Overview

1 Comment

RAPID-ML™, the Modeling Language for Resource API over Data, is a structured language for comprehensively describing and documenting data-oriented, REST-style APIs. RAPID-ML can be used by generators or runtime interpreters for API documentation, visualization, data schemas, test frameworks, client SDKs and service implementations.

RAPID-ML is designed for general-purpose API modeling, emphasizing a few areas of particular importance:

- RAPID-ML enables the definition of shared, canonical data models, and the adaptation of these shared models to meet the needs of a particular API usage context. RAPID-ML uses the term *realization* to describe a data type adapted for use in a given resource or message. Realizations vary from their underlying data types only in well-defined ways that do not break conformance with the underlying data model. This constrained variation promotes interoperability by making it practical for APIs to converse in a common, structured data vocabulary.
- RAPID-ML includes its own technology-neutral data modeling sub-language. This allows straightforward description of Data Model semantics that are meaningful in APIs, without syntactic noise or unnecessary complexity associated with specific message formats. RAPID-ML leaves most of the wire format details to be handled by message schema generators.
- RAPID-ML supports formal binding of data models into the API. Resources may be bound to data structures. URI and message parameters may be bound to data properties. Hyperlinking and embedding may be defined by reference properties in the underlying data structures. These data binding semantics capture common REST idioms in a concise, natural way, and enable more complete downstream implementation of the API's intended behavior.
- RAPID-ML uses a highly readable, indent-based syntax that features *optional fluency*. The language syntax includes optional *fluency keywords* that have no semantic effect, but serve to make the syntax more natural. By including or omitting these optional keywords, developers can adopt a fluent or terse coding style.

This specification provides all the information necessary to describe resource APIs and their associated data types in RAPID-ML

1 Comment

Language Syntax

0 Comments

Language

RAPID-ML uses an indent-based syntax where leading tab characters denote block scope, similar to Python or YAML. A RAPID Model can Import Data Models, Resource API models and Definition Libraries from other RAPID Models.

RAPID-ML also includes documentation comments, which can be included in generated documentation formats.

Syntax Notation

The following table summarizes the syntax *notation* used in this specification.

- Variables are enclosed in '<' and '>' symbols and are described in the topic parameters table.
- Variable keywords are described in the topic parameters table.
- Variable qualifier keywords are described in the topic parameters table.
- Optional fluency keywords, which are there to help readability, are preceded by a '~' symbol and are not described in the topic parameters table.
- Fixed keywords, which don't have any special meaning other than to delineate relevant parts of the statement, are also not described in the topic parameters table.

Form	Grouped	Optional	Repeating	Example
symbol	No	No	No	rapidModel
[symbol] or [~symbol]	No	Yes	No	<property-name> : [~as] [containing] reference [~to] <structure-name>
symbol...	No	No	Yes	<response>...
(group of symbols | group of symbols)	Yes	No	No	enum (int | string) <enum-name>
[group of symbols]	Yes	Yes	No	[<namespace-declaration>]
[group of symbols]...	Yes	Yes	Yes	[<property-name> : <property-type>]...
(group of symbols)...	Yes	No	Yes	(<built-in-media-type> | <user-defined-media-type>)...

Delimited Lists

Lists MAY be delimited either with commas or newlines. For example, in a Property Set we can do this:

```
with all properties including
   firstName!, lastName!
excluding
   addresses
```

Or we can do this:

```
with all properties including
   firstName!
   lastName!
excluding
   addresses
```

Lists having nested structure MUST use the newline-delimited form:

```
with all properties including
   id! // Add cardinality override
   ssn // Add regex contraint
      matching regex '(?!000|666)[0-8][0-9]{2}-(?!00)[0-9]{2}-(?!0000)[0-9]{4}'
```

Conventions

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 [RFC2119].

0 Comments

Model Element References

0 Comments

Defined model elements such as Data Structures, Resource Definitions and Link Relations, are frequently referenced from other elements. For example, a Data Structure may refer to an Enumeration as a Primitive Property type and a Resource Definition refers to its bound Data Structure. Such references have various different forms, rules and guidelines which are detailed below.

Element Names (Name)

The term 'Name' as used in this specification refers to an atomic model element name. Names MUST consist of a combination of upper or lower case letters, numbers and underscore characters matching the following regular expression:

^([a-z]|[A-Z]|[_])([a-z]|[A-Z]|[_]|[0-9])*

Names are used in a variety of contexts where they may or may not be qualified.

Qualified Element Reference (QName)

The term 'QName' as used in this specification is short for 'qualified Name' and means an appropriately qualified *or* unqualified Name, according to the rules for model element references documented below. This usage is consistent with the XML usage of QName. (XML QNames can be Namespace-prefixed or not; 'QName' just means that the Name MAY be qualified.)

Scoped Element Reference (SName)

The term 'SName' as used in this specification is short for 'scoped Name' and denotes a Name that is *never* qualified. For example, `includedProperties` listed in a Reference Link or Reference Embed; or scopes listed in a secured element. Such model element references are never qualified because they occur in the scope of a referenced element, so there is no need to qualify them further.

Forms of Reference

Model element references use different forms, depending on the circumstances.

Form	Description	Syntax	Examples
Unqualified Name	The model element is referenced by its Name, without any qualification.	<element-name>	`Replenishment`
Model-Qualified Name	The model element is referenced by both its RAPID Model Name and its own	<containing-model>.<element-name>	`InventoryData.Replenishment`

	Name. Only available for elements whose containing model is defined locally, within the same RAPID-model as the referrer.		
Alias-Qualified Name	The model element is referenced by an alias Name and its own Name. Only available when the element's containing model is imported with an alias.	<alias>. <element-name>	`inv.Replenishment`

Fully-Qualified Name	The model element is referenced by its RAPID Model Name, its containing model Name and its Name. Note that if the referenced model element is defined in a RAPID Model that specifies a Namespace, the Namespace Name MUST be included in the fully-qualified element reference,	[<namespace>.] <rapid-model>. <containing-model>. <element-name>	With Namespace:
			`megafacture.mrp.MRPModel.InventoryData`
			Without Namespace:
			`MRPModel.InventoryData.Replenishment`

	as a prefix to the RAPID Model Name.		

Allowed and Preferred Reference Forms

The following table summarizes the allowed and preferred model element reference forms, according to the the circumstances.

- Conformant RAPID-ML editors and processors MUST recognize all allowed forms.
- Conformant RAPID-ML editors and validators MUST raise an error condition wherever a model element reference appears in a disallowed form.
- Conformant RAPID-ML editors and generators SHOULD suggest the preferred form, as specified here.

Element Origin	Has Alias	Name Clash	Unqualified Form	Model-Qualified Form	Alias-Qualified Form	Fully Qualified Form
Local	N/A	No	**Preferred**	Allowed	N/A	Allowed
Local	N/A	Yes	Disallowed	Allowed	N/A	**Preferred**
Imported	No	No	**Preferred**	N/A	N/A	Allowed
Imported	No	Yes	Disallowed	N/A	N/A	**Preferred**
Imported	Yes	No	**Preferred**	N/A	Allowed	Allowed
Imported	Yes	Yes	Disallowed	N/A	**Preferred**	Allowed

Terminology

Accessible

An element is *accessible* or *in scope* if it is defined in the same RAPID Model, or is defined in a sub-model that is explicitly imported into the current RAPID Model.

RAPID Model

For this specification, *RAPID Model* refers to a top-level RAPID-ML model, corresponding to an individual file and containing one or more Data Models, Resource APIs, or Definition Libraries.

RAPID-ML Specification

RAPID-ML Specification refers to this document.

Realization

Realizations are adaptations of canonical Data Structures in order to tailor them to a specific application context. The realization is described by applying formally defined operations that maintain certain guarantees of conformance to the underlying data type. Realization operations include:

- Property Set, which specifies inclusion or exclusion of certain properties;
- Reference Treatments, which represent reference properties as hyperlinks or embedded data; and
- Constraints, which further restrict the value space or Cardinality of data properties in the context of a resource or Method.

REST

REST is used in the context of an API implemented using the principles of REST. The REST acronym stands for Representational State Transfer and was first introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.

RAPID-ML supports, but does not enforce, fully RESTful APIs. API designers are free to design APIs with appropriate tradeoffs among various architectural properties such as loose coupling, type safety, and ease of use. Designers can also choose among various ways to formalize the API contract, using media types and link relations as prescribed by REST, or using less formal out-of-band information.

Revision History

0 Comments

Version	Date	Change Summary
1.0	2015-07-15	Initial Release of RAPID-ML Specification.

0 Comments

RAPID Model File Format

0 Comments

RAPID Model files MAY contain up to three kinds of top-level elements: a Namespace declaration, Import statements and a RAPID Model in this order.

Syntax

[<namespace-declaration>]
[<import-statement>]...
<rapid-model>

Examples

```
rapidModel TaxBlaster
```

```
namespace taxmasters.api
rapidModel TaxBlaster
```

```
namespace taxmasters.api
import taxmasters.data.Taxation.GeneralTypes from
"http://data.taxmasters.com/DataModels.rapid" as data
rapidModel TaxBlaster
```

Parameters

None.

Child Elements

Name	Topic	Description
<namespace-declaration>	Namespace	The Namespace in which this RAPID Model resides. Must be on first line of the file.
<import-statement>	Import	Imports a child element from another RAPID Model for usage in this one.
<rapid-model>	RAPID Model	Top-level container in a RAPID-ML document.

Parent Elements

None.

Discussion

By convention, RAPID model files are named with a .rapid extension . Tools SHOULD encourage use of the .rapid naming convention, but MAY save and recognize RAPID models not conforming to this naming convention.

0 Comments

Namespace

0 Comments

The `namespace` element, if used, MUST be on the first line of the RAPID-ML™ document. Namespaces are used to create fully-qualified, unique identifiers for RAPID Model child elements. This may be necessary because RAPID-ML allows these elements to be imported.

Syntax

namespace <name-segment>[.<name-segment>]...

Examples

```
namespace taxmasters.data
```

```
namespace taxmasters.api
```

Parameters

Name	Type	Description
<name-segment>	Name	Namespaces MUST be comprised of one or more '.' separated Names.

Child Elements

None.

Parent Elements

None.

Discussion

The assigned Namespace distinguishes models that may have the same names as other imported or locally defined models. In these cases, model imports and model element references can be disambiguated by using the *fully-qualified name*, which includes the Namespace.

See Import and Model Element References for more information.

0 Comments

Import

0 Comments

The `import` element is used to import RAPID Model child elements from other RAPID Models for usage in the scope of this RAPID Model.

