Data Format Specification Standards

Version 1.0 24 March 2014

Introduction

It is very important that the data formats exchanged between components in the Smarter Balanced assessment delivery system be well-designed and well-documented. Each format governs multiple components, and these formats will have greater longevity than their corresponding source code. Accordingly, Smarter Balanced has defined the following standards for data format specifications.

These standards are to be used when designing data formats, when documenting them, and when reviewing them. Any variation from these standards will require explanation in the data format documentation, and that variation will have to be approved as part of the data format review process.

These standards are designed according to the Four-Layer Framework for Data Standards. Accordingly, they are divided into the following sections:

- Data Dictionary
- Data Model and Serialization
- Data Exchange Protocol

The key words "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. All Smarter Balanced Data Format Specifications SHOULD use the same convention.

Format documents MUST include version numbers, approval status, and corresponding dates. Versions SHOULD include major and minor version numbers delimited by a decimal point (e.g. 3.4). The Data Dictionary, Data Model, and Serialization SHOULD all share the same version number. Data Exchange Protocols SHOULD share the same major version number with the corresponding Dictionary, Model, and Serialization, but minor version numbers MAY be incremented independently. Version numbers SHOULD NOT be updated unless material changes have been made to the specification.

While it is required to include references to existing standards from which elements are drawn, the specification document is the authoritative source. Therefore, it MUST include all information necessary to implement the data format or protocol, and if there is any disagreement between the specification and a referenced source, the specification, not the referenced document, SHOULD be followed. A statement to this effect SHOULD be included at the beginning of the specification.

Data Dictionary

Every data format specification MUST have a Data Dictionary as the first major section. The Data Dictionary MUST declare every element, attribute, or field in the data format. With the exception of the section on XML, this document uses “element” as a generic term to include elements, attributes, and fields.

AIF, CEDS and Other Existing Data Standards

Element names and definitions SHOULD be drawn from the following sources, *listed below in order of preference*. It should be rare to encounter an element that is not already defined by one of these:

- Assessment Interoperability Framework (AIF): https://ceds.ed.gov/aif.aspx (In most cases, AIF standards have been incorporated into CEDS. Accordingly, the element references will be to CEDS rather than AIF.)
- Common Education Data Standards (CEDS): https://ceds.ed.gov/
- Learning Resource Metadata Initiative (LRMI): http://www.lrmi.net/
- Schema.org: http://schema.org
- Learning Object Metadata: http://ltsc.ieee.org/wg12/
- Dublin Core: http://dublincore.org/

Reference

The Data Dictionary MUST include a "Reference" column that SHOULD contain a reference to the origin of the element definition, whether from one of the above standards or from elsewhere. The reference SHOULD include the version number of the referenced standard and a URL to the element definition. In cases where the existing standard doesn't offer URLs directly to the element level, the reference SHOULD be a URL to the overall standard, with additional information on how to find the actual element. (e.g. http://somestandard.org/document.pdf, page: 3, heading: SomeElement). The reference column SHOULD also describe any variation from the referenced standard, such as name or field length.

Element Names

Element names SHOULD meet the following rules. When rules conflict, the first of the conflicting rules, as listed below, takes precedence.

1. Names MUST meet the requirements of the chosen data format (e.g. XML, CSV)
2. Names MUST be unique within a particular Data Dictionary.
3. Names SHOULD start with a letter and MUST be composed strictly of letters, digits, and the underscore "_" character.
4. Names SHOULD be the same as the name from the specified data standard.
    - For CEDS, this means that the "Element Technical Name" should be used.
    - When the name is contextual, then the name should be prefixed with the appropriate context and delimited with an underscore. For example, in CEDS the element for a school district name (Local Education Authority or LEA name) is "OrganizationName". However, this element is also used for the names of other organizations such as the State Education Authority (SEA) and the School (Facility). So, the element name for the name of a School District SHOULD be "LEA_OrganizationName".
5. XML attributes SHOULD be avoided (see the XML subsection under "Data Model and Serialization") When they are used, attribute names SHOULD be all lower case.
6. Names of all other elements SHOULD use upper camel case (e.g., "UpperCamelCase").

