hData Packaging and Network Transport Specification v0.6

Gerald Beuchelt, Robert Dingwell, Andrew Gregorowicz, Harry Sleeper

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

1   Introduction

The hData Record Format (HRF) [1] describes the XML representation of the continuity of care information in an electronic health record (EHR). This specification creates a compact serialization format for the entire HRF , and a network transport API for accessing components of an HRF.

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/metadata	SectionDocument metadata

1.2   Glossary (Non-Normative)

hData Record Format (HRF) - The part of the hData specification that defines the abstract hierarchy, meta-data schema, and document organization of the hData record.

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

hData Restful API (HRA) - the part of the hData specification that defines the basic HTTP-based API for accessing or modifying an HDR.

24 **hData Specification** - a normative specification that defines the HRF, the HRA, and a file-based
25 serialization format.

26 **hData Content Profile (HCP)** - a profile of the medical content of an HDR. An HCP is specified separately
27 from the HRF. The hData Project defines an initial HCP (iHCP) that covers the 35 data elements for
28 EHRs/EMRs defined by the National Quality Foundation.

29 **Electronic Medical Record (EMR)** - the medical record or records of a single patient in the IT system of
30 an actor (health provider, government entity, payer, etc.). In this definition, an HDR is a type of EMR.

31 **Electronic Health Record (EHR)** - the collection of all EMRs of a single patient, across organizational and
32 national boundaries.

33 **EHR System** - An IT system that creates, stores, and manages EMRs.

34 **Clinical Document Architecture (CDA)** - an XML specification by Health Layer 7 (HL7) that is intended to
35 be used for EMRs.

36 **Continuity of Care Record (CCR)** - a specification by ASTM that is intended to be used for
37 summary/continuity of care documentation. A CCM is a type of EMR.

38 **Continuity of Care Document (CCD)** - a profile of the CDA that accommodates the medical information
39 of the CCR.

40 **HITSP/C32 (C32)** - a constrained profile of the CCD that is intended to simplify implementation and
41 improve interoperability. There is no normative schema for C32. Note that HITSP has recently split up
42 C32 into HITSP/C80 and HITSP/C83.

43 **MITRE/L32 (L32)** - a significantly constrained profile of the C32 specification. L32 comes with a
44 normative schema and can be mapped onto the HRF.

1.3   Notational Conventions

46 The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT",
47 "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC
48 2119.

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

53

54 Note also that only the W3C XML schemas linked in Appendix A at the end of this document are
55 normative – any schema fragment or other schema description is informational only.

2   File System Serialization

56

57 An HDR can be serialized for transporting the entirety of all data contained in the HDR. Serialization is a
58 simple process that uses the ZIP archive format [2] in the same way the Open Document Format does.
59 Sections are represented as file system folders and the root document and section documents are
60 stored in as serialized XML files, using UTF-8 encoding. The following process can be applied to UNIX and
61 Windows file systems.

2.1   Section Mapping

62

63 The first step in the file system serialization process is to map existing sections to a file folder structure
64 on disk. For this, the *base folder* is selected, which MUST be completely empty. For each section directly
65 below the `<sections>` element in the root document, a new file folder with name equal to the `path`
66 attribute of the corresponding `<section>` element is created in the *base folder*.

67 The same process is then iteratively applied to each child node of the first level `<section>` nodes and
68 their children. The result is a folder hierarchy which represents the section structure of the HDR.

2.1.1   Document Meta Data Serialization

69

70 In the hData Record Format specification [1], section 2.4.1 associates meta data with each Section
71 Document. For the RESTful API, this meta data is represented within an Atom [3] feed by attaching the
72 `<hrf-md:DocumentMetaData>` fragment as foreign markup to each entry (see section 3.3.1
73 below).

74 For the purposes of the file system serialization, this Atom feed, containing all document meta data
75 fragments, is serialized into a file called "`section.xml`" within the file folder that represents the
76 particular section.

