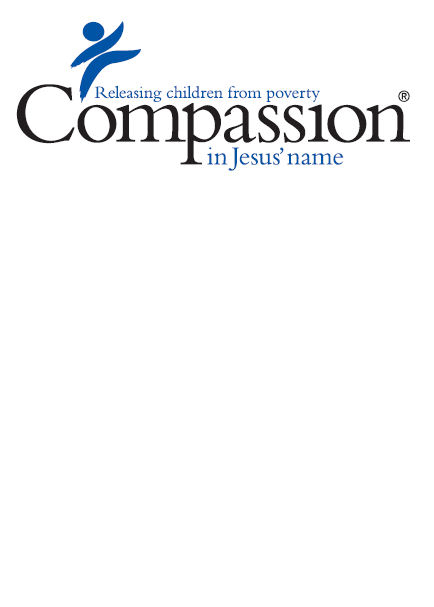
ESD Enterprise Services

REST Messaging Standard v1.2



# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author(s) |
| 05/28/2012 | 0.1 | Initial Document | Muthuraman Muthukkaruppan |
| 06/04/2013 | 0.2 | Changed SendersTrackingId, SendersAppName and SendersName to be optional fields.  Removed the GUID from SendersTrackingId | Suzanne Swanson |
| 04/02/2014 | 1.0 | Revised to incorporate web standards more fully | Bill Spafford |
| 08/22/2014 | 1.1 | Revised to incorporate input from stakeholders and potential service users | Bill Spafford |
| 9/9/2014 | 1.2 | Changes incorporated from Architect and Tech Lead meeting | Bill Spafford |

# Key Stakeholders & Organizational Signoffs

Stakeholders shall vary by project. Use “approval needed” column to indicate signoff/approval is required of a key stakeholder

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Title | Functional Area | Approval Needed |
| Les Butler | ESD Product Manager | ESD | Yes |
| Lester Henderson | Sr. Architect | Enterprise Solutions | Yes |
| Bill Spafford | Technical Lead | ESD | Yes |
| Brandon Colston | Technical Lead | ESD | Yes |
| Scott Kelso | Lead / Designer | ESD |  |
| Mike Maricevic | Project Manager | ESD |  |
| Suzanne Swanson | Business Analyst | ESD |  |

Contents

[Revision History 2](#_Toc392841788)

[Key Stakeholders & Organizational Signoffs 2](#_Toc392841789)

[1. Overview 4](#_Toc392841790)

[2. Introduction 4](#_Toc392841791)

[3. REST Operations 5](#_Toc392841792)

[Retrieve Resource 5](#_Toc392841793)

[Create Resource 6](#_Toc392841794)

[Update Resource 6](#_Toc392841795)

[Delete Resource 7](#_Toc392841796)

[Retrieve Header Data 7](#_Toc392841797)

[4. Uniform Resource Identifiers (URIs) 8](#_Toc392841798)

[URI Syntax 8](#_Toc392841799)

[URI Examples 9](#_Toc392841800)

[5. Resource Representations 9](#_Toc392841801)

[XML Representation in REST 9](#_Toc392841802)

[JSON Representation in REST 9](#_Toc392841803)

[Adding to Existing Namespaces 15](#_Toc392841804)

[6. HTTP Status Codes 15](#_Toc392841805)

[Internal Error Response Mapping to HTTP Response Codes 16](#_Toc392841806)

[7. HTTP Headers 18](#_Toc392841807)

[8. Outstanding Questions 19](#_Toc392841808)

# Overview

This document will define a standard for enterprise REpresentational State Transfer (REST) services and messaging at Compassion International. It is a living document and is expected to be updated and revised as more is understood about the needs for REST services throughout Compassion’s global ministry. It will define the expected structure of the REST API, including formation of URIs, use of HTTP verbs, Status Codes, header information for requests and responses, and the format of the message body.

At a high level, the model described by URIs, parameters and resource representations should reflect the canonical data model in use by Compassion International rather than being specific to source systems.

# Introduction

At their most basic level, REST APIs provide access to Compassion resources via Uniform Resource Identifier (URI) paths. A resource is a set of related information, such as a beneficiary object, a constituent profile, a global identifier or a collection of items. To access a resource for retrieval or perform some operation on it (i.e. update, delete), an application calls a supported HTTP verb on the resource URI.

