

hData RESTful API Specification v0.13

Editor: Gerald Beuchelt

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

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

© 2009-10 The MITRE Corporation. All rights reserved.

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. The hData Record Format (HRF) [1] describes an XML representation of the information in an electronic health record (EHR) and contains a glossary of terms used in this specification.

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 notation (e.g.,

25 /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element
26 wildcard. The use of @{any} indicates the presence of an attribute wildcard.

27 **2 hData Record RESTful API**

28 **2.1 Overview**

29 Any HDR can be represented as a set of HTTP resources in a canonical way. The entire HDR is referenced
30 by a base URL which depends on the implementation. See IETF RFC 3986, section 5 for more details. This
31 base URL will be denoted as *baseURL* throughout this document.

32 **2.1.1 Out of Scope**

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

38 **2.1.2 General Conventions**

39 Any GET, PUT, POST, or DELETE operations on a given resource that are either (i) unspecified or (ii) not
40 implemented MUST return an HTTP response with a status code of 405 that includes an Allow header
41 that specifies the allowed methods. All operations may return HTTP status codes in the 5xx range if
42 there is a server problem.

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

46 **2.2 Operations on the Base URL**

47 **2.2.1 GET**

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

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

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

56 Status Code: 200, 404

57 **2.2.2 POST – Parameters: extensionId, path, name**

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

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

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

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

73 Status Code: 201, 400, 406, 409

74 **2.2.3 PUT**

75 This operation is undefined by this specification.

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

77 **2.2.4 DELETE**

78 This operation is undefined by this specification.

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

80 **2.3 *baseURL*/root.xml**

81 **2.3.1 GET**

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

84 Status Code: 200

85 **2.3.2 POST, PUT, DELETE**

86 These operations MUST NOT be implemented.

87 Status Code: 405

88 **2.4 *baseURL/sectionpath***

89 **2.4.1 GET**

90 This operation MUST return an Atom 1.0 [3] compliant feed of all section documents and child sections
91 contained in this section. Each entry MUST contain a link to a resource that uniquely identifies the
92 section document or child section. If the section document type defines a creation time, is
93 RECOMMENDED to set the Created node to that datetime.

94 For section documents, the Atom Content element MUST contain the XML representation of its
95 metadata (see [1], Section 2.4.1).

96 Status Code: 200

97 **2.4.2 POST**

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

101 **2.4.2.1 *Add new section – Parameters: extensionId, path, name***

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

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

111 When creating the section resource, the server MUST update the root document: in the node of the
112 parent section a new child node must be inserted. The server MUST return a 201 status code.

113 Status Code: 201, 400, 406, 409

114 **2.4.2.2 *Add new document***

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

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

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

129 Status Code: 201, 400.

130 2.4.3 PUT

131 This operation is not defined by this specification.

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

133 2.4.4 DELETE

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

138 Status Code: 204, 404

139 2.5 *baseURL/sectionpath/documentname*

140 2.5.1 GET

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

145 If no document of name *documentname* exists, the implementation MUST return a HTTP status code
146 404.

147 Status Codes: 200, 404

148 2.5.2 PUT

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

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

157 Status Code: 200, 400.

158 **2.5.3 POST**

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

163 The request Media Type MUST be application/xml. The body MUST contain the document metadata. It
164 MUST conform to the XML schema for the document metadata, defined in [1]. If the metadata is not of
165 media type application/xml or it cannot be validated against the document metadata XML schema, the
166 server MUST return a status code of 400.