Simple Elements

Simple elements are those that contain data but no other elements. Simple elements MUST be fully *defined* in the Data Dictionary. At a minimum, each simple element definition MUST include the following components:

- **Name**
- **Description:** MUST unambiguously describe the meaning of the element
- **Acceptable Values:** Including definitions of acceptable values
- **Data Standard Reference:** A reference to a source element in a data standard from which the element was adopted and a URL whenever available. SHOULD also include a description of any variation from the referenced element.

Description, acceptable values, and definitions of the acceptable values SHOULD be written in plain English that is accessible to moderately technical readers.

Complex Elements

Complex elements are those that contain other elements. While complex elements MUST be *declared* in the Data Dictionary, a full and unambiguous definition may require a combination of the Data Dictionary and Data Model and Serialization entries. This is because information about the relationships between elements (those containing or contained by other elements) MAY be deferred to the Data Model section of the specification. In the Data Dictionary each complex element *declaration* MUST include the following components:

- Name
- **Description:** MUST unambiguously describe the meaning of the element
- **Acceptable Values:** Including definitions of acceptable values, as applicable.
- **Data Standard Reference:** A reference to a source element in a data standard from which the element was adopted and a URL whenever available. SHOULD also include a description of any variation from the referenced element.

Under most circumstances, complex elements SHOULD NOT contain any data that is not part of a sub-element. Description and acceptable values SHOULD be written in plain English that is accessible to moderately technical readers.

When specifying acceptable values, data dictionaries MAY use references to W3C XML Schema datatypes using the "xsd:" prefix. For example, "xsd:normalizedString".

Note that the use of the "xsd:" prefix to reference the "http://www.w3.org/2001/XMLSchema" namespace in data dictionaries is by convention only. Within the Data Model schema files, any prefix MAY be used so long as it is declared properly according to XML syntax.

Universally Unique Identifiers (UUIDs)

The terms, "Universally Unique Identifier", "Globally Unique Identifier", and their acronyms, UUID, and "GUID" are synonymous and MUST only be used to refer to 128-bit unique identifiers generated according to the algorithms and encoding standards of RFC 4122. Functions for properly generating and encoding GUIDs are built into most contemporary programming environments.

When included in a Smarter Balanced data format, UUIDs MUST be encoded in 32-character hexadecimal form using lower-case letters, with dashes after the 8th, 12th, 16th, and 20th digit, with no prefix or brackets (e.g., "f81d4fae-7dec-11d0-a765-00a0c91e6bf6"). This is according to section 3 of RFC 4122 minus the "urn:uuid:" prefix. The order of the hexadecimal digits and mapping to the 128-bit binary form MUST be in accordance with RFC 4122.

Smarter Balanced systems that receive UUIDs SHOULD tolerate the absence of dashes and the use of upper-case hexadecimal digits, and they SHOULD regard GUIDs as being equivalent when the difference is limited to formatting differences such as dashes or case. This MAY be accomplished either by using the binary encoding internal to a component, or by normalizing the format of the GUID upon receipt.

Other Identifiers

Non-GUID identifiers SHOULD be constrained according to the xsd:token datatype, in that they have no leading or trailing spaces and internal spaces are condensed.

The definition of the identifier element MUST describe the scheme for the IDs, how they are generated, and the domain in which they must be unique. It MUST describe whether identifiers are

case-sensitive or case-insensitive. It MUST also describe any identifier normalization that is required before comparing identifiers. For example, case-insensitive identifiers MAY be normalized to upper-case or lower-case; which option to use MUST be specified.

While CEDS uses the xsd:normalizedString for most identifiers, Smarter Balanced uses the more-constrained xsd:token datatype. Also, CEDS typically includes a maximum length of 30 characters. However, string-encoded GUIDs, including dashes, are 36-characters in length. Some states may also use a secure hash function such as HMAC-MD5 or HMAC-SHA1 to generate identifiers. These algorithms result in 32 and 40 hexadecimal digit strings respectively. Therefore, identifiers SHOULD have a max length of at least 40 characters.

Sample Data Dictionary

The following sample Data Dictionary is not from any actual Smarter Balanced specification. It is intended to demonstrate *one way* of organizing the information required by these standards. The layout is not prescriptive. Specifications should use the format that is best suited to each application.