2.2   Document Serialization

77

78 The root document is serialized into the *base folder* created in section 2.1. For each `<section>` under
79 the `<sections>` node in the root document, all documents belonging into this section are
80 enumerated, serialized into XML standalone documents with UTF-8 encoding and stored in with file
81 name "*document name*.xml" in the file folder that represents the section. If two documents have an
82 identical document name, any UTF-8 encoded character-based discriminator may be appended to the
83 document name, but prior to the ".xml" extension.

84 This process is performed on all `<section>` nodes and all of their children.

2.3   Packaging

85

86 Once all sections, the root document, and all section documents are represented in the file folder
87 hierarchy, a ZIP archive [2] is created of the *base folder* and all its descendant files and folders.

3   Network Transport and 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. This base URL will be denoted as *baseURL* in the following. This specification does not dictate the format of the base URL, but it is NOT RECOMMENDED to use matrix parameterization. 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. All operations may return HTTP status codes in the 5xx range if there is a server problem.

3.1   Operations on the Base URL

3.1.1   GET

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

Status Code: 404

3.1.1.1   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

3.1.1.2   TE Header: Deflate

If the HTTP TE header contains "deflate", the server MAY return a ZIP archive containing the serialized version of the current content of the entire HDR. Implementations MAY also define other mechanisms to obtain a serialized version of the HDR, but these MUST NOT collide with any provisions of this specification.

Status Code: 200

3.1.1.3   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

119 **3.1.2   POST**

120 *3.1.2.1   Parameters: type, typeId, requirement*

121 For this operation, the value of type MUST equal "extension". The typeId MUST be a URI string that
122 represents a type of section document. The requirement parameter MUST be either "optional" or
123 "mandatory". If any parameters are incorrect or not existent, the server MUST return a status code of
124 400.

125 If the system supports the extension identified by the typeId URI string, this operation will modify the
126 extensions node in the root document and add this extension with the requirement level identified by
127 the requirement parameter. The server MUST return a 201 status code.

128 If the system does not support the extension, it MUST not accept the extension if the requirement
129 parameter is "mandatory" and return a status code of 409. If the requirement is "optional" the server
130 MAY accept the operation, update the root document and send a status code of 201.

131 Status Code: 201, 400, 409

132 *3.1.2.2   Parameters: type, typeId, path, name*

133 The type parameter MUST equal "section". The typeId MUST be an URI that is registered in the
134 extensions node of the root document of the HDR identified by *baseURL*. The path MUST be a string that
135 can be used as a URL path segment. If any parameters are incorrect or not existent, the server MUST
136 return a status code of 400.

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

139 If the typeId is not registered as a valid extension, the server MUST return a status code of 406. It MAY
140 provide additional entity information.

141 When creating the section resource, the server MUST update the root document: in the node of the
142 parent section a new child node must be inserted. The server MUST return a 201 status code and
143 SHOULD include the location of the new section.

144 Status Code: 201, 400, 406, 409

145 **3.1.3   PUT**
146 This operation MUST NOT be implemented.

147 Status Code: 405

148 **3.1.4   DELETE**
149 This operation MAY be implemented, but special precaution should be taken: if a DELETE is sent to the
150 *baseURL*, the **entire** HDR represented by the *baseURL* is completely deleted. Future requests to the
151 *baseURL* SHOULD return a status code of 410, unless the record is restored. Any DELETE operation
152 MUST by logged by the underlying system.

153   Status Code: 204, 410

3.2   *baseURL/root.xml*

3.2.1   GET

3.2.1.1   Default
157   This operation return an XML representation of the current root document.

158   Status Code: 200

3.2.2   POST, PUT, DELETE
160   This operations MUST NOT be implemented.

161   Status Code: 405

3.3   *baseURL/sectionpath*

3.3.1   GET

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

170   Status Code: 200

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

174   If "type" is "sections", the server MUST return an Atom 1.0 compliant feed of all child sections, as
175   identified in the corresponding sections node in the root document. Each entry MUST contain a link to
176   the resource for each child section.