In addition to using URIs to identify resources, this version of the REST standard will support use of HTTP headers for content negotiation. For example, the client will use the “Accept” header to indicate requested content format and both client and server can use the “Content-Type” headers to indicate actual body content format for exchanged data.

This implementation will support the following Http verbs:

|  |  |
| --- | --- |
| **HTTP Verb** | **REST Operation** |
| GET | Retrieve the resource |
| POST | Create new resource on the server |
| PUT | Update an existing resource on the server (this can be an update of a few fields or a full replacement) |
| DELETE | Delete an existing resource on the server |
| HEAD | Retrieve header information (most recent change date/time) for a resource without retrieving the resource itself |
| OPTIONS | Communicate with the server about policies such as cross-domain policies. External facing façade (Mashery) should support options for enforcement by configuration. |

# REST Operations

## Retrieve Resource

The client sends the HTTP GET verb and a resource URI to a server and retrieves the representation of the resource in either JSON or XML format. The default resource format will be JSON.

The client can add the HTTP **Accept** header with one of the following values: “application/json,” “application/xml”. The default is “application/json”.

The server will respond with the resource representation in the response message body and HTTP Status Code 200 “OK” for a successful call (see Table 1 HTTP Status Codes).

The server will use the **Content-Type** header to communicate the format of the resource body in the response.

The response will contain an HTTP **Last-Modified** header with a timestamp of the resource’s last update.

If the request is recognized as incorrect for some reason, the server will return Http Status Code 400 “Bad Request” and an error message in the body. This will include unsupported content types in the Accept header.

If the request requires authorization, and the caller is not authorized, the server will return Http Status Code 401 “Unauthorized”.

If the resource is not found the server will return Http Status Code 404 “Not found” and an empty body.

If the request fails due to a system failure, the server will return Http Status Code 500 “Internal Server Error”.

In keeping with the intention of the REST standard, the GET operation will be idempotent and will not cause changes on the server.

## Create Resource

The client will send the HTTP POST verb and a “base resource URI” to the server. The client does not know the URI of the yet-to-be-created resource. Therefore, the client is sending the request to a URI which is used to create the type of resource – called here a “base resource URI”. For example, if the URI to a specific community is: [https://api2.compassion.com/ci/v1/field/communities/{community-id}](https://api2.compassion.com/ci/1/field/community/%7bcommunity-id%7d)   
the base URI for the resource is: <https://api2.compassion.com/ci/v1/field/communities>

The body of the request will contain the complete representation of the resource to be created.

The client will use **Content-Type** header to indicate whether the request body contains the resource as XML or JSON. The client will use the **Accept** header to indicate the desired format of any response body.

The server will respond with an empty message body and HTTP Status Code 201 “Created” for a successful call (see Table 1 HTTP Status Codes).

The response will contain a **Location** header which will contain the absolute URI of the newly created resource. For example: “https://api2.compassion.com/ci/v1/children/AA1234567”

In the event of an error, the server will use the **Content-Type** header to communicate the format of a message body in the response.

If the request is recognized as incorrect for some reason, the server will return Http Status Code 400 “Bad Request” and an error message in the body. This will include unsupported content types in the Accept header.

If the request requires authorization, and the caller is not authorized, the server will return Http Status Code 401 “Unauthorized”.

If the request fails due to a system failure, the server will return Http Status Code 500 “Internal Server Error”.

## Update Resource

The client will send the HTTP “PUT” verb and the specific resource URI to the server. The body of the request will contain the complete representation of the resource to be updated. The client will use **Content-Type** header to indicate whether the request body contains the resource as XML or JSON. The client will use the **Accept** header to indicate the desired format of any response body.

The server will respond with an empty message body and HTTP Status Code 204 “No Content” for a successful call (see Table 1 HTTP Status Codes).

In the event of an error, the server will use the **Content-Type** header to communicate the format of a message body in the response.

If the request is recognized as incorrect for some reason, the server will return Http Status Code 400 “Bad Request” and an error message in the body.

If the request requires authorization, and the caller is not authorized, the server will return Http Status Code 401 “Unauthorized”.

