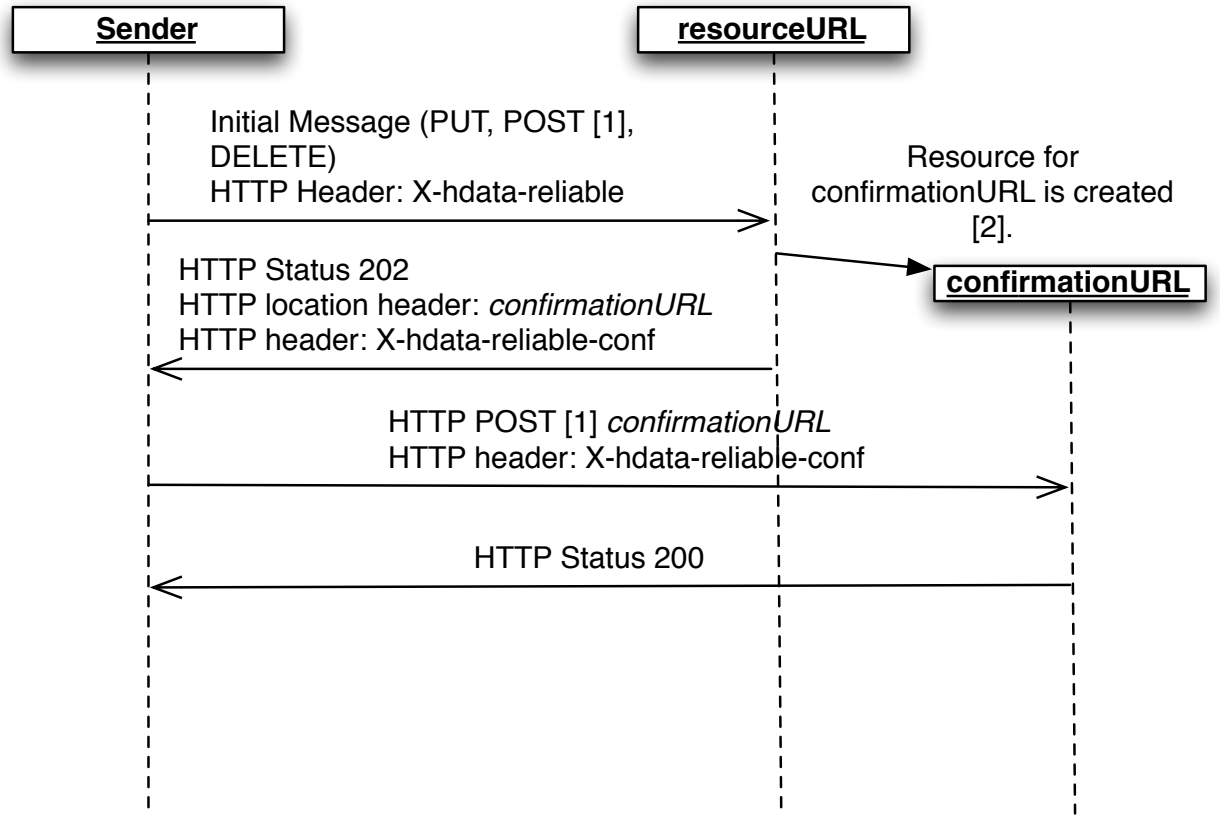
https://example.com/PRPA%5FRM201301UV02 -- patient activate

https://example.com/muITS-hdata/admitPatient -- admit discharge transfer

3.3 Special Messaging Patterns

3.3.1 Reliable Messaging

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.



[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.

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 message pattern, it sets the HTTP message header "X-hdata-reliable".
2. If the *resourceURL* is capable of performing the reliable message 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 messaging pattern, it MUST return an HTTP status code of 405 and discard the message.
3. The sender MUST then POST 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 is validated, the service processor MUST process the message immediately.

4. If the validation is successful, the *confirmationURL* returns an HTTP result with status code 200. If the validation is not successful, the service MUST return an HTTP status code of 405.

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. It MAY provide the content of the message through the GET method.
3. 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.
4. 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 meta data for the resource URL) also as the *confirmationURL*.

3.3.2 Broadcast Messaging

This pattern is RECOMMENDED for situations where a sender needs to notify more than one service. It MUST use the Atom 1.0 publishing protocol, which requires the recipient of messages to subscribe to the sender's publication feed. This pattern cannot be applied to situations where a sender initiated "push" is required, or where it is impractical to publish the sender's messages at a well-known URL.

Example (non-normative):

https://example.com/admitPatient/newPatient.feed

3.4 Complex Messaging Example (Non-Normative)

3.4.1 Reserve New Bed Example (Non-Normative)

URL: https://example.com/beds/ward1

Sender: POST <<Request for a new bed>>

Resource: HTTP 200

location-header: https://example.com/beds/ward1/32

```
<bed>
  <ward>1</ward<
  <number>32</number>
</bed>
```

Sender:

PUT https://example.com/beds/ward1/32

```
<patient>
  <name>John Doe</name>
  <identifier>http://example.com/patients/12345</identifier>
</patient>
```

3.4.2 Admission Message Example (Non-Normative)

TBD

4 Security Considerations

While not required by this standard, it is RECOMMENDED that all HTTP methods on resources are properly protected.

5 Bibliography

[1] G. Beuchelt, R. Dingwell, A. Gregorowicz, and H. Sleeper, "hData Record Format," The MITRE Corporation, 2009.

[2] PKWare, Inc. .ZIP Application Note. [Online]. http://www.pkware.com/support/zip-application-note

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