

# hData Packaging and Network Transport Specification v0.3

---

Gerald Beuchelt, Robert Dingwell, Andrew Gregorowicz, Harry Sleeper

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

© 2009 The MITRE Corporation. All rights reserved.

## 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	<a href="http://projecthdata.org/hdata/schemas/2009/06/core">http://projecthdata.org/hdata/schemas/2009/06/core</a>	Namespace for elements in this document

### 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.

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

### 45 **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.

## 56 **2 File System Serialization**

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.

### 62 2.1 Section Mapping

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.

### 69 2.2 Document Serialization

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

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

### 77 2.3 Packaging

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

## 80 3 Network Transport and RESTful API

81 Any HDR can be represented as HTTP resources in a canonical way. The entire HDR is referenced by a  
82 base URL which depends on the implementation. This base URL will be denoted as *baseURL* in the  
83 following. This specification does not dictate the format of the base URL, but it is NOT RECOMMENDED  
84 to use matrix parameterization. All content within an HDR MUST be expressible as a HTTP resource. In  
85 the following, the minimum version for HTTP is 1.1, unless explicitly specified otherwise.

86 This specification does not define any access controls to the web resources. It is RECOMMENDED that a  
87 comprehensive access control management system is always deployed with any hData installation.

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

## 91 **3.1 Operations on the Base URL**

### 92 **3.1.1 GET**

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

94 Status Code: 404

#### 95 **3.1.1.1 Default**

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

98 Status Code: 200

#### 99 **3.1.1.2 TE Header: Deflate**

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

104 Status Code: 200

#### 105 **3.1.1.3 Parameter: type**

106 The parameter "type" MUST have value "sections". The server MUST return an Atom 1.0 compliant feed  
107 of all child sections, as identified in the corresponding sections node in the root document. Each entry  
108 MUST contain a link to the resource for each child section.

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

110 Status Code: 200, 404

## 111 **3.1.2 POST**

### 112 **3.1.2.1 Parameters: type, typeId, requirement**

113 For this operation, the value of type MUST equal "extension". The typeId MUST be a URI string that  
114 represents a type of section document. The requirement parameter MUST be either "optional" or  
115 "mandatory". If any parameters are incorrect or not existent, the server MUST return a status code of  
116 400.

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

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

123 Status Code: 201, 400, 409

124 **3.1.2.2 Parameters: type, typeId, path, name**

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

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

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

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

136 Status Code: 201, 400, 406, 409

137 **3.1.3 PUT**

138 This operation MUST NOT be implemented.

139 Status Code: 405

140 **3.1.4 DELETE**

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

145 Status Code: 204, 410

146 **3.2 baseURL/root.xml**

147 **3.2.1 GET**

148 **3.2.1.1 Default**

149 This operation returns an XML representation of the current root document.

150 Status Code: 200

151 **3.2.2 POST, PUT, DELETE**

152 These operations MUST NOT be implemented.

153 Status Code: 405

### 154 **3.3 *baseURL/sectionpath***

#### 155 **3.3.1 GET**

##### 156 **3.3.1.1 *Default***

157 This operation MUST return an Atom 1.0 compliant feed of all section documents contained in this  
158 section. Each entry MUST contain a link to a resource that uniquely identifies the section document. If  
159 the section document type defines a creation type, it is RECOMMENDED to set the Created node to that  
160 datetime. The Content node MAY contain the XML representation of each section document.

161 Status Code: 200

##### 162 **3.3.1.2 *Parameter: type***

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

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

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

169 Status Code: 200, 404

#### 170 **3.3.2 POST**

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

##### 175 **3.3.2.1 *Parameters: type, typeId, path, name***

176 The type parameter MUST equal "section". The typeId MUST be an URI that is registered in the  
177 extensions node of the root document of the HDR identified by *baseURL*. The path MUST be a string that  
178 can be used as a URL path segment. If any parameters are incorrect, the server MUST return a status  
179 code of 400.

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

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

184 Status Code: 201, 400, 409

##### 185 **3.3.2.2 *Parameters: type, Content Type: application/xml***

186 When sending a section document, type MUST be of value "document", and the content MUST conform  
187 to the XML schema that corresponds to the typeId of this section. If the parameter is incorrect or the

188 content cannot be validated against the schema identified by the `typeld` of this section, the server MUST  
189 return a status code of 400.

190 If the request is successful, the new section document MUST show up in the document feed for the  
191 section. The server returns a 201.

192 Status Code: 201, 400.

### 193 3.3.3 PUT

194 This operation MUST NOT be implemented.

195 Status Code: 405

### 196 3.3.4 DELETE

197 This operation SHOULD be implemented, but special precaution should be taken: if a DELETE is sent to  
198 the section URL, the **entire** section, its documents, and subsections are completely deleted. Future  
199 requests to the section URL MAY return a status code of 410, unless the record is restored. Any DELETE  
200 operation MUST be logged by the underlying system.

201 Status Code: 204, 410

## 202 3.4 *baseURL/sectionpath/documentname*

### 203 3.4.1 GET

204 This operation returns a UTF-8 encoded representation of the document that is identified by  
205 *documentname* within the section identified by *sectionpath*. The *documentname* is typically assigned by  
206 the underlying system and is not guaranteed to be stable across two different systems. Implementations  
207 MAY use identifiers contained within the infoset of the document as *documentnames*.

208 If no document of name *documentname* exists, the implementation MUST return a HTTP status code  
209 404.

210 Status Codes: 200, 404

### 211 3.4.2 PUT

212 This operation is used to update a document. The content MUST conform to the XML schema that  
213 corresponds to the `typeld` of this section and the media type MUST be `application/xml`. If the parameter  
214 is incorrect or the content cannot be validated against the schema identified by the `typeld` of this  
215 section, the server MUST return a status code of 400.

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

218 Status Code: 200, 400.

### 219 3.4.3 POST

220 This operation MUST NOT be implemented.

221 Status Code: 405

#### 222 **3.4.4 DELETE**

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

227 Status Code: 204, 410

## 228 **4 Bibliography**

229

[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>

230

231