If the request fails due to a system failure, the server will return Http Status Code 500 “Internal Server Error”.

## Delete Resource

The client will send the HTTP “DELETE” verb and the specific resource URI to the server. The body of the request should be empty. The client will use the **Accept** header to indicate the desired format of any response body.

The server will respond with an empty message body and HTTP Status Code 204 “No Content” for a successful call (see Table 1 HTTP Status Codes).

In the event of an error, the server will use the **Content-Type** header to communicate the format of a message body in the response.

If the request is recognized as incorrect for some reason, the server will return Http Status Code 400 “Bad Request” and an error message in the body.

If the request requires authorization, and the caller is not authorized, the server will return Http Status Code 401 “Unauthorized”.

If the request fails due to a system failure, the server will return Http Status Code 500 “Internal Server Error”.

## Retrieve Header Data

The client will send the HTTP “HEAD” verb and the specific resource URI to the server. The body of the request should be empty. The client will use the **Accept** header to indicate the desired format of any response body.

The server will respond with an empty message body and HTTP Status Code 200 “OK” for a successful call (see Table 1 HTTP Status Codes).

The response will contain an HTTP **Last-Modified** header with a timestamp of the resource’s last update.

In the event of an error, the server will use the **Content-Type** header to communicate the format of a message body in the response.

If the request is recognized as incorrect for some reason, the server will return Http Status Code 400 “Bad Request” and an error message in the body.

If the request requires authorization, and the caller is not authorized, the server will return Http Status Code 401 “Unauthorized”.

If the request fails due to a system failure, the server will return Http Status Code 500 “Internal Server Error”.

# Uniform Resource Identifiers (URIs)

At their most basic level, REST APIs provide access to resources via URI paths. A resource is a chunk of related information, such as a user profile, a collection of updates, or a global user ID. Each resource is identified by one or more Uniform Resource Identifiers (URIs). To access the resource, an application calls a HTTPs operation (verb) on one of the resource's URIs. URIs will be represented as lower case by default.

## URI Syntax

The URIs for REST-based ESD Enterprise SOA services shall have the following syntax:

https://api2.compassion.com/{org}/{version}/{resource path}?{query string parameters}

The elements in this syntax are as follows:

* {org} - The root element should indicate the source organization for the resource.   
  For example: “ci” represents the GMC
* {version} - The version of the API. For all the URIs, the version number used currently is v1. This can be a single (“v1”)or multi-part number (“v1.1”). This represents versioning at the API level, not at the resource level.
* {resource path} - A path that defines a singleton resource or a collection of resources. Every {resource path} includes a “{domain}/{subdomain}/{plural-entityname}/{entity-id}/{qualifier}”
  + {domain} – this is a classifier for the resource. For example, if the resource is defined in Compassion International’s Global Program this may be “field” or “globalprogram” or “program”.
  + {subdomain} – this will be a modifier to indicate an area within the prior domain. For example, in the “field” domain, one may wish to identify a subdomain for Implementing Church Partners (icp).
  + {plural-entityname} – one may wish to identify resource representations of Countries, Communities, ChurchPartners or Children.
  + {entity-id} – used to represent a specific instance of an entity. Used to retrieve a specific instance (GET) of a resource, update (PUT) or delete (DELETE) an instance
  + {qualifier} - qualifiers can be added to further differentiate detailed requests. For example, when requesting detailed information about a child, a qualifier of “casestudy” may act as a suffix after the {entity-id} to further define the request.
  + A version can be used in the path at each level – this can be used to direct the call to a different endpoint as services are added or updated.
* {query string parameters} – these will be different for every call but may include filter criteria when retrieving collections or paging information in the future.

## URI Examples

[https://api2.compassion.com/ci/v1/children/{child-key}](https://api2.compassion.com/ci/v1/children/%7bchild-key%7d)

[https://api2.compassion.com/ci/v1/field/churchpartners/{churchpartner-id}](https://api2.compassion.com/ci/v1/field/churchpartners/%7bchurchpartner-id%7d)

Example to keep verb out of URI

# Resource Representations

Resource representations will be formed from canonical resources defined by the Enterprise Information Management group. This will ensure consistency among various services created at different times and hosted by different systems. These canonical types will be self-describing as to version so that it will be possible to determine whether a given type is compatible with prior client implementations.