Syntax

import [<namespace-name>.]<rapid-model-name>.(<data-model-name> | <resource-api-name> | <definition-library-name>) from "<import-uri>" [as <alias>]

Examples

```
import taxmasters.data.Taxation.GeneralTypes from "DataModels.rapid"
```

```
import taxmasters.data.Taxation.GeneralTypes from
"http://data.taxmasters.com/DataModels.rapid" as data
```

```
import taxmasters.api.Taxation.GeneralTypes from
"http://data.taxmasters.com/InterfaceModels.rapid" as api
```

Parameters

Name	Type	Description
<namespace-name>	SName(Namespace)	The Namespace identifier of the model to be imported. The <namespace-name> MUST be specified and must match the namespace declared in the imported RAPID Model. If the imported RAPID Model does not declare a namespace, the Import statement MUST NOT specify a namespace.
<rapid-model-name>	SName(RAPID Model)	The name of the RAPID Model from which a child element will be imported.
<data-model-name>	SName(Data Model)	The name of the Data Model to be imported.
<resource-api-name>	SName(Resource API)	The name of the Resource API model to be imported.
<definition-library-name>	SName(Definition Library)	The name of the definition library to be imported.
<import-uri>	Filepath or URI	The URI of the RAPID Model from which a child element will be imported. May be a file path (absolute, or relative to this RAPID Model file) or a URI.
<alias>	Name	Arbitrary short alias for this Import. May be used to disambiguate references to imported or locally defined model elements. MUST be unique in the context of this file.

Child Elements

None.

Parent Elements

None.

Discussion

References to Imported Definitions

Once a Data Model, Resource API, or Definition Library has been imported, its contained definitions may be referenced in qualified or scoped forms, depending on whether the referenced model element is uniquely named within the addressable scope of the referrer, and depending on whether the Import statement specifies an alias.

The rules for allowed reference forms are described in Model Element References.

Imported File Specification

- Conformant implementations MUST allow imports using URLs with an HTTP or HTTPS scheme. URLs with these schemes are expected to be portable across implementations and environments.
- Conformant implementations MUST allow imports using URLs with a FILE scheme. URLs with these schemes are expected to be portable across implementations. FILE URLs are portable across environments only in cases where the referenced models are accessible through the same file structure specified in the URL.
- Implementations MAY allow imports using URLs with schemes other than FILE, HTTP or HTTPS. URLs with these schemes are not expected to be portable across implementations and environments.
- Conformant implementations MUST allow imports using relative and absolute file paths in place of the <import-uri>. The allowable file path is defined by the filesystem in use, and therefore some forms of file paths may not be portable across environments. Relative file paths SHOULD be interpreted as relative to the importing model.

RAPID Model

0 Comments

The `rapidModel` element is the top-level container in an RAPID-ML document.

Syntax

rapidModel <rapid-model-name>
    [<resource-api>]...
    [<data-model>]...
    [<primitive-types-library>]
    [<media-types-library>]
    [<link-relations-library>]
    [<security-schemes-library>]

Examples

```
rapidModel TaxBlaster
    resourceAPI InterfaceModel baseURI "http://localhost:8080"
    dataModel DataModel
```

```
rapidModel TaxBlaster
    resourceAPI InterfaceModel1 baseURI "http://localhost:8080/taxblaster1"
    resourceAPI InterfaceModel2 baseURI "http://localhost:8080/taxblaster2"
    dataModel DataModel1
    dataModel DataModel2
```

Parameters

Name	Type	Description
<rapid-model-name>	Name	The Name of this RAPID Model.

Child Elements

Name	Topic	Description
<resource-api>	Resource API	A Resource API, containing a related collection of resource definitions.
<data-model>	Data Model	A Data Model, containing arelated collection of Data Structure and User-defined Type definitions.
<primitive-types-library>	Primitive Types Library	Container for custom primitive type definitions.
<media-types-library>	Media Types Library	Container for custom media type definitions.
<link-relations-library>	Link Relations Library	Container for custom link relations definitions.
<security-schemes-library>	Security	Container for custom security scheme definitions.

	Schemes Library	

Parent Elements

None.

0 Comments

Code and Documentation Comments

0 Comments

Documentation comments MAY be added to many RAPID-ML™ language elements, and code comments MAY be added *anywhere* in the code. Documentation comments may span multiple lines. Code comments have two forms: single-line and multi-line.

Syntax

Documentation Comments

/** <comment> */

Single-line Code Comments

// <comment>

Multi-line Code Comments

/* <comment> */

Examples

```
/** This is a documentation comment for the TaxBlaster rapidModel element. */
rapidModel TaxBlaster
   /** This is a documentation comment for the InterfaceModel resourceAPI element. */
   resourceAPI InterfaceModel baseURI "http://localhost:8080" // This is a single-line code
   comment inserted at the end of a line.
   /** This is a documentation comment for the DataModel dataModel element. */

   // This is a single-line code comment, which can be inserted *anywhere*.

   dataModel DataModel
   /* This is a multi-line
      code
      comment. */
      /** This is a documentation comment
         spanning
         multiple
         lines. */
      structure MyDataType
         id : string
```

Parameters

Name	Type	Description
<comment>	Free text	A free text description of an element or some aspect of the code.

Child Elements

None.

Parent Elements

None.

Discussion

Documentation for Model Elements

Documentation comments MAY precede all of the elements listed below:

- RAPID Model
- Resource API
- Collection Resource
- Object Resource
- Template Parameter
- Method
- Request
- Response
- Data Model
- Data Structure
- Primitive Property
- Reference Property
- Simple Type
- Enumeration
- Enumeration Constant
- Media Type Definition
- Link Relation Definition
- Security Scheme Library
- Security Scheme Definition

Whitespace Normalization in Documentation Comments

Within a documentation comment, whitespace is normalized as follows:

- A whitespace sequence containing zero or more spaces and/or tabs, and containing a single line break anywhere in the sequence, is normalized to a single space.
- A whitespace sequence containing zero or more spaces and/or tabs, and containing two line breaks anywhere in the sequence, is normalized to a sequence of two line breaks, resulting in a blank line between the preceding and following non-whitespace character sequences.

Conformant editors and parsers MUST apply these whitespace normalization rules.

Documentation as Model Properties

Model elements allowing documentation comments treat the whitespace-normalized documentation content as an element property. Conformant parsers SHOULD preserve the whitespace-normalized documentation in the abstract syntax tree or other intermediate and output representations.

Code comments are not part of the model, and are not expected to be preserved as model element properties Conformant parsers MAY safely ignore or discard code comments.

0 Comments

Data Model

0 Comments

The **dataModel** element defines data types which MAY be realized in Resources, Request or Response messages and MAY be used by other data types. The **dataModel** MAY contain any combination of Data Structure, Enumeration and Simple Type definitions.

Syntax

dataModel <data-model-name>
[<structure> | <enum> | <simple-type>]...

Examples

```
dataModel GeneralTypes
   structure TaxFiling
      ...
   structure Person
      ...
   enum int TaxFilingStatus
      ...
   simpleType SSN defined as string
      ...
   structure Address
      ...
```

Parameters

Field Name	Type	Description
<data-model-name>	Name	The Name assigned to the Data Model. MUST be unique in the context of this RAPID Model.

Child Elements

Name	Topic	Description
<structure>	Data Structure	Data Structure definition.
<enum>	Enumeration	Enumeration definition.
<simple-type>	Simple Type	Simple Type definition.

Parent Elements

- RAPID Model

0 Comments

Simple Type

0 Comments

The **simpleType** element defines a data type based on either a Built-in Primitive Type, a User-defined Primitive Type or on another Simple Type. Simple Types MAY be defined with or without Constraints. When defined without Constraints they simply provide an additional layer of meaning over the underlying primitive type.

Syntax

simpleType <simpletype-name> [~defined] as (<built-in-primitive-type> | <user-defined-primitive-type> | <simple-type>)
    [<constraint-spec>]...

Examples

```
// Based on Built-in string type.
simpleType SocialSecurityNumber defined as string
   matching regex '(?!000|666)[0-8][0-9]{2}-(?!00)[0-9]{2}-(?!0000)[0-9]{4}'
// Based on Built-in integer type.
simpleType DayOfYear1 as integer
   valueRange from '1' to '366'
// Based on Built-in integer type.
simpleType DayOfYear defined as integer
   with valueRange from minimum '1' up to maximum '366' inclusive
// Based on Built-in integer type.
simpleType WholesaleQuantity as integer
   with valueRange from minimum '1000'
// Based on Built-in decimal type.
simpleType FractionalValue as decimal
   with valueRange from '0' exclusive up to '1' exclusive
// Based on Built-in integer type.
simpleType NoMoreThan12 defined as integer
   valueRange up to '12'
// Based on another Simple Type.
simpleType ChildTravelerAge defined as NoMoreThan12
```

Parameters

Field Name	Type	Description
<simpletype-name>	Name	The Name assigned to this user-defined type.
<built-in-primitive-type>	SName(Built-in Primitive Type)	The Name of a Built-in Primitive Type from which this type is derived.
<user-defined-primitive-type>	QName(Primitive Type)	The QName of a User-defined Primitive Type from which this type is derived.
<simple-type>	QName(Simple Type)	The QName of another Simple Type from which this type is derived.

Child Elements

Name	Topic	Description
<constraint-spec>	Constraints	One or more Constraint specifications.

Parent Elements

- Data Model

Discussion

A Primitive Property MAY specify a user-defined Simple Type as data type of the property.

A Simple Type may also derive from another Simple Type. Circular dependencies in the Simple Type derivation graph are disallowed, and processors SHOULD guard against such cycles.

A Simple Type MAY NOT be used as the data type of a URI parameter or Method parameter.

0 Comments

Enumeration

0 Comments

The **enum** element defines a data type consisting of a set of named Enumeration Constants. Enumeration elements are of type int or string.

Syntax

enum (int | string) <enum-name>
    [<enumeration-constant>]...

Examples

```
/** Integer enum using assigned, sequential integer values. */
enum int TaxFilingStatusEnum
    DRAFT
    PENDING_CPA_REVIEW
    PENDING_CLIENT_REVIEW
    FILED
    AMENDED
    CLOSED

/** Integer enum using explicit integer values. */
enum int SpecialValueEnum
    NORMAL_VALUE : 0
    NOT_AVAILABLE : -65534
    NOT_APPLICABLE : -65533
    RESTRICTED : -65532

/** Currency code enum using explicit string values. */
enum string CurrencyCodeEnum
    EUR : "Euro"
    CAD : "Canadian Dollar"
    USD : "US Dollar"
    CHF : "Swiss Franc"
    JPY : "Japanese Yen"
    INR : "Indian Rupee"
    BRL : "Brazilian Real"
```

Parameters

Field Name	Type	Description
int	SName(Built-in Primitive Type - int)	Indicates that this is an int type Enumeration.
string	SName(Built-in Primitive Type - string)	Indicates that this is a string type Enumeration.
<enum-name>	Name	The Name assigned to this Enumeration. MUST be unique in the context of the parent Data Model.