Name	Description	Acceptable Values	Reference
StateOfResidence	A person's permanent address as determined by such evidence as a driver's license or voter registration. For entering freshmen, state of residence may be the legal state of residence of a parent or guardian.	AK, AL, AR, AS, AZ, CA, CO, CT, DC, DE FL, FM, GA, GU, HI, IA, ID, IL, IN, KS, KY, LA, MA, MD, ME, MH, MI, MN, MO, MP, MS, MT, NC, ND, NE, NH, NJ, NM, NV, NY, OH, OK, PA, PR, PW, RI, SC, SD, TN, TX, UT, VA, VI, VT, WA, WI, WV, WY (See the CEDS reference for meanings of these values.)	https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=3268 CEDS v3.0
LEA_OrganizationName	The name of the Local Education Authority, typically a School District.	xsd:token MaxLength: 60	https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=3204 CEDS v3.0
LocalEducationAgencyIdentifier	A unique number or alphanumeric code assigned to a local education agency by a school system, a state, or other agency or entity. Must be unique within the state.	xsd:token MaxLength: 40 Comparisons are case-insensitive. Identifiers are normalized to all upper-case.	https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=3153 CEDS v3.0 Datatype is “xsd:token” instead of “xsd:normalizedString” and MaxLength is extended to 40.

educationalAlignment	An alignment to an established educational framework.	Complex element of type AlignmentObject *(Note that a complete Data Dictionary would have to include all subelements included in AlignmentObject)*	http://schema.org/educationalAlignment Schema.org v1.0e (part of the LRMI additions to Schema.org)
AssessmentItemDifficulty	The percentage of students who answered the item correctly during trial testing of the item.	xsd:decimal MinValue: 0.00 MaxValue: 100.00 Format: ###.##	https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=3383 CEDS v3.0
AssessmentItemGUID	A GUID to uniquely identify this assessment item.	GUID Format: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx	https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=3623 CEDS v3.0 Element name changed from AssessmentItemIdentifier to AssessmentItemGUID. MaxLength extended to 36 in order to accommodate GUIDs

Data Model and Serialization

Metadata formats typically define an Abstract Data Model that may be encoded, or serialized, in one of several ways. For example, the Smarter Balanced Assessment Item Metadata format defines a set of names and values that may be encoded differently depending on where the corresponding assessment item is stored and how it is encoded.

Most other Smarter Balanced data formats are concrete in that the specific serialization must be defined down to the binary layer. The Data Model section MUST express whether the model is abstract or concrete.

Abstract Data Models

An abstract data model that is flat (i.e., does not have any complex elements, hierarchy, or relationships between entities) may simply be an augmented Data Dictionary. The additional information required in the Data Dictionary SHOULD include the following:

- The "Acceptable Values" column is required and MUST be unambiguous about the format of the values. For example, a Boolean value might be encoded as xsd:token with acceptable values of "Yes" and "No", or it might be encoded as xsd:boolean.
- Whether an element is required or optional.
    - If an element is optional, the default value or behavior MUST be described.
- Whether an element may have multiple values or only one value.
- Whether the values of a multi-valued element are ordered, and if so, how to indicate the order of the values.

Abstract Data Models with additional structure, such as hierarchy or complex objects, MUST have additional detail about the data structure. In such cases, the Data Model SHOULD follow the documentation pattern established by Schema.org, and SHOULD use plain English descriptions that are accessible to moderately technical readers, as much as possible.

Concrete Data Models

Concrete Data Models MUST use an existing serialization format. CSV and XML are the preferred formats.

CSV

Data Models that are flat or nearly flat SHOULD use CSV format. Data models that use CSV format MUST follow the version specified in RFC 4180 and use UTF-8 character encoding. These facts (use of RFC 4180 and UTF-8) MUST be included in the data format specification. The order of fields (columns) in the CSV file MUST be defined by the data format.

Components that generate CSV files MUST include one header line at the beginning of the file, and must export fields in the order specified by the data format.

Components that accept CSV files SHOULD detect and tolerate the absence of a header line, and SHOULD accept fields in the order that the header specifies regardless of whether it is the same as in the specification.