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

178   Status Code: 200, 404

3.3.2   POST
180   Both POST operations MUST provide a type parameter which MUST be either of value "section" or
181   "document". For "section", three additional parameters are required, and the POST will create a new
182   child section within this section. For "document" an XML document MUST be sent that conforms to the
183   schema identified by the typeId attribute of this section.

184 *3.3.2.1   Parameters: type, typeId, path, name*

185 The type parameter MUST equal "section". The typeId MUST be an URI that is registered in the

186 extensions node of the root document of the HDR identified by *baseURL*. The path MUST be a string that

187 can be used as a URL path segment. If any parameters are incorrect, the server MUST return a status

188 code of 400.

189 The system MUST confirm that there is no other section registered as a child node that uses the same

190 path name. If there is a collision, the server MUST return a status code of 409.

191 When creating the section resource, the server MUST update the root document: in the node of the

192 parent section a new child node must be inserted. The server MUST return a 201 status code.

193 Status Code: 201, 400, 409

194 *3.3.2.2   Parameters: type, content*

195 When the type parameter is "document", the request Media Type MUST be multipart/mixed. The

196 content parameter MUST contain the section document. The content MUST conform to the media type

197 of either the section or the media type identified by meta data of the section document. For the

198 application/xml media type, the document MUST also conform to the XML schema identified by the

199 content type in for the section or the document meta data. If the parameter is incorrect or the content

200 cannot be validated against the media type and the XML schema identified by the content type of this

201 section, the server MUST return a status code of 400.

202 If the request is successful, the new section document MUST show up in the document feed for the

203 section. The server returns a 201.

204 Status Code: 201, 400.

205 **3.3.3   PUT**

206 This operation MUST NOT be implemented.

207 Status Code: 405

208 **3.3.4   DELETE**

209 This operation SHOULD be implemented, but special precaution should be taken: if a DELETE is sent to

210 the section URL, the **entire** section, its documents, and subsections are completely deleted. Future

211 requests to the section URL MAY return a status code of 410, unless the record is restored. Any DELETE

212 operation MUST by logged by the underlying system.

213 Status Code: 204, 410

214 **3.4   *baseURL/sectionpath/documentname***

215 **3.4.1   GET**

216 This operation returns a UTF-8 encoded representation of the document that is identified by

217 *documentname* within the section identified by *sectionpath*. The *documentname* is typically assigned by

218 the underlying system and is not guaranteed to be stable across two different systems. Implementations
219 MAY use identifiers contained within the infoset of the document as *documentnames*.

220 If no document of name *documentname* exists, the implementation MUST return a HTTP status code
221 404.

222 Status Codes: 200, 404

3.4.2   PUT

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

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

232 Status Code: 200, 400.

3.4.3   POST

234 This operation is used to replace meta data on a section document. This operation SHOULD NOT be used
235 unless necessary for replicating information within an organization. If a section document is copied from
236 one system to another, a new document meta data instance SHOULD be constructed from the original
237 meta data. The derived attribute on the DocumentMetaData/Source node SHOULD then be set to true,
238 and a link to the original document SHOULD be included.

3.4.3.1   *Parameters: metadata*

240 The request Media Type MUST be multipart/mixed. The metadata parameter MUST contain the
241 document metadata. The metadata MUST be of media type application/xml. It MUST conform to the
242 XML schema for the document meta data, defined in [1]. If the metadata is not of media type
243 application/xml or it cannot be validated against the document meta data XML schema, the server
244 MUST return a status code of 400.

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

247 Status Code: 201, 400.

3.4.4   DELETE

249 This operation SHOULD be implemented, but special precaution should be taken: if a DELETE is sent to
250 the document URL, the document is completely deleted. Future requests to the section URL MAY return
251 a status code of 410, unless the record is restored. Any DELETE operation MUST by logged by the
252 underlying system.

253   Status Code: 204, 410

4   Bibliography

255

[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

256

257