Child Elements

Name	Topic	Description

<enumeration-constant>	Enumeration Constant	Defines a named value included in the Enumeration.

Parent Elements

- Data Model

0 Comments

Enumeration Constant

0 Comments

An Enumeration Constant specifies a uniquely named value within an Enumeration. The value may be defined explicitly, or derived implicitly from the name or ordinal position of the Enumeration Constant. (See the discussion for a full explanation.)

Syntax

<enumeration-constant-name> [: (<int-value> | "<string-value>")]

Examples

See the Examples section of the Enumeration topic.

Parameters

Field Name	Type	Description
<enumeration-constant-name>	Name	The Name assigned to the Enumeration Constant, which MUST be unique within the containing Enumeration. Conventionally in UPPER_UNDERSCORE style.
<int-value>	Integer Value	An integer value, which MAY be assigned to the Enumeration Constant if the containing Enumeration is of integer type. In the absence of an explicitly assigned value, the Enumeration Constant is assigned an implicit value equal to its zero-based ordinal position within the containing Enumeration.
<string-value>	String Value	A string value, which MAY be assigned to the Enumeration Constant if the containing Enumeration is of string type. In the absence of an explicitly assigned value, the Enumeration Constant's name is implicitly assigned as the value.

Child Elements

None.

Parent Elements

- Enumeration

Discussion

Default Values

Conformant tools MUST apply default values to Enumeration constants not having explicitly defined values, according to the rules specified in Parameters, above. A single Enumeration may have Enumeration Constants with a combination of implicit default values and explicitly

assigned values. If an implicitly assigned value duplicates an explicitly assigned value, conformant editors SHOULD warn the user of this condition, which the user may subsequently resolved by assigning an explicit value.

0 Comments

Data Structure

0 Comments

The **structure** element defines a canonical Data Structure, which may be realized in Resource Definitions, Request and Response Messages.

Syntax

structure <structure-name>
    [<property>]...
    [<data-example>]...

Examples

```
structure TaxFiling
   filingID : string
   taxpayer : reference to Person
   jurisdiction : string
   year : gYear
   period : int
   currency : string
   grossIncome : decimal
   taxLiability : decimal
   dataExample ''' <?xml version="1.0" encoding="UTF-8"?>
      <TaxFiling version="1.0">
         <filingID>#123456</filingID>
         <taxpayer>ssn-xx-xxxx</taxpayer>
         <jurisdiction>PA0010000</jurisdiction>
         <year>2014</year>
         <period></period>
         <currency>USD</currency>
         <grossIncome>32,000.00</grossIncome>
         <taxLiability>9,600.00</taxLiability>
      </TaxFiling> '''
```

Parameters

Field Name	Type	Description
<structure-name>	Name	The Name assigned to this Data Structure, which MUST be unique within the containing Data Model.

Child Elements

Name	Topic	Description
<property>	Primitive Property or Reference Property	A property definition.
<data-example>	Data Example	An example of what this structure would look like rendered in a particular media type.

Parent Elements

- Data Model

0 Comments

Primitive Property

0 Comments

A Primitive Property holds a primitive value, or a collection of primitive values, of a specified type. The Primitive Property has an assigned data type, which may be a Built-in Primitive Type, a User-defined Primitive Type, an Enumeration or a Simple Type.

Syntax

<property-name> : (<built-in-primitive-type>|<user-defined-primitive-type>|<enumeration>|<simple-type>)[<cardinality-indicator>]
    [<constraint-spec>]...

Examples

```
structure Person
   // Properties using Built-in types
   id : string!
   firstName : string!
   lastName : string[1..1]
   age : int
   // User-defined Enumeration type
   maritalStatus : MaritalStatusEnum
   // User-defined Simple Type
   ssn : SocialSecurityNumber
```

Parameters

Field Name	Type	Description
<property-name>	Name	The Name assigned to this property.
<built-in-primitive-type>	SName(Built-in Primitive Type)	The Built-in Primitive Type assigned to this property.
<user-defined-primitive-type>	QName(User-defined Primitive Type)	A User-defined Primitive Type assigned as the data type of this property.
<enumeration>	QName(Enumeration)	An Enumeration assigned as the data type of this property.
<simple-type>	QName(Simple Type)	A user-defined Simple Type assigned as the data type of this property.
<cardinality-indicator>	Cardinality	Optional Cardinality indicator for this property. Primitive properties have a default Cardinality of 'zero or one'.

Child Elements

Name	Topic	Description
<constraint-spec>	Constraints	Defines constraints on the property value.

Parent Elements

- Data Structure

Discussion

See the Built-in Primitive Data Types topic for a full list of the Built-in Primitive Property types.

0 Comments

Reference Property

0 Comments

Reference properties are denoted by the **reference** keyword. A Reference Property refers to an instance of a specified Data Structure in the same Data Model or in an imported Data Model.

Syntax

<property-name> : [~as] [containing] reference [~to] <referenced-structure> [inverse <inverse-reference>][<cardinality-indicator>]

Examples

```
structure TaxFiling
   taxpayer : as reference to Person inverse taxfiling

structure Person
   taxfiling : as reference to TaxFiling inverse taxpayer
   homeAddress : as containing reference to Address
   workAddress :as reference to Address

structure Address
```

Parameters

Field Name	Type	Description
<property-name>	Name	The name assigned to this property.
containing	keyword	Indicates that the referenced object is contained by the referring object. See the discussion below for more information.
<referenced-structure>	QName(Data Structure)	The target of this Reference Property.
<inverse-reference>	SName(Reference Property)	Specifies a Reference Property in the referenced Data Structure that forms a bi-directional association with the referring Data Structure. See the discussion below for more information.

Child Elements

None.

Parent Elements

- Data Structure

Discussion

Containing

The optional **containing** keyword indicates a whole/part relationship, where the referenced object is contained by the referring object. The reference should be managed at runtime to enforce UML composition semantics, namely:

- An instance of the contained structure can have at most one container at a time. It cannot be referenced simultaneously through more than one containing reference, even if those containing references are defined separately.
- If a composite is deleted, all of its parts are normally deleted with it.

Note that it is up to the application to enforce the above semantics at runtime. Generated code SHOULD enforce containment semantics where such enforcement is a natural part of the scope of the generated code; but this is generally not sufficient to guarantee these semantics.

Inverse

The **inverse** keyword means that the reference guarantees consistency in the following forms:

- In the simplest case, starting with object O1, traversing the association through its reference R1 yields an object O2 whose specified inverse reference R2 yields O1.
- One or both of the references may be multi-valued (Cardinality 0..* or 1..*). In this case:
  - Starting with object O1, traversing a multi-valued inverse reference R1 yields a set of objects, each of which refers back to O1 through its inverse reference R2.
  - R2 refers back to O1 if:
    - (a) R2 is a single-valued reference whose value is O1; or
    - (b) R2 is a multi-valued reference, at least one of whose values is O1.

The inverse keyword is only valid if it meets the following conditions:

- **inverse** must be specified on both ends of the association.
- The inverse properties must be reciprocal, such that the named reference properties on each side of the association refer to each other.
- The types of the references must be consistent.

To summarize: If Structure S1 defines a reference R1 to structure S2, declared as an inverse of Reference Property R2, then S2 must define the reference R2 to S1, declared as an inverse of Reference Property R1.

0 Comments

Data Example

0 Comments

The **dataExample** element specifies an example message representation of the containing Data Structure. The Data Example may be used for documentation and testing.

Syntax

dataExample ''' <example-text> '''

Examples

```
dataExample ''' <?xml version="1.0" encoding="UTF-8"?>
   <TaxFiling version="1.0">
      <filingID>#123456</filingID>
      <taxpayer>ssn-xx-xxxx</taxpayer>
      <jurisdiction>PA0010000</jurisdiction>
      <year>2014</year>
      <period></period>
      <currency>USD</currency>
      <grossIncome>32,000.00</grossIncome>
      <taxLiability>9,600.00</taxLiability>
   </TaxFiling> '''
```

Parameters

Field Name	Type	Description
<example-text>	Free text	Ideally an example of what a legal message of this data type would look like represented in a particular media type.

Child Elements

None.

Parent Elements

- Data Structure

0 Comments

Constraints

0 Comments

Constraints define the domain of legal values for a Simple Type, a Primitive Property, or a Primitive Property Realization. There are four types of Constraint: numeric value range, string fixed length, string length, and regular expression.

Syntax

Numeric value range constraint

[~with] valueRange from [~minimum] <min-value> [inclusive | exclusive] [~up] to [~maximum] <max-value> [inclusive | exclusive]

String fixed length constraint

[~of] length <fixed-length>

String length constraint

[~of] length [from [~minimum] <min-length>] [[~up] to [~maximum] <max-length>]

String regular expression constraint

[~matching] regex r"<regular-expression>"

Examples

```
// Integer value range constraint applied to Built-in int type property within structure
definition
structure TaxFiling
   dayOfYear : int
      with valueRange from minimum 1 inclusive up to maximum 366 inclusive

// String fixed length constraint applied to User-defined Simple Type
simpleType CurrencyCode defined as string
   of length 3

// String length constraint applied to a User-defined Simple Type
simpleType UserName defined as string
   of length from minimum 6 up to maximum 12

// String regular expression constraint applied to Built-in structure property in Realization
property set
objectResource PersonObject type Person
   URI people/{id}
   with all properties including
      ssn
         matching regex r"(?!000|666)[0-8][0-9]{2}-(?!00)[0-9]{2}-(?!0000)[0-9]{4}"
```

Parameters

Numeric Value Range Constraint

Field Name	Type	Description
<min-value>	number	The lower bound of the value range, specified as an integer or decimal number.

<max-value>	number	The upper bound of the value range, specified as an integer or decimal number.
inclusive	keyword	Indicates that the value range includes the preceding <min-value> or <max-value>. Upper and lower bounds are inclusive by default. The inclusive keyword is provided only for readability.
exclusive	keyword	Indicates that the value range excludes the preceding <min-value> or <max-value>.

String Fixed Length Constraint

Field Name	Type	Description
<fixed-length>	integer	The required length of the string, specified as a non-negative integer. A string value is valid only if its length matches this value exactly.

String Length Constraint

Field Name	Type	Description
<min-length>	integer	The minimum legal length for this string, specified as a non-negative integer.
<max-length>	integer	The maximum legal length for this string, specified as a non-negative integer.