Resource representations in REST services may also contain information other than the resource itself, such as hyperlinks.

Resources may contain a single entity, a homogeneous collection of entities or a composite of entities of various types. Single entities and their collections may be accessed from the same base URI. Composite entities should be treated as a completely new resource.

## XML Representation in REST

The XML representation of an entity will be defined by XML Schema and versioned by namespace in a way that is consistent with the current ESD Services Standards. These schemas will be checked into TFS source control in a location identified for service versioning.

Service messages which will need to include common Compassion types will reference these types from their common location. Common Compassion types will be checked into TFS source control in a location which is identified for Compassion common types.

## JSON Representation in REST

The primary goal of Compassion’s REST services is to be as natural as possible for a javascript developer in consuming the API. A second goal, one that is important in an enterprise, is that data structures are self-describing as to version. Thirdly, resource representations are different than the entity itself, in that they may contain hyperlinks which allow for application state change such as navigation to other important resources. In keeping with those goals, the following is offered for review:

**Single Entities**

The JSON representation of an entity will start with braces at the root node. It will also include optional version information in an attributes node and optional hypermedia links in a links node.

For example:

A child resource:

{

“ChildKey”: “BD5100171”,

“FirstName”: “Bireng”,

“LastName”: “Mru”,

<any other attributes or objects>,

“attributes:” {

“version”: “2011/05”,

“url”: “/ci/v1/children/BD5100171/information”

},

“links:” {

“image”: “/ci/v1/children/BD5100171/image”,

“casestudy”: “/ci/v1/children/BD5100171/casestudy”,

“localchurchpartner”: “/ci/v1/localchurchpartners/510”,

“country”: “/ci/v1/countries/iso/BD”

},

}

A Local Church Partner resource:

{

“LocalChurchPartnerId”: 510,

“OrganizationName”: “Local Church Partner Name”,

“StartDate”: “2006-11-01T00:00:00Z”,

“ChurchHasElectricity”: “T”,

<any other attributes or objects>,

“attributes:” {

“version”: “2011/05”,

“url”: /ci/v1/localchurchpartners/510

},

“links:” {

“image”: “/ci/v1/localchurchpartners/BD510/image”,

“community”: “/ci/v1/communities/510”,

“country”: “/ci/v1/countries/iso/BD”

},

}

**Special Values**

**Boolean**: values which values is a special problem for these representations since Compassion source data often represents these fields in different ways and may include more than two values.

Wherever possible, Boolean values will be represented as true Booleans in JSON. For example {“birthdateKnown”:true}

**Datetime**: Date and time values should be represented in a consistent way which handles dates and times in a global enterprise. As such, the standard for datetime values will be UTC format. For example: 2006-11-01T00:00:00Z.

Business requirements and source system restrictions may result in constraints on how Boolean, Datetime and other special values are returned. In some cases data may need to be return as string and interpreted by the client.

**Single Entity with Internal Arrays**

If a JSON node contains a collection of objects, it should define a plural array element (this can be a plural noun or the object type with “List” or “Collection” append) as shown in the following examples:

**Array Example One**

{

“attribute1”: “attribute data 1”,

“items”: [

{ “itemAttr1”: “attrValue”, “itemAttr2”: “attrValue”,… “version”: “optional version”},

{ “itemAttr1”: “attrValue”, “itemAttr2”: “attrValue”,… “version”: “optional version”},

{ “itemAttr1”: “attrValue”, “itemAttr2”: “attrValue”,… “version”: “optional version”},

{ “itemAttr1”: “attrValue”, “itemAttr2”: “attrValue”,… “version”: “optional version”}

]

“attribute2”: “attribute data 2”,

“attribute3”: “attribute data 3”,

<any other attributes or objects>,

“attributes”: “optional version information for the object”,

“links”: ”optional hypermedia links for the object”

}

**Array Example Two**

{

“attribute1”: “attribute data 1”,

“itemList”: [

{ “itemAttr1”: “attrValue”, “itemAttr2”: “attrValue”,…},

{ “itemAttr1”: “attrValue”, “itemAttr2”: “attrValue”,…},

{ “itemAttr1”: “attrValue”, “itemAttr2”: “attrValue”,…},

{ “itemAttr1”: “attrValue”, “itemAttr2”: “attrValue”,…}

]