CSV-based specifications MUST include a short sample of simulated data.

XML

Data models that have some degree of structure SHOULD use the XML format. This may be the case even when most of the data is flat but there is a structured header to give context. It is acknowledged that JSON may be more efficient in many cases, but XML support is, at present, more widespread in programming environments, and it benefits from better validation tools.

An XML-based specification MUST include a formal schema definition, and SHOULD use W3C XML Schema (XSD) for that definition. It MUST also include a more reader-friendly form. As with abstract data models, the documentation pattern established by Schema.org is recommended with the addition that element descriptions SHOULD include sample XML fragments.

XML namespaces MUST be used. The data format MUST have its own namespace. The namespace SHOULD be rooted in http://www.smarterapp.org/{version}/" (e.g., http://www.smarterapp.org/3.5/MetaWidget). The data format MUST properly reference namespaces for any elements imported from other XML formats. Elements adopted from other standards (e.g., CEDS), MAY be redefined in the new format, especially when there are differences such as the changes to datatype and minimum length required for identifiers.

XML elements are preferred over XML attributes. Attributes SHOULD NOT be used unless they are imported from another schema. Doing so facilitates automated translation into other formats that do not have attributes (e.g., JSON).

Data Exchange Protocol

The Data Exchange Protocol MUST be specified separately from the Data Format. When part of the same document, the Data Exchange Protocol and the Data Format MUST be in separate sections. All transfers, regardless of whether student data is included, MUST use standard channel encryption (i.e., HTTPS, FTPS, SFTP).

APIs and protocols SHOULD use RESTful design patterns.

Data Transfer Protocols

All data transfers that originate with the recipient, whether large or small, SHOULD use the HTTPS protocol.

When responding to requests for potentially large files, HTTP servers MUST efficiently support client requests to segment the file into multiple parts and/or restart a broken transfer. This means that the server MUST assume that the client will make multiple "GET" or "POST" requests, each with a "Range" header that indicates a portion of the whole file. For example, the client may request the transfer of a 100MB file in 50 2MB chunks, each using a "Range" header to specify the portion being requested. When responding to such a series of requests, the server SHOULD respond with segments of a cached copy of the data. The server SHOULD NOT re-generate the data in response to each request for a part of the response.

Small data transfers that originate with the transmitter SHOULD use the HTTPS protocol.

Large data transfers that originate with the transmitter MAY use FTPS or SFTP. However, it is RECOMMENDED that such transfers use the HTTPS protocol, and that they utilize the "Content-Range" header to transmit a large file in multiple small segments. This will result in similar reliability to the use of FTP family protocols, while facilitating the use of the same RESTful protocol designs and Single Sign On security that are used in the balance of the API.

Authentication and Authorization

The same SAML-Based Single Sign On and Permissions infrastructure that is used to authenticate and authorize users for browser-to-server communications SHOULD also be used to authenticate and authorize systems for server-to-server communications. Under certain circumstances, this may necessitate creating accounts within the SSO system for services as opposed to users. For example, an account might be created for a Test Delivery Instance, and that account would be authorized to transmit data about a particular state into the data warehouse.

Atomic Transfers

All transmissions SHOULD be atomic in that all data that is associated by dependencies SHOULD be combined into a single document transmitted in a single transaction. For large documents, that transmission MAY be divided into multiple parts as described above in the "Data Transfer Protocols" section. Document validity SHOULD be verified and the data MUST be stored in a reliable datastore before receipt is acknowledged. In this way, the sender of the data can safely dispose of data once receipt has been acknowledged.

Atomic transfers of this sort simplify protocols by avoiding asynchronous acknowledgements, distributed transaction coordination, and related complexities. When a large number of documents need to be transferred, the synchronous nature of such protocols can slow things down, since the sender has to wait for an acknowledgement following transmission. Such overhead can be minimized by using parallel processes to transmit multiple documents at the same time.

Repeat Transmissions

If the same data is transmitted multiple times, the result SHOULD be equivalent to sending the data one time. This is in accordance to REST design principles. When data is intended to be repeated in a dataset (e.g. the same student took the same test multiple times) then there SHOULD be some value in the document that distinguishes between the multiple instances so that the latter instance doesn't replace the former.