String Regular Expression Constraint

Field Name	Type	Description
<regular-expression>	string	A regular expression which the string value must match. Since regular expressions often contain backslashes and other special characters, *raw string* literal syntax is recommended. See the Discussion below.

Child Elements

None.

Parent Elements

- Simple Type
- Primitive Property (excluding Enumeration)
- Property Set (within a Primitive Property realization)

Discussion

String Length Sub-Constraints

The String Length Constraint allows minimum length and maximum length sub-constraints. A valid string length Constraint MUST specify at least one of these two sub-constraints, and MAY specify both.

Regular Expression Syntax

RAPID-ML supports the subset of the ECMA 262 regular expression dialect recommended in the JSON Schema specification.

Note that beginning- and end-of-string anchors are implicitly defined. Explicit anchors are disallowed in RAPID-ML regular expressions.

Raw Strings

String literals in RAPID-ML may be specified as normal strings or raw strings. Normal string literals are delimited by single or double quotes, and support standard escape sequences. Raw string literals use the form `r"<string>"` - a lower-case 'r' followed by a double-quoted string. Raw strings are not escaped, so backslashes and other special characters are treated literally.

For this reason, raw string literals are recommended for regular expression strings.

0 Comments

Cardinality

0 Comments

Cardinality is the measure of the number of elements in a set. In RAPID-ML™, Cardinality indicators are used in conjunction with Data Structure properties and when modeling Data Structure realizations in Resource API definitions.

Syntax

(<property-definition>|<property-name>)(<cardinality-symbol> | <cardinality-spec>)

Examples

```
// Applied after the types of properties within a structure
otherNames : string*
addresses : as reference to Address[1..*]
```

```
// Applied after the Names of properties in a Property Set defined in a Collection Resource or
Object Resource
with all properties including
   filingID!
   addresses[1..*]
```

Parameters

Field Name	Type	Description
<property-definition>	Primitive Property or Reference Property	In a property definition, the Cardinality follows the property type.
<property-name>	SName(Primitive Property)or SName(Reference Property)	In a Property Set, the Cardinality follows the property name.
<cardinality-symbol>	Cardinality symbol	A single-character symbol indicating the Cardinality in short form. See Notation table below.
<cardinality-spec>	Cardinality expression	A square bracket-delimited expression indicating the Cardinality in long form. See Notation table below.

Child Elements

Not applicable.

Parent Elements

Not applicable.

Discussion

Notation

Symbol	Long Form	Description
Omitted ("")	N/A	zero or one
?	[0..1]	zero or one
*	[0..*]	zero or more
!	[1..1]	exactly one
+	[1..*]	one or more

0 Comments

Resource API

0 Comments

The **resourceAPI** element defines a REST-style API as a collection of resources provided to consuming applications. The API may optionally be secured by one or more specified security schemes.

Syntax

resourceAPI <resource-api-name> baseURI "<uri>"
[<secured>]
[<collection-resource> | <object-resource>]...

Examples

```
resourceAPI InterfaceModel baseURI "http://localhost:8080"
   secured by
      QueryStringKeyAuth
      auth.OAuth2
         authorized for scopes
            admin, manager
   collectionResource TaxFilingCollection type TaxFiling
      ...
   collectionResource PersonCollection type Person
      ...
   objectResource TaxFilingObject type TaxFiling
      ...
   objectResource PersonObject type Person
      ...
   objectResource Addressbject type Address
      ...
```

Parameters

Field Name	Type	Description
<resource-api-name>	Name	The Name assigned to this Resource API.
<uri>	URI	The base URI of the resources defined in this Resource API which themselves declare URIs relative to this.

Child Elements

Name	Topic	Description
<secured>	Secured	Applies a Security Scheme to the Resource API.
<collection-resource>	Collection Resource	Defines a resource representing a collection of objects.
<object-resource>	Object Resource	Defines a resource representing a single object.

Parent Elements

- RAPID Model

0 Comments

Collection Resource

0 Comments

The **collectionResouce** element describes how a collection of a particular data type is exposed by the API and how it may be accessed. Collection Resources are bound to a Data Structure from an *accessible* Data Model.

Syntax

[default] collectionResource <resource-name> [~bound ~to] type <bound-data-structure>
    [<uri>]
    [<secured>]
    [<property-set>]
    [<reference-embed> | <reference-link>]...
    [<link-descriptor-definition>]...
    [<media-types>]
    [<method>]...
    [<example>]...

Examples

```
default collectionResource PersonCollection type Person
    URI people
    secured by
     auth.Basic
    // Combined Property Set: uses including and excluding keywords
    with all properties
    including
        taxpayerID!
    exluding
        otherNames
    // Reference treatment: embedded
    referenceEmbed > addresses
    linkDescriptor MyResourceLink
        firstName
        lastName
    mediaTypes
        application/json
    method GET getPersonCollection
    method POST postPersonCollection
    example ''' Any text can go here '''
    example ''' But this should show what a legal message from this resource should look like
    '''
```

Parameters

Name	Type	Description
default	keyword	Explicitly designates this Collection Resource as the default hyperlink target for references to the bound Data Structure. See Automatic Linking and Embedding for a full explanation.
<resource-name>	Name	The Name of this resource. Unique in the context of this Resource API.

<bound-data-structure>	QName(Data Structure)	A reference to the underlying Data Structure bound to this Collection Resource.

Child Elements

Name	Topic	Description
<uri>	URI	The URI of this resource, relative to the baseURI of the containing Resource API.
<secured>	Secured	Describes how this resource is secured.
<property-set>	Property Set	Part of the default realization of this resource.
<reference-embed>	Reference Embed	Part of the default realization of this resource.
<reference-link>	Reference Link	Part of the default realization of this resource.
<link-descriptor-definition>	Link Descriptor Definition	A reusable descriptor for use in Reference Links from other resources.
<media-types>	Media Types	The default media types supported by this resource.
<method>	Method	An HTTP method definition.
<example>	Example	An example of what valid data passing to/from this resource should look like.

Parent Elements

- Resource API

Discussion

Linked vs. Embedded Realization

A Collection Resource represents a list of references to its bound Data Structure. These references may be realized as embedded representations, or as hyperlinks, according to the following rules:

- If the Collection Resource specifies a Property Set, specifies one or more Reference Links, or specifies one or more Reference Embeds:
  - The Collection Resource is realized as a list of embedded objects, following the realization model specified in the Property Set, Reference Links and/or Reference Embeds.
- If the Collection Resource does not specify a Property Set and does not specify any Reference Links or Reference Embeds:
  - If there is a Default Object Resource for the bound Data Structure in scope, the Collection Resource will be realized as a list of hyperlinks targeting the Default Object Resource.
  - If there is not a Default Object Resource for the bound Data Structure in scope, the Collection Resource will be realized as a list embedded objects, using the default realization rules.

0 Comments

Object Resource

0 Comments

The **objectResouce** element describes how an instance of a particular data type is exposed by the API and how it may be accessed. Object Resources are bound to a Data Structure from an *accessible* Data Model. An **objectResource** is a reference to a single resource object which is always realized as embedded (it would make no sense to realize the object as a hyperlink to another object resource).

Syntax

[default] objectResource <resource-name> [~bound ~to] type <bound-data-structure>
[<uri>]
[<secured>]
[<property-set>]
[<reference-embed> | <reference-link>]...
[<link-descriptor-definition>]...
[<media-types>]
[<method>]...
[<example>]...

Examples

```
default objectResource PersonObject type Person
   URI people/{id}
   secured by
      auth.Basic
   // Combined Property Set: uses including and excluding keywords
   with all properties
   including
      taxpayerID!
      ssn
         matching regex '(?!000|666)[0-8][0-9]{2}-(?!00)[0-9]{2}-(?!0000)[0-9]{4}'
   exluding
      otherNames
   // Reference treatment: embedded
   referenceLink > addresses
   linkDescriptor MyResourceLink
      firstName
      lastName
   mediaTypes
      application/json
   method GET getPersonObject
   method PUT putPersonObject
   example ''' Any text can go here '''
   example ''' But this should show what a legal message from this resource should look like
   '''
```

Parameters

Name	Type	Description
default	keyword	Explicitly designates this Object Resource as the default hyperlink target for references to the bound Data Structure. See Automatic Linking and Embedding for a full explanation.

<resource-name>	Name	The Name assigned to this resource. Unique in the context of the containing Resource API.
<bound-data-structure>	QName(Data Structure)	A reference to the underlying Data Structure bound to this Object Resource.

Child Elements

Name	Topic	Description
<uri>	URI	The URI of this resource, relative to the baseURI of the containing Resource API.
<secured>	Secured	Describes how this resource is secured.
<property-set>	Property Set	Part of the default realization of this resource.
<reference-embed>	Reference Embed	Part of the default realization of this resource.
<reference-link>	Reference Link	Part of the default realization of this resource.
<link-descriptor-definition>	Link Descriptor Definition	A reusable descriptor for use in Reference Links from other resources.
<media-types>	Media Types	The default media types supported by this resource.
<method>	Method	An HTTP method definition.
<example>	Example	An example of what valid data passing to/from this resource should look like.

Parent Elements

- Resource API

0 Comments

URI

0 Comments

The **URI** element specifies the URI for this resource, relative to the **baseURI** of the containing Resource API.

Syntax

URI <relative-uri>
    [<template-parameter>]...

Examples

```
URI people/{id}
   required templateParam id bound to property taxpayerID
```

Parameters

Field Name	Type	Description
<relative-uri>	Relative URI	The relative URI of the containing resource with '/' delimiters. It MAY contain one or more Template Variables surrounded by '{' and '}'. These variables mark a section of a URI as replaceable using template parameters.

Child Elements

Name	Topic	Description
<template-parameter>	Template Parameter	Specifies the data type or property binding of a variable specified in the relative URI.

Parent Elements

- Collection Resource
- Object Resource

0 Comments

Template Parameter

0 Comments

The **templateParam** element defines the data type or property binding of a variable specified in the containing URI.

Syntax

[required] templateParam <variable-name> ([~bound [~to]] property <property-name>) | ([~of] type <data-type>)

Examples

```
required templateParam id bound to property taxpayerID
```

```
templateParam id of type string
```

Parameters

Field Name	Type	Description
required	keyword	If present, specifies that the template parameter is required by this resource. A URI without a specified value for this template parameter will be considered invalid.
<variable-name>	SName(URI Template Variable)	The name of the template variable, which MUST be defined in the containing URI.
<property-name>	SName(Primitive Property)	The name of a Primitive Property defined in the Data Structure to which the parent resource is bound. This property binding indicates that the template parameter is used as an identifier, filter, or other data-bound operator, associated directly with the bound property. It also implies that the parameter value SHOULD match the data type of the bound property.
<data-type>	Name(Built-in Primitive Data Type)	The name of one of the Built-in Primitive Data Types, indicating the expected data type of the parameter.