“attribute2”: “attribute data 2”,

“attribute3”: “attribute data 3”,

<any other attributes or objects>,

“attributes”: “optional version information for the object”,

“links”: ”optional hypermedia links for the object”

}

**Multiple Resources Returned as Array**

If a JSON response contains an array of objects, these will be enclosed in brackets at the root:

A Local Church Partner array:

[

{

“LocalChurchPartnerId”: 510,

“OrganizationName”: “Local Church Partner Name”,

“StartDate”: “2006-11-01T00:00:00Z”,

“ChurchHasElectricity”: true,

<any other attributes or objects>,

“attributes:” {

“version”: “2011/05”,

“url”: “/ci/v1/localchurchpartners/510”

},

“links:” {

“image”: “/ci/v1/localchurchpartners/BD510/image”,

“community”: “/ci/v1/communities/510”,

“country”: “/ci/v1/countries/iso/BD”

}

},

{

“LocalChurchPartnerId”: 781,

“OrganizationName”: “Local Church Partner Name”,

“StartDate”: “2006-11-01T00:00:00Z”,

“ChurchHasElectricity”: true,

<any other attributes or objects>,

“attributes:” {

“version”: “2011/05”,

“url”: “/ci/v1/localchurchpartners/510”

},

“links:” {

“image”: “/ci/v1/localchurchpartners/BD510/image”,

“community”: “/ci/v1/communities/510”,

“country”: “/ci/v1/countries/iso/BD”

}

},

{

“LocalChurchPartnerId”: 9002,

“OrganizationName”: “Local Church Partner Name”,

“StartDate”: “2006-11-01T00:00:00Z”,

“ChurchHasElectricity”: “T”,

<any other attributes or objects>,

“attributes:” {

“version”: “2011/05”,

“url”: “/ci/v1/localchurchpartners/510”

},

“links:” {

“image”: “/ci/v1/localchurchpartners/BD510/image”,

“community”: “/ci/v1/communities/510”,

“country”: “/ci/v1/countries/iso/BD”

}

}

]

## Adding to Existing Namespaces

We can continue to add new types to the version at <yyyy>/<mm> until there is a breaking change to one of the existing published types. Then, we need to start a new version location, and create a new release of all types. Since changing types causes trouble for clients, we want to do this on a controlled and timed basis.

Breaking changes occur when objects or attributes are removed or re-defined. Simple additions of attributes do not constitute breaking changes.

# HTTP Status Codes

The following table shows how Compassion International REST services will behave with regard to HTTP Status.

Table 1 HTTP Status Codes

|  |  |  |
| --- | --- | --- |
| **Status Code** | **Reason Phrase** | **Usage/Scenarios** |
| 200 | OK | Used to indicate nonspecific success; it must not be used to communicate errors in the response body. |
| 201 | Created | Used to indicate successful resource creation. |
| 204 | No Content | Used when the response body is intentionally empty. For example, the response of PUT and DELETE. |
| 400 | Bad Request | Used to indicate nonspecific failure; used when no other 4xx error code is appropriate.  For errors in the 4xx category, the response body will contain a document describing the client’s error (unless the request method is HEAD). The error document will also contain an error ID which is useful for finding the logged exception/error messages on server side. |
| 401 | Unauthorized | Used when there’s a problem with the client’s credentials. It indicates that client has provided the wrong credentials or none at all. |
| 404 | Not Found | Used when a client’s URI cannot be mapped to a resource. |
| 500 | Internal Server Error | Used to indicate REST API malfunction. |

## Internal Error Response Mapping to HTTP Response Codes

Compassion’s internal Request/Reply services may be the service provider for REST APIs. Therefore the Business Error codes need to map back to standard HTTP Error Codes so that the HTTP Response message can be properly understood by the requesting consumer.