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

169 Status Code: 201, 400.

170 **2.5.4 DELETE**

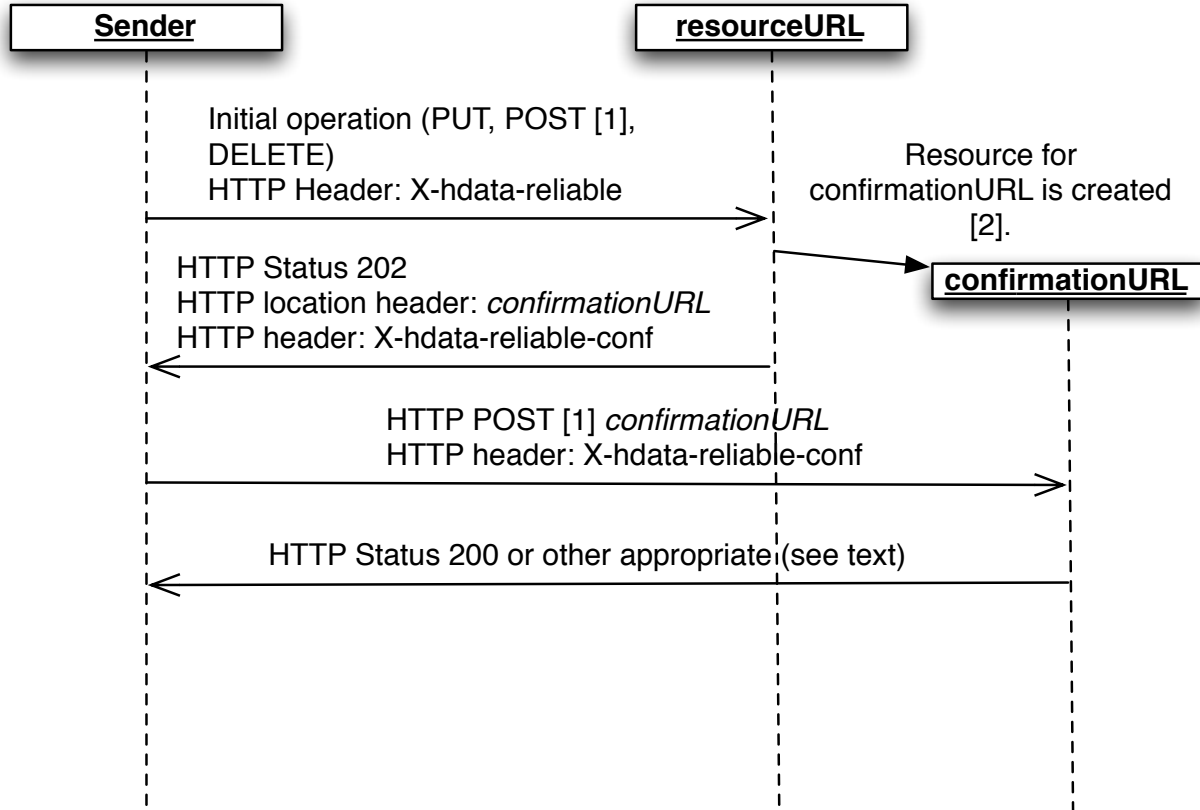
171 This operation SHOULD be implemented, but special precaution should be taken: if a DELETE is sent to
172 the document URL, the document is completely deleted. Future requests to the section URL MAY return
173 a status code of 410, unless the record is restored.

174 Status Code: 204, 410

175 **3 Reliable Operations**

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

179 The use of the reliable operations pattern will be governed by the business requirements of the
180 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.