Child Elements

None.

Parent Elements

- URI

0 Comments

Realization Modeling

0 Comments

RAPID-ML promotes the concept of realization modeling, whereby canonical Data Structures can either be fully embedded (accessed via a single Collection Resource or Object Resource) or linked (factored out across multiple resources), enabling clients to get data in an economic, or lazy, manner.

Realization modeling also enables Property Sets to be defined for purposes of filtering out unwanted properties from the canonical types, for layering on Cardinality overrides and other Constraints that make sense in the application or service context. RAPID-ML realization modeling overcomes traditional, one-size-fits-no-one arguments against canonical data usage and enables emergent data type standardization across APIs, making these more interoperable, easier to learn and to integrate.

0 Comments

Automatic Linking and Embedding

0 Comments

About Automatic Linking and Embedding

RAPID-ML™ promotes convention over configuration by including useful and intuitive default behaviors. Modelers only need to specify relationships and behaviors that differ from the defaults.

One of the important areas of behavior in a Resource API is linking and embedding -- the realization of a Reference Property as a hyperlink to another resource, or as an embedded object within the referring resource. RAPID-ML determines linked vs. embedded realization according to the rules defined here.

About Default Resources

When a resource or a request or response message is bound to a Data Structure, the Data Structure is realized within this bound context. The resource or message may include a Property Set, Reference Links and/or Reference Embeds in order to explicitly specify the realization. But there are rules that determine a default realization for any Reference Property whose realization is not explicitly specified as a Reference Link or Reference Embed.

Where possible, RAPID-ML's default realization rules will interpret reference properties as hyperlinks. In order to do this, there needs to be an appropriate target resource for the hyperlink. We refer to this as a default resource.

More specifically, a default resource is applicable to reference properties:

- realized within a given Resource API;
- referring a specific Data Structure;
- with a given multiplicity - single-valued or multi-valued.

So there is not one single "default resource." Within the context of a Resource API, different accessible resources (locally defined or imported) can serve as default resources. Each default resource is the default hyperlink target for either single-valued or multi-valued references to a certain data type.

Rules for Determining Default Resources

Within a Resource API, the default resource for references to a given Data Structure, with a given given multiplicity, is determined as follows:

- If there is no resource in the addressable scope that is bound to the referenced Data Structure and that has the required multiplicity -- an Object Resource for a single-valued reference or a Collection Resource for a multi-valued reference -- then there is no default resource.
- If there is exactly one resource in scope that is bound to the referenced Data Structure and that has the required multiplicity, that resource is designated implicitly as the default resource.
- If there is more than one resource in scope that is bound to the referenced Data Structure and that has the required multiplicity:
  - If exactly one of those resources is explicitly designated as the default resource by use of the default qualifier keyword, that resource is the default resource.

    If none of those resources is explicitly designated as the Default Resource, or if more than

- one of those resources is so designated, then there is no default resource.

Default Link Descriptors

In RAPID-ML, Reference Links can be plain hyperlinks, or "decorated" with one or more properties of the target Data Structure. The most direct way to decorate a hyperlink is by adding target properties to the Reference Link.

Sometimes it is useful for a resource to define a reusable decorated link specification, so that referrers do not have to specify the target properties explicitly in each instance. This reusable specification is called a Link Descriptor Definition. A resource can define any number of link descriptors, and a Reference Link targeting that resource can apply any of these by name.

Furthermore, one of these link descriptors can be designated implicitly or explicitly as the default, such that any resource link targeting that resource will automatically apply the default Link Descriptor Definition unless otherwise specified.

The rules for default link descriptors are as follows:

- If a resource defines exactly one Link Descriptor Definition, that Link Descriptor Definition is implicitly designated as the default.
- If a resource defines more than one Link Descriptor Definition:
  - If one of the defined link descriptors is explicitly designated as the default by use of the default keyword, that Link Descriptor Definition is the default.
  - If none of the defined link descriptors is excplicitly designated as the default, there is no default Link Descriptor Definition.
  - If more than one Link Descriptor Definition on a single resource is defined with the default keyword, this is an error condition.
- Automatic Reference Links will use the target resource's default Link Descriptor Definition, if there is one.
- Explicit Reference Links will use the target resource's default Link Descriptor Definition, unless the Reference Link explicitly specifies a different named Link Descriptor Definition, or specifies a delimited list of target properties.

Default Realization of Reference Properties

Building on the above specified rules for default resources and default link descriptors, reference properties are realized as follows:

- If there is an explicit Reference Link or Reference Embed for the Reference Property, this defines the realization.
  - If there is an explicit Reference Link that does not specify a Link Descriptor Definition or target properties, and the target resource has a default Link Descriptor Definition, that Link Descriptor Definition is applied to the Reference Link.
- If there is no explicit Reference Link or Reference Embed:
  - If there is a default resource in scope for the reference, the reference is realized as a hyperlink. If the default resource has a default Link Descriptor Definition, the Link Descriptor Definition applies to the automatic hyperlink.
  - If there is no default resource in scope, the reference is realized as an embedded object.

0 Comments

Property Set

0 Comments

The **properties** element is used within a resource or message to adapt the set of properties in the bound Data Structure for optimal use within the API context. A Property Set may specify a subset of the bound properties, and may add context-specific Constraints to the included properties. Property Sets are one aspect of realization modeling.

Syntax

```
[~with] (all | only) properties
[~including]
   (<property-name>[<cardinality-indicator>]
      [<constraint-spec>]...
   )...
[excluding
   <property-name>...
]
```

Examples

```
// 1. Itemized
with only properties
   id! // May include constraints as well.
   ssn
   firstName
   lastName
```

```
// 2. Constraining
with all properties including
   id! // Add cardinality override
   ssn // Add regex contraint
      matching regex '(?!000|666)[0-8][0-9]{2}-(?!00)[0-9]{2}-(?!0000)[0-9]{4}'
```

```
// 3. Excluding
with all properties excluding
   id
```

```
// 4. Combination, constraining and excluding
with all properties
including
   id!
excluding
   addresses
```

Parameters

Field Name	Type	Description
all	keyword	Indicates that the Property Set will contain all properties except those that are explicitly excluded (if any).
only	keyword	Indicates that the Property Set will contain only those properties that are explicitly listed. This is the default

		behavior, so the only keyword may be omitted.
excluding	keyword	Indicates the beginning of the excluding clause. Properties listed here will not be included in the Property Set. The excluding clause MAY be used alone or in combination with an inclusive all properties list. It MUST NOT be used in combination with an exclusive only properties list.
<property-name>	SName	The name of a Primitive Property or Reference Property to be included and/or constrained, or to be excluded. Multiple properties MAY be specified as a delimited list.
<cardinality-indicator>	Cardinality	Optional Cardinality indicator for an included property. May be used to narrow but not broaden Cardinality of the property as defined in its containing Data Structure.
<constraint-spec>	Constraint	A context-specific Constraint applied to the preceding property. If the property already has (directly or indirectly) a Constraint of the same type, the Constraint specified in the Property Set overrides the inherited property Constraint.

Child Elements

Name	Topic	Description
<constraint-spec>	Constraints	Describes Constraints that apply to a particular property in this context.

Parent Elements

- Collection Resource
- Object Resource
- Request
- Response

0 Comments

Link Descriptor Definition

0 Comments

The **linkDescriptor** element is used to define a set of properties that may be used to decorate implicit or explicit Reference Links.

Syntax

[default] linkDescriptor <link-descriptor-name>
    [<primitive-property>]...

Examples

```
default linkDescriptor Names
   firstName
   lastName
linkDescriptor NamesAndAddresses
   firstName
   lastaName
   addresses
```

Parameters

Name	Type	Description
default	keyword	Indicates that this Link Descriptor Definition provides the default decoration for Reference Links to the containing resource, except where those Reference Links explicitly apply a different Link Descriptor or explicitly specify **includedProperties**. See Automatic Linking and Embedding for a full explanation.
<link-descriptor-name>	Name	The name of this Link Descriptor. MUST be unique in the context of the containing Collection Resource or Object Resource.
<primitive-property>	SName(Primitive Property)	Reference to a Primitive Property defined in the referenced Data Structure. Multiple primitive properties MAY be specified as a delimited list.

Child Elements

None.

Parent Elements

- Collection Resource
- Object Resource

0 Comments

Reference Embed

0 Comments

The **referenceEmbed** element defines an embedded object representation as the realization of a Reference Property from the Data Structure bound to the containing resource. The Reference Embed element can specify the properties of the referenced Data Structure to be included in the embedded object, and can recursively specify Reference Embed and Reference Link elements to define the realization of included reference properties.

Syntax

referenceEmbed > <reference-property-name>
[targetProperties <target-properties>]
[referenceEmbed <reference-embed>]...
[referenceLink <reference-link>]...

Examples

```
referenceEmbed > taxpayer
```

```
referenceEmbed > taxpayer
   targetProperties
      firstName
      lastName
      DOB
   referenceEmbed > addresses
   referenceLink > taxFilings
```

Parameters

Field Name	Type	Description
<reference-property-name>	SName(Reference Property)	The name of the Reference Property to be realized as an embedded object.
<target-properties>	List(Primitive Property or Reference Property)	A delimited list of Primitive Property and/or Reference Property names, each of which MUST be defined in the referenced Data Structure. If a Reference Property is realized by a <reference-embed> or <reference-link> child element, it is implicitly included as a target property, but MAY still be specified in the target properties list for clarity and completeness.

Child Elements

Name	Topic	Description
<reference-embed>	Reference Embed (this	Recursively specifies an embedded object realization of a Reference Property of the bound Data Structure.

	topic)	
<reference-link>	Reference Link	Specifies a hyperlink realization of a Reference Property of the bound Data Structure.

Parent Elements

- Collection Resource
- Object Resource
- Reference Embed
- Request
- Response

Discussion

Reference Properties will be embedded by default if there is no other accessible default resource bound to the referenced Data Structure, and having the required multiplicity. See Automatic Linking and Embedding for more information.

0 Comments

Reference Link

0 Comments

The **referenceLink** element defines a hyperlink as the realization of a Reference Property in the Data Structure bound to the containing resource. A Reference Link also enables hyperlinks to be *decorated* with selected properties from the referenced Data Structure. Decorated hyperlinks mean that clients can obtain key properties of the referenced Data Structure without having to traverse the hyperlink to the target resource.