The table below dictates the mappings:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Business Error Name | Error Code | HTTP Status Code | HTTP Error Message | Additional Info on the Error |
| System Error | 4000 | HTTP 500 | Internal Server Error | An unexpected error has occurred while processing this request |
| Authentication Error | 4001 | HTTP 401 | Unauthorized | Authentication has failed or credentials haven’t been supplied |
| Authorization Error | 4002 | HTTP 401 | Unauthorized | Authentication has failed or credentials haven’t been supplied |
| Validation Error | 4004 | HTTP 400 | Bad Request | One or more parameters have failed validation. Verify the parameters according to the software specification. |
| Timeout Error | 4005 | HTTP 500 | Request Timeout | The downstream service provider did not produce a response within the time that the service was prepared to wait. |
| Communication Error | 4006 | HTTP 500 | Internal Server Error | An unexpected internal error has occurred while processing this request |
| [Item] Not Found Error | 4009 | HTTP 404 | [Item] Not Found. | A retrieval based upon [Item] selection criteria returned no results. This could be an image, a sponsor id, a sponsor email address, etc. |
| Service Not Available | 4009 | HTTP 500 | Service Not Available | Service unavailable: [Service Name] |
| Message Router Error | 4011 | HTTP 400 | Bad Request | There were errors while trying to route the message. |
| Message Process Error | 4012 | HTTP 500 | Processing Error | There were errors while trying to process the message. |
| Service Provider Error | 4013 | HTTP 500 | Bad Request | There were errors while trying to process the message. |
| Exception Rule Processing Error | 4014 | HTTP 500 | Internal Server Error | An internal error has occurred attempting to process message. |

# HTTP Headers

|  |  |  |
| --- | --- | --- |
| **Header** | **Values** | **Notes** |
| Accept | “application/json” or “application/xml”  Default: “application/json” | Used by the client to indicate the desired mime type of the message body to be returned by the server. |
| Content-Type | “application/json” or “application/xml”  Default: “application/json” | Used by sender (either client or server) to indicate the mime type of the message body. |
| Last-Modified | Timestamp in the format: “Tue, 15 Apr 2014 08:12:31 GMT”  There is no default value. | Filled by the server when returning a resource and when this information is available. If not available, the entry is empty. This is dependent upon the availability of the data from the source system. |
| Location | Absolute URI to the resource at this server. For example: “https://api2.compassion.com/ci/v1/children/AA1231234”  There is no default value. | Filled by the server when a new resource is created. Contains the Compassion External Façade URL at which to access this resource in the future (Mashery URL). |

# Outstanding Questions (Architect and Tech Lead discussion with answers below)

1. What is the most effective way to support resource versioning?   
   Resource versioning occurs when the content or structure of the resource is changed.

**Answer:** Changing the public locations of resources will present customers with difficulties. Therefore, resource versioning should not require a new URI. Resource versioning would be internal to the response message – for example, in the “attributes” node.

Changing the version in the uri from /v1/ to /v2/ would accompany a whole new version of the API. This should be an infrequent occurrence, and does not necessarily mean that all resources have changed.

1. Do we want to support retrieval of any additional information such as api documentation through the OPTIONS command?

**Answer:** This item will be addressed in a future release if needed.

1. Do we want to support conditional requests using Last-Modified and ETag response headers? From *RESTful Webservice Cookbook*:

“Servers use Last-Modified and ETag response headers to drive conditional requests.

Clients use the following request headers to make requests conditional:

If-Modified-Since and If-None-Match for validating cached representations.

If-Unmodified-Since and If-Match as preconditions for concurrency control.

The efficacy of processing conditional requests depends on how quickly the server can validate the Last-Modified and ETag headers. ”

**Answer:** The topics of caching and concurrency control will be addressed in a future release if needed.

1. Singular or Plural for base URIs?
   1. For example:
      1. /ci/v1/children/BD5100171 or /ci/v1/children/BD5100171?
      2. /ci/v1/countries/BD or /ci/v1/countries/BD?
      3. /ci/v1/communities/451 or /ci/v1/communities/451?
      4. /ci/v1/localchurchparter/510 or /ci/v1/localchurchpartners/510?

**Answer:** We will use the convention of plural base URIs. Accordingly, all examples in this document should demonstrate this convention.