181 Please see the text for more details on the interactions.

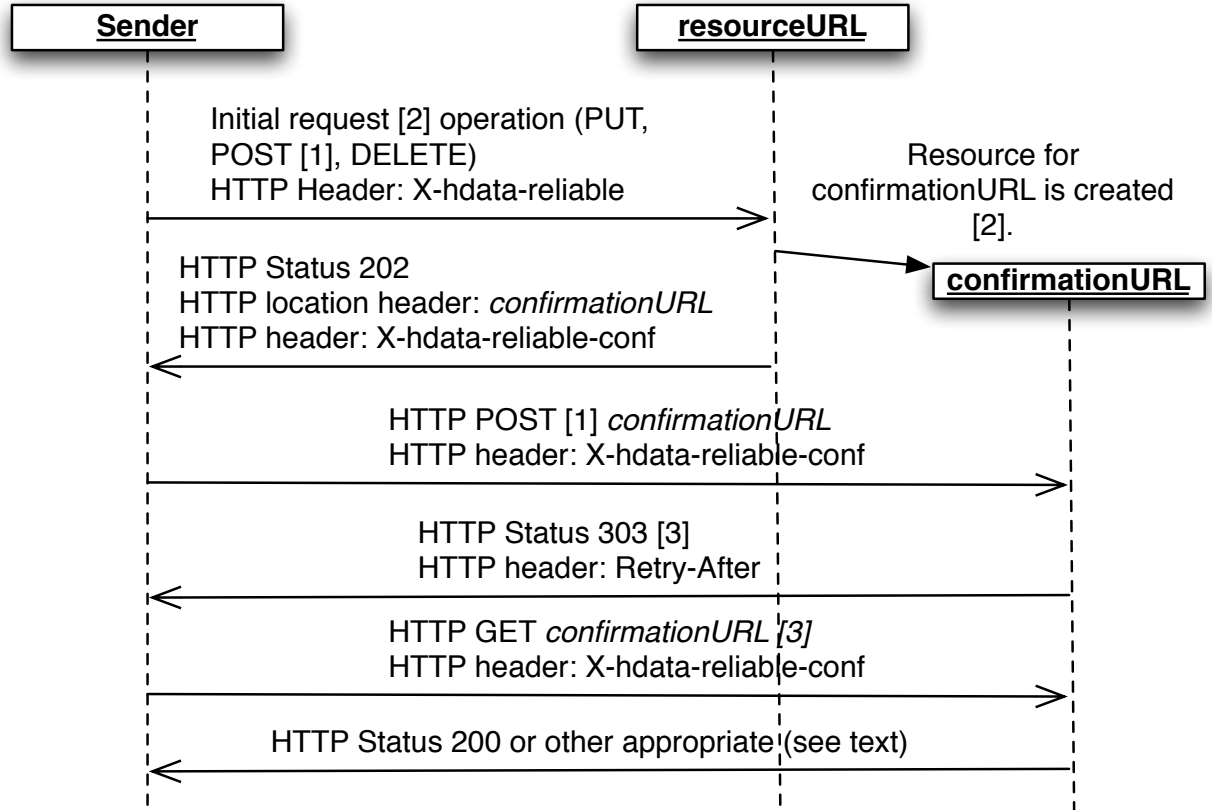
182 The flow of the patterns is as follows:

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

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

201 **Remarks:**

- 202 1. Since POST is not idempotent, the service MUST implement a safe guard against duplicity of
203 requests for all posts in this flow. It is RECOMMENDED that the service implements “POST Once
204 Exactly” (POE), as described in <http://www.mnot.net/drafts/draft-nottingham-http-poe-00.txt>.
- 205 2. The *confirmationURL* resource MAY be destroyed after the reliable message pattern flow is
206 complete. The service MAY maintain the *confirmationURL* after the pattern flow completes.
- 207 3. If the initial operation in step 1 above is an application-level request message or document, the
208 *confirmationURL* MAY provide an application-level response in step 4. The response MAY be
209 provided by returning the response body in the final step; the HTTP status code MUST NOT be
210 409. For asynchronous responses, the *confirmationURL* MAY return an HTTP status 303 with a
211 “Retry-After” header, indicating when the response will be available through a GET operation at
212 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.

213
214

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

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

224 the service issued a "Retry-After" header in response (as indicated in Remark 3.), it MUST
225 provide the confirmationURL until after the expiration of the time indicated by this header.
226 5. For operations on hData Records (as described in section 2) special provision MUST be taken to
227 prevent alteration of the resource once the reliable message pattern is initiated. The service
228 MUST provide the old status of the resource until step 3 completes. It is RECOMMENDED to use
229 the resource URL (which is different from the URL for the meta data metadata for the resource
230 URL) also as the *confirmationURL*.

231 4 Security Considerations

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

234 5 Bibliography

235

- [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>

236

237