Syntax

referenceLink > <reference-property-name>
[targetResource <target-resource>]
[(targetProperties <target-properties>) | (linkDescriptor <link-descriptor>)]
[linkRelation <link-relation>]

Examples

```
// Explicit Reference Link to default target resource
referenceLink > taxpayer
```

```
// Target resource and properties explicitly specified
referenceLink > taxpayer
   targetResource PersonObject2
   targetProperties
      firstName
      lastName
      DOB
   linkRelation about
```

```
// Using linkDescriptor defined in targetResource to select target properties
referenceLink > taxpayer
   targetResource PersonObject2
   linkDescriptor EssentialProperties
```

Parameters

Field Name	Type	Description
<reference-property-name>	SName(Reference Property)	The name of the Reference Property to be realized as a hyperlink.
<target-resource>	QName(Collection Resource or Object Resource)	A reference to an Object Resource or Collection Resource designated as the target of the hyperlink. The target resource MUST be bound to the same Data Structure as the Reference Property.
<target-properties>	List(SName(Primitive Property))	A delimited list of Primitive Property names, each of which MUST be defined in the referenced Data Structure. A Reference Link MAY specify <target-properties> or a <link-descriptor>, but not both.

<link-descriptor>	SName(Link Descriptor)	The name of a Link Descriptor to be applied to this Reference Link. The referenced Link Descriptor MUST be defined within the target resource. The <link-descriptor> parameter MAY be used only if the <target-resource> is explicitly specified, and MUST NOT be used in combination with <target-properties>.
<link-relation>	SName(Built-in Link Relation) or QName(User-defined Link Relation)	A reference to a Built-in or User-defined link relation that describes the semantics of the link.

Child Elements

None.

Parent Elements

- Collection Resource
- Object Resource
- Reference Embed
- Request
- Response

Discussion

Reference Properties will be hyperlinked by default if there is an available default resource bound to the referenced Data Structure, and having the required multiplicity. See Automatic Linking and Embedding for more information.

0 Comments

Example

0 Comments

There are two types of Example element: `example` and `externalExample`. Both types are used to add an example message to the parent element. An example message documents what a legal message would look like.

Syntax

example '''<example-message-text>'''
externalExample "<example-message-file>"

Examples

```
example ''' <?xml version="1.0" encoding="UTF-8"?>
   <TaxFiling version="1.0">
      <filingID>#123456</filingID>
      <taxpayer>ssn-xx-xxxx</taxpayer>
      <jurisdiction>PA0010000</jurisdiction>
      <year>2014</year>
      <period></period>
      <currency>USD</currency>
      <grossIncome>124,000.00</grossIncome>
      <taxLiability>45,000.00</taxLiability>
   </TaxFiling> '''
```

```
externalExample "messages/TaxFiling.xml"
```

Parameters

Name	Type	Description
<example-message-text>	Free text	Representation of a message that would be valid in this context.
<example-message-file>	File path	Absolute or relative path to a text file containing a representation of a valid message. Relative paths should be interpreted as relative to the referring RAPID Model.

Child Elements

None.

Parent Elements

- Collection Resource
- Object Resource
- Request
- Response

0 Comments

Method

0 Comments

The **method** element defines an operation which may be invoked on the containing resource. The Method specifies an HTTP verb, and contains request and response message definitions.

Syntax

```
method <http-method> <method-name>
   [<request>]
   [<response>...]
```

Examples

```
method GET getTaxFilingCollection
   request
   response TaxFilingCollection statusCode 200

method POST updateTaxFilingCollection
   request TaxFilingCollection
   response statusCode 200
   response statusCode 400
```

Parameters

Field Name	Type	Description
<http-method>	keyword	One of the HTTP methods as defined in section 9 of *RFC 2616*: CONNECT, DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT and TRACE.
<method-name>	Name	Name assigned to this Method. Must be unique in the context of the containing Resource API.

Child Elements

Name	Topic	Description
<request>	Request	An optional request message specification. If omitted, the Method is assumed to have an empty request.
<response>	Response	An optional response message specification. If omitted, the Method is assumed to have a single response with an empty message payload and a statusCode of 200.

Parent Elements

- Collection Resource
- Object Resource

0 Comments

Request and Response Messages

0 Comments

The **request** and **response** elements define the Request and Response Messages, respectively, for a Method. The message specification includes the message body, or payload, message parameters, and (for a response message) the status code. The message payload is specified as a Data Structure realization; either the same realization as the containing resource, a different resource, or a different Data Structure altogether.

Syntax

Message with Empty Payload

request | (response [statusCode <response-code>])
    [mediaTypes <media-types>]
    [<message-param>]...
    [<example> | <external-example>]...

Message with Resource-Defined Payload

(request | response) [~with] ((this [[~resource] <resource-realization>]) | ([~resource] <resource-realization>)) [statusCode <response-code>]
    [mediaTypes <media-types>]
    [<message-param>]...
    [<example> | <external-example>]...

Message with Data Structure Realization as Payload

(request | response) [~with] type <data-structure> [statusCode <response-code>]
    [<property-set>]
    [<reference-link> | <reference-embed>]...
    [mediaTypes <media-types>]
    [<message-param>]...
    [<example> | <external-example>]...

Examples

```
method GET getPersonObject
    // Empty request
    request
    // response with a representation of the current resource
    response with this PersonObject statusCode 200

// Request with empty message payload and optional parameters
method GET getPersonCollection
    request
        param nameContains of type string in query
        param inCountry of type string in query
        param includeInactive of type boolean in query

// Request with a representation of the current resource
method PATCH updatePersonObject
    request with this PersonObject

// Request with a representation defined by another resource
```

```
method GET searchPersonCollection
   request with resource SavedSearchObject

// Request with a specialized data structure realization
method POST addPersonToCollection
   request with type Person
      with all properties including
         firstName!
         lastName!
      excluding
         taxpayerID
      referenceEmbed > addresses
```

Parameters

Name	Type	Description
request	keyword	Identifies the message as a request. A Method MAY define at most one request.
response	keyword	Identifies the message as a response. A Method MAY define any number of responses.
this	keyword	A special case of a resource-defined message payload, where the payload is a representation of the object instance bound to the resource at runtime. Whereas a message specifying the name of the containing resource, but not specifying **this**, could represent any runtime instance of the resource; using **this** asserts that the message represents the same object instance as the resource on which the Method is invoked. When **this** is specified, the <resource-realization> parameter MAY be specified for clarity, but is optional. If <resource-realization> is specified following **this**, it MUST be a valid qualified or unqualified reference to the containing resource.
<resource-realization>	QName(Collection Resource or Object Resource)	Identifies a resource that defines the message payload model. The runtime message payload will contain a representation of that resource. More precisely, it will contain an instance of the Data Structure bound to the resource, and conforming to the realization specified (explicitly or implicitly) in the resource.
<data-structure>	QName(Data Structure)	Identifies a Data Structure that defines the message payload model. The runtime message payload will contain an instance of the Data Structure, conforming to the realization specified (explicitly or implicitly) in the message definition.
<response-code>	integer	A valid HTTP response code, as defined in RFC 2616, section 10. <response-code> MAY be

		specified only for response messages, not for requests.
<media-types>	List(SName(Built-in Media Type)or QName(User-defined Media Type))	A delimited list of references to media types allowed in this message. Overrides media types specified at the resource level.

Child Elements

Name	Topic	Description
<property-set>	Property Set	An optional Property Set, used to customize the realization of the specified <data-structure>.
<reference-link>	Reference Link	An optional Reference Link, used to realize a reference in the <data-structure> as a hyperlink.
<reference-embed>	Reference Embed	An optional Reference Embed, used to realize a reference in the <data-structure> as an embedded object..
<message-param>	Message Parameter	Defines a parameter carried in the header or query string, separate from the message payload.
<example>	Example	Specifies an in-line example of the message, for documentation and testing.
<external-example>	Example	References an externally defined example of the message, for documentation and testing.

Parent Elements

- Method

0 Comments

Message Parameter

0 Comments

The **parameter** element defines a parameter that may be passed as part of a Request (header or query) or Response (header only).

Syntax

[required] param <parameter-name> (([~of] type <data-type>) | ([~bound [~to]] <property-name>)) [[~located] in (header | query)]

Examples

Request

```
// roleCode in the header will be treated like an integer
required param roleCode of type int located in header
// roleName in the header will be treated like a string
required param roleName of type string located in header
// searchNameLike in the query will be treated like a string
param searchNameLike of type string located in query
// id in the query will be bound to the taxpayerID of the Data Structure bound to the parent resource
param id bound to property taxpayerID
```

Response

```
// roleCode in the header will be treated like an integer
required param roleCode of type int in header
// roleName in the header will be treated like a string
required param roleName of type string in header
```

Parameters

Name	Type	Description
required	keyword	Indicates that this parameter MUST be specified in a valid request.
<parameter-name>	Name	The name assigned to this parameter.
<data-type>	SName(Built-in Primitive Type)	The data type of the parameter. MUST be one of the built-in types specified in Built-in Primitive Data Types.
<property-name>	SName(Primitive Property)	The name of a Primitive Property defined in the Data Structure to which the containing message is bound. This property binding indicates that the Message Parameter is used as an identifier, filter, or other data-bound operator, associated directly with the bound property. It also implies that the parameter value SHOULD match the data type of the bound property.
header	keyword	Indicates that the parameter is passed in the header

		of the request or response message. For response message parameters, header is the default location, and the only allowable parameter location. However, the **header** keyword MAY still be included for clarity in response parameters.
query	keyword	Indicates that the parameter is passed in the query string of the request. Only request message parameters MAY use the **query** keyword. For request message parameters, query is the default location. However, the **query** keyword MAY still be included for clarity in request parameters.

Child Elements

None

Parent Elements

- Request
- Response

0 Comments

Security Scheme Application

0 Comments

The **secured** element applies one or more Security Schemes to control access to a Method, an Object Resource or Collection Resource, or an entire Resource API.

Syntax

```
secured [~by]
   (<security-scheme>
      [[~authorized [~for]] scopes <scopes>])...
```

Examples

```
secured by
   auth.Basic
```

```
secured by
   QueryStringKeyAuth
   auth.OAuth2
      authorized for scopes
         admin, manager
```

Parameters

Name	Type	Description
<security-scheme>	QName(Security Scheme Definition)	A reference to a defined Security Scheme Definition that controls access to the Method(s) defined within the containing element.
<scopes>	List(Scope)	A delimited list of scopes authorized to use the Method(s) defined within the containing element. Each scope named in this list MUST be defined in the referenced <security-scheme>.

Child Elements

None.