1. Where best to “clean up” the data – if at all?
   1. In relation to “magic strings” comment.   
      Example #1: ProgramStatus=”A” – should this be changed to “Active” in the MIL? Example #2: HIVCategory = “AFFCTD” or “NOTAFF” – should these be changed to be user understandable so that the client doesn’t have to do this?
   2. In relation to date/time: StartDate: "2012-09-20T15:48:55.71-06:00" – should this be changed to just return a Date and no time even though the source has time?
   3. Data interpretation - ProgramImplementor has ElementaryCompletionStatus = “1”

**Answer:** This kind of cleanup should be pushed as close to the source system as possible. For example, if this cannot be done in the source database, then we will prefer doing the cleanup in the MIL. If it cannot be done in the downstream system then it will be done in the ESB.

1. Element naming – if the schema is based upon a canonical logical model (i.e. a Child resource has a ChildKey attribute) should we remove the resource type “Child” from all attributes “ChildKey” when we create the physical implementation? The goal of the comment was to avoid Child.ChildKey.

**Answer:** Element naming will be driven by the canonical model. However, we will seek to make it as intuitive as possible for consumers. Therefore, we will remove the entity name where it makes the contract more intuitive and understandable for consumers.

1. Schema element casing - Should our REST services whether returning XML or JSON adopt a revised casing model with an associated change to the XSDs? For example, if the entity is Child and the entity has an attribute ChildPersonalName, should we return lower case “child” and camel case “childPersonalName” in both XML and JSON responses?

**Answer:** we will keep pascal casing for xml and camel case for JSON.

1. Schema element typing - Should our REST services whether returning XML or JSON adopt a revised data type model with an associated change to the XSDs? For example, if the entity is ProgramImplementor and the entity has an attribute MonthlyVolunteerHours, should we limit the XSD to an Integer type regardless of what is in the source system? (We could decide to only do this where the source type is an integer…) [see “f” above]
2. ID inclusion – Should our REST services return the ID from the URI in the data? For example, if we retrieve a Child resource with /ci/v1/children/BD5100171/information, should the response payload include “ChildKey”: “BD5100171”?

**Answer:** We will continue to return Identifiers in the response.

1. How intuitive is the interface represented by the following URIs? Governance will be crucial to ongoing success with this.   
   1. Child API
      1. https://api2.compassion.com/iptest/ci/v1/children/BD5100171/information
      2. https://api2.compassion.com/iptest/ci/v1/children/BD5100171/casestudy
      3. https://api2.compassion.com/iptest/ci/v1/children/BD5100171/exitdetails
      4. https://api2.compassion.com/iptest/ci/v1/children/BD5100171/image
   2. Field Partner Churches API  
      Comments have indicated that the similarities between some of these are confusing.
      1. https://api2.compassion.com/iptest/ci/v1/cdspimplementors/BD510
      2. https://api2.compassion.com/iptest/ci/v1/cdspimplementors/BD510/agegroups
      3. https://api2.compassion.com/iptest/ci/v1/cspimplementors/BD510
      4. https://api2.compassion.com/iptest/ci/v1/programimplementors/BD510
      5. https://api2.compassion.com/iptest/ci/v1/localchurchpartners/8227
      6. https://api2.compassion.com/iptest/ci/v1/field/churches/8227/cdsp/BD510
   3. Country API
      1. https://api2.compassion.com/iptest/ci/v1/countries/50
      2. https://api2.compassion.com/iptest/ci/v1/countries/iso/BD
      3. https://api2.compassion.com/iptest/ci/v1/countries/legacy/BD
   4. Community API
      1. https://api2.compassion.com/iptest/ci/v1/communities/8015

**Items for future follow-up:**

1. Providing API documentation through the OPTIONS command.
2. Data cleansing should be done in the MIL layer if possible.
3. With regard to date and time, more requirement and specification work should be done to ensure that Compassion can truly support a global enterprise with international dates and times.
4. Support for caching should be considered as part of the standard.
5. Support for concurrency should be considered, especially prior to the development of write-oriented services. At the time of this writing, the services being developed are read-only.
6. A governance process should provide oversight on the development of intuitive URIs and resources. Customer input would be extremely valuable in this.
7. The treatment of non-XML and JSON media types (such as images) should be considered.
8. Security standards to be used with REST should be considered.