Parent Elements

- Resource API
- Collection Resource
- Object Resource
- Method

Discussion

Security Schemes are applied with the following precedence rules:

1. A Security Scheme Application on a Method takes precedence for that Method, overriding any Security Scheme Application on the containing resource or Resource API.
2. A Security Scheme Application on a resource applies to all methods defined in that resource, except where overridden at the Method level.
3. A Security Scheme Application on a Resource API applies to all methods defined on all resources within that API, except where overridden at the resource or Method level.

0 Comments

Security Schemes Library

0 Comments

Defines a reusable library of Security Schemes, which may be imported and applied to control access to the methods defined in an API.

Syntax

securitySchemesLibrary <library-name>
    <security-scheme>...

Examples

```
securitySchemesLibrary TaxBlasterAuthSchemes
   securityScheme Basic
      ...
   securityScheme OAuth2
      ...
```

Parameters

Name	Type	Description
<library-name>	Name	Name assigned to this security scheme library.

Child Elements

Name	Topic	Description
<security-scheme>	Security Scheme Definition	A definition of a security scheme.

Parent Elements

- RAPID Model

0 Comments

Security Scheme Definition

The **securityScheme** keyword defines and configures a security scheme, which may be applied to control access to the methods defined in an API.

Syntax

```
securityScheme <scheme-name>
    type (basic | oauth2 | custom)
        [methodInvocation
            [requires authorization
                <method-auth-param>...]
            [errorResponse statusCode <error-status-code>]...]
        [[~defines] scopes
            <auth-scope>...]
        [[~uses] flow <auth-flow>...]
        [settings
            (<setting-name> : "<setting-value>")...]
```

Examples

```
/** HTTP Basic authentication.
   https://www.ietf.org/rfc/rfc2617.txt */
securityScheme Basic
   type basic
   methodInvocation
      requires authorization
         /** userid and password, separated by a single colon (":") character, within a
         base64 [7] encoded string in the credentials.*/
         param basic_credentials type base64Binary in header
      errorResponse statusCode 401 //Unauthorized
```

```
/** OAuth2 authentication.
   https://tools.ietf.org/rfc/rfc6749.txt */
securityScheme OAuth2
   type oauth2
   methodInvocation
      requires authorization
         param token type string in header
         param access_token type string in query
      errorResponse statusCode 401
   defines scopes
      /** System administator. */
      admin
      /** Read-only user */
      user
      /** Manager of the system */
      manager
   settings
      authorization_url : "http://test.com/oauth/authorize"
      request_token_url : "http://test.com/oauth/oauth/request-token"
```

Parameters

Name	Type	Description
<scheme-name>	Name	Name assigned to this Security Scheme Definition.
basic	keyword	Indicates that this scheme uses Basic Authentication, as defined in IETF RFC 2617.
oauth2	keyword	Indicates that this scheme uses the OAuth 2.0 Authorization Framework defined in IETF RFC 6749.
custom	keyword	Indicates that this scheme uses a custom authentication/authorization framework. RAPID-ML can accommodate custom security schemes, to the extent that they can be configured using the provided security scheme parameters.
<error-status-code>	integer	A status code which MAY be returned from the secured Method in the event of an authorization failure.
<auth-scope>	Name	Defines a named scope used within this security scheme. For security schemes that define authorization scopes, the Security Scheme Application SHOULD specify the scope(s) that apply to the applicable Method. In the OAuth example above, some methods may be available only to users authenticated at the manager or admin level. OAuth 2.0 defines scopes as part of the protocol. Custom security schemes MAY also use scopes. While scopes MAY also be specified for a Basic security scheme, they have no meaning in that context.
<auth-flow>	SName	Specifies an OAuth 2.0 defined flow, which MUST be one of the following values: ACCESS_CODE, APPLICATION, IMPLICIT, or PASSWORD. This parameter is specific to OAuth 2.0. However, it MAY be specified for any security scheme, and may be ascribed special meaning for custom security schemes.
<setting-name>	Name	The name assigned to a setting, used to configure the security scheme. Any legal name is permissible, but the following OAuth 2.0 settings are pre-defined, and SHOULD be offered as available setting names by RAPID-ML editors: • `access_token_url` • `authorization_url` • `redirect_url` • `request_token_url` • `token_url`
<setting-value>	string	The value assigned to a setting.

Child Elements

Name	Topic	Description
<method-auth-param>	Message Parameter	Specifies a Message Parameter that must be passed in the secured Method request, to authenticate the client or

		to authorize access to the Method. As with standard message parameters, the Method authorization parameter may specify a data type and location (in the header or query string). However, Method authorization parameters may not be bound to a property of a Data Structure.

Parent Elements

- Security Schemes Library

0 Comments

Primitive Types Library

0 Comments

Defines a reusable library of Primitive Types, which may be imported and used as data types of primitive properties and user-defined simple types.

Syntax

primitiveTypesLibrary <library-name>
  <primitive-type>...

Examples

```
primitiveTypesLibrary QuantumTypes
   primitiveType principal_quantum_number
   primitiveType spin_state
   ...
```

Parameters

Name	Type	Description
<library-name>	Name	Name assigned to this primitive types library.

Child Elements

Name	Topic	Description
<primitive-type>	Primitive Type	Definition of a primitive type.

Parent Elements

- RAPID Model

Discussion

NOTE: Primitive types are essential for code generation and tools interoperability. For completeness and for low-level extensibility, this topic describes the mechanism for defining primitive types. But user-defined Simple Types are the preferred way of creating custom data types. Defining custom primitive types is not recommended at this time.

0 Comments

Primitive Type Definition

0 Comments

Defines a Primitive Type, which may be used as the data type of primitive properties and user-defined simple types.

Syntax

primitiveType <primitive-type-name>

Examples

```
primitiveType principal_quantum_number
primitiveType spin_state
```

Parameters

Name	Type	Description
<primitive-type-name>	Name	Name assigned to this Primitive Type

Child Elements

None.

Parent Elements

- Primitive Types Library

Discussion

NOTE: Primitive types are essential for code generation and tools interoperability. For completeness and for low-level extensibility, this topic describes the mechanism for defining primitive types. But user-defined Simple Types are the preferred way of creating custom data types. Defining custom primitive types is not recommended at this time.

0 Comments

Media Types Library

0 Comments

Defines a reusable library of Media Types, which may be imported and referenced as an expected or supported message format.

Syntax

mediaTypesLibrary <library-name>
  <media-type>...

Examples

```
mediaTypesLibrary FieldStudyMediaTypes
   mediaType application/observation
      specURL "http://lifescience-society.org/registry/observation.txt"
   mediaType application/subject+json
      specURL "http://lifescience-society.org/registry/subject-json.txt"
      derivedFrom (application/json)
```

Parameters

Name	Type	Description
<library-name>	Name	Name assigned to this media types library.

Child Elements

Name	Topic	Description
<media-type>	Media Type Definition	Definition of a media type.

Parent Elements

- RAPID Model

0 Comments

Media Type Definition

0 Comments

Defines a Media Type, which may be referenced as an expected or supported message format.

Syntax

mediaType <media-type-name>
    [specURL "<spec-url>"]
    [derivedFrom (<derived-from-media-types)])

Examples

```
mediaType application/observation
    specURL "http://lifescience-society.org/registry/observation.txt"
mediaType application/subject+json
    specURL "http://lifescience-society.org/registry/subject-json.txt"
    derivedFrom (application/json)
```

Parameters

Name	Type	Description
<media-type-name>	Name	Name assigned to this media type.
<spec-url>	URL	URL containing a specification of the media type.
<derived-from-media-types>	List(SName(Built-in Media Type) or QName(User-defined Media Type))	A delimited list of media type names, from which this media type is derived. Derivation means that the media type defined here is a specialization of the derived-from media type. The derived-from media type MUST be defined and accessible in the scope of the derived media type definition.

Child Elements

None.

Parent Elements

- Resource API

0 Comments

Link Relations Library

0 Comments

Defines a reusable library of Link Relations, which may be imported and applied to Reference Links.

Syntax

linkRelationsLibrary <library-name>
    <link-relation>...

Examples

```
linkRelationsLibrary MfgLinkRelations
   linkRelation inventory specURL "http://mrp.org/definitions/inventory.txt"
   linkRelation work-order specURL "http://mrp.org/definitions/work-order.txt"
```

Parameters

Name	Type	Description
<library-name>	Name	Name assigned to this link relations library.

Child Elements

Name	Topic	Description
<link-relation>	Link Relation Definition	A definition of a link relation.

Parent Elements

- RAPID Model

0 Comments

Link Relation Definition

0 Comments

The **linkRelation** element uses a named identifier, normally included in a registry, specifying the semantics of a Reference Link.

Syntax

linkRelation <link-relation-name> [specURL "<spec-URL>"]

Examples

```
linkRelation inventory specURL "http://mrp.org/definitions/inventory.txt"
linkRelation work-order specURL "http://mrp.org/definitions/work-order.txt"
```

Parameters

Name	Type	Description
<link-relation-name>	string	Name assigned to this link relation.
<spec-URL>	URI	Optional URL containing the specification for the Link Relation.

Child Elements

None.

Parent Elements

- Link Relations Library

0 Comments

Built-in Primitive Data Types

0 Comments

The following list contains the Primitive Property types that RAPID-ML™ supports. These are a sub-set of the built-in datatypes from XML Schema. Compliant tools MUST support each of these types.

Type	Description
NCName	See http://www.w3.org/TR/xmlschema-2/#NCName
QName	See http://www.w3.org/TR/xmlschema-2/#QName
anyURI	See http://www.w3.org/TR/xmlschema-2/#anyURI
base64Binary	See http://www.w3.org/TR/xmlschema-2/#base64Binary
boolean	See http://www.w3.org/TR/xmlschema-2/#boolean
date	See http://www.w3.org/TR/xmlschema-2/#date
dateTime	See http://www.w3.org/TR/xmlschema-2/#dateTime
decimal	See http://www.w3.org/TR/xmlschema-2/#decimal
double	See http://www.w3.org/TR/xmlschema-2/#double
duration	See http://www.w3.org/TR/xmlschema-2/#duration
float	See http://www.w3.org/TR/xmlschema-2/#float
gDay	See http://www.w3.org/TR/xmlschema-2/#gDay
gMonth	See http://www.w3.org/TR/xmlschema-2/#gMonth
gMonthDay	See http://www.w3.org/TR/xmlschema-2/#gMonthDay
gYear	See http://www.w3.org/TR/xmlschema-2/#gYear
int	See http://www.w3.org/TR/xmlschema-2/#int
integer	See http://www.w3.org/TR/xmlschema-2/#integer
long	See http://www.w3.org/TR/xmlschema-2/#long
string	See http://www.w3.org/TR/xmlschema-2/#string
time	See http://www.w3.org/TR/xmlschema-2/#time

0 Comments

Built-in Link Relations

0 Comments

RAPID-ML™ SHOULD support the full list of IANA Link Relations.

Type	Description
about	Refers to a resource that is the subject of the link's context. See http://tools.ietf.org/html/rfc6903.
alternate	Refers to a substitute for this context See http://www.w3.org/TR/html5/links.html#link-type-alternate.
appendix	Refers to an appendix. See http://www.w3.org/TR/1999/REC-html401-19991224.
archives	Refers to a collection of records, documents, or other materials of historical interest. See http://www.w3.org/TR/2011/WD-html5-20110113/links.html#rel-archives.
author	Refers to the context's author. See http://www.w3.org/TR/html5/links.html#link-type-author.
bookmark	Gives a permanent link to use for bookmarking purposes. See http://www.w3.org/TR/html5/links.html#link-type-bookmark.
canonical	Designates the preferred version of a resource (the IRI and its contents). See http://tools.ietf.org/html/rfc6596.
chapter	Refers to a chapter in a collection of resources. See http://www.w3.org/TR/1999/REC-html401-19991224.
collection	The target IRI points to a resource which represents the Collection Resource for the context IRI. See http://tools.ietf.org/html/rfc6573.
contents	Refers to a table of contents. See http://www.w3.org/TR/1999/REC-html401-19991224.
copyright	Refers to a copyright statement that applies to the link's context. See http://www.w3.org/TR/1999/REC-html401-19991224.
create-form	The target IRI points to a resource where a submission form can be obtained. See http://tools.ietf.org/html/rfc6861.
current	Refers to a resource containing the most recent item(s) in a collection of resources. See http://tools.ietf.org/html/rfc5005.
describedby	Refers to a resource providing information about the link's context. See http://www.w3.org/TR/powder-dr/#assoc-linking.
describes	The relationship A 'describes' B asserts that resource A provides a description of resource B. There are no constraints on the format or representation of either A or B, neither are there any further constraints on either resource. See http://tools.ietf.org/html/rfc6892.
disclosure	Refers to a list of patent disclosures made with respect to material for which 'disclosure' relation is specified. See http://tools.ietf.org/html/rfc6579.
duplicate	Refers to a resource whose available representations are byte-for-byte identical with the corresponding representations of the context IRI. See

	http://tools.ietf.org/html/rfc6249.
edit	Refers to a resource that can be used to edit the link's context. See http://tools.ietf.org/html/rfc5023.
edit-form	The target IRI points to a resource where a submission form for editing associated resource can be obtained. See http://tools.ietf.org/html/rfc6861.
edit-media	Refers to a resource that can be used to edit media associated with the link's context. See http://tools.ietf.org/html/rfc5023.
enclosure	Identifies a related resource that is potentially large and might require special handling. See http://tools.ietf.org/html/rfc4287.
first	An IRI that refers to the furthest preceding resource in a series of resources. See http://tools.ietf.org/html/rfc5988.
glossary	Refers to a glossary of terms. See http://www.w3.org/TR/1999/REC-html401-19991224.
help	Refers to context-sensitive help. See http://www.w3.org/TR/html5/links.html#link-type-help.
hosts	Refers to a resource hosted by the server indicated by the link context. See http://tools.ietf.org/html/rfc6690.
hub	Refers to a hub that enables registration for notification of updates to the context. See http://pubsubhubbub.googlecode.com.
icon	Refers to an icon representing the link's context. See http://www.w3.org/TR/html5/links.html#link-type-icon.
index	Refers to an index. See http://www.w3.org/TR/1999/REC-html401-19991224.
item	The target IRI points to a resource that is a member of the collection represented by the context IRI. See http://tools.ietf.org/html/rfc6573.
last	An IRI that refers to the furthest following resource in a series of resources. See http://tools.ietf.org/html/rfc5988.
latest-version	Points to a resource containing the latest (e.g., current) version of the context. See http://tools.ietf.org/html/rfc5829.
license	Refers to a license associated with this context. See http://tools.ietf.org/html/rfc4946.
lrdd	Refers to further information about the link's context, expressed as a LRDD ("Link-based Resource Descriptor Document") resource. See [RFC6415] for information about processing this relation type in host-meta documents. When used elsewhere, it refers to additional links and other metadata. Multiple instances indicate additional LRDD resources. LRDD resources MUST have an "application/xrd+xml" representation, and MAY have others. See http://tools.ietf.org/html/rfc6415.
memento	The Target IRI points to a Memento, a fixed resource that will not change state anymore. See http://tools.ietf.org/html/rfc7089.
monitor	Refers to a resource that can be used to monitor changes in an HTTP resource. See http://tools.ietf.org/html/rfc5989.
monitor-group	Refers to a resource that can be used to monitor changes in a specified group of HTTP resources. See http://tools.ietf.org/html/rfc5989.

next	Indicates that the link's context is a part of a series, and that the next in the series is the link target. See http://www.w3.org/TR/html5/links.html#link-type-next.
next-archive	Refers to the immediately following archive resource. See http://tools.ietf.org/html/rfc5005.
nofollow	Indicates that the context's original author or publisher does not endorse the link target. See http://www.w3.org/TR/html5/links.html#link-type-nofollow.
noreferrer	Indicates that no referrer information is to be leaked when following the link. See http://www.w3.org/TR/html5/links.html#link-type-noreferrer.
original	The Target IRI points to an Original Resource. See http://tools.ietf.org/html/rfc7089.
payment	Indicates a resource where payment is accepted. See http://tools.ietf.org/html/rfc5988.
predecessor-version	Points to a resource containing the predecessor version in the version history. See http://tools.ietf.org/html/rfc5829.
prefetch	Indicates that the link target should be preemptively cached. See http://www.w3.org/TR/html5/links.html#link-type-prefetch.
prev	Indicates that the link's context is a part of a series, and that the previous in the series is the link target. See http://www.w3.org/TR/html5/links.html#link-type-prev.
preview	Refers to a resource that provides a preview of the link's context. See http://tools.ietf.org/html/rfc6903.
previous	Refers to the previous resource in an ordered series of resources. Synonym for "prev". See http://www.w3.org/TR/1999/REC-html401-19991224.
prev-archive	Refers to the immediately preceding archive resource. See http://tools.ietf.org/html/rfc5005.
privacy-policy	Refers to a privacy policy associated with the link's context. See http://tools.ietf.org/html/rfc6903.
profile	Identifying that a resource representation conforms to a certain profile, without affecting the non-profile semantics of the resource representation. See http://tools.ietf.org/html/rfc6906.
related	Identifies a related resource. See http://tools.ietf.org/html/rfc4287.
replies	Identifies a resource that is a reply to the context of the link. See http://tools.ietf.org/html/rfc4685.
search	Refers to a resource that can be used to search through the link's context and related resources. See http://www.opensearch.org/Specifications/OpenSearch/1.1.
section	Refers to a section in a collection of resources. See http://www.w3.org/TR/1999/REC-html401-19991224.
self	Conveys an identifier for the link's context. See http://tools.ietf.org/html/rfc4287.
service	Indicates a URI that can be used to retrieve a service document. See

	http://tools.ietf.org/html/rfc5023.
start	Refers to the first resource in a collection of resources. See http://www.w3.org/TR/1999/REC-html401-19991224.
stylesheet	Refers to a stylesheet. See http://www.w3.org/TR/html5/links.html#link-type-stylesheet.
subsection	Refers to a resource serving as a subsection in a collection of resources. See http://www.w3.org/TR/1999/REC-html401-19991224.
successor-version	Points to a resource containing the successor version in the version history. See http://tools.ietf.org/html/rfc5829.
tag	Gives a tag (identified by the given address) that applies to the current document. See http://www.w3.org/TR/html5/links.html#link-type-tag.
terms-of-service	Refers to the terms of service associated with the link's context. See http://tools.ietf.org/html/rfc6903.
timegate	The Target IRI points to a TimeGate for an Original Resource. See http://tools.ietf.org/html/rfc7089.
timemap	The Target IRI points to a TimeMap for an Original Resource. See http://tools.ietf.org/html/rfc7089.
^type	Refers to a resource identifying the abstract semantic type of which the link's context is considered to be an instance. See http://tools.ietf.org/html/rfc6903.
up	Refers to a parent document in a hierarchy of documents. See http://tools.ietf.org/html/rfc5988.
version-history	Points to a resource containing the version history for the context. See http://tools.ietf.org/html/rfc5829.
via	Identifies a resource that is the source of the information in the link's context. See http://tools.ietf.org/html/rfc4287.
working-copy	Points to a working copy for this resource. See http://tools.ietf.org/html/rfc5829.
working-copy-of	Points to the versioned resource from which this working copy was obtained. See http://tools.ietf.org/html/rfc5829.

0 Comments

Built-in Media Types

0 Comments

RAPID-ML™ SHOULD support the following list of media types which are a subset from the IANA registry of media types:

Type	Spec URL	Derived Fr
application/*	http://www.iana.org/assignments/media-types/application	
application/atom+xml	http://www.rfc-editor.org/rfc/rfc4287.txt	application/xml
application/atomdeleted+xml	http://www.rfc-editor.org/rfc/rfc6721.txt	application/ator
application/atomcat+xml	http://www.rfc-editor.org/rfc/rfc5023.txt	application/xml
application/atomsvc+xml	http://www.rfc-editor.org/rfc/rfc5023.txt	application/xml
application/auth-policy+xml	http://www.rfc-editor.org/rfc/rfc4745.txt	application/xml
application/calendar+xml	http://www.rfc-editor.org/rfc/rfc6321.txt	application/xml
application/ecmascript	http://www.rfc-editor.org/rfc/rfc4329.txt	
application/^example	http://www.rfc-editor.org/rfc/rfc4735.txt	
application/EDI-X12		
application/EDIFACT		
application/fastinfoset	http://www.iana.org/assignments/media-types/application/fastinfoset	
application/font-woff		
application/gzip	http://www.ietf.org/rfc/rfc6713.txt	
application/javascript	http://www.rfc-editor.org/rfc/rfc4329.txt	

application/json	http://www.ietf.org/rfc/rfc4627.txt	application/ecm application/java
application/json-patch+json	http://www.rfc-editor.org/rfc/rfc6902.txt	application/json
application/mp4	http://www.rfc-editor.org/rfc/rfc4337.txt	
application/octet-stream	http://www.rfc-editor.org/rfc/rfc2045.txt	text/plain
application/ogg	http://www.rfc-editor.org/rfc/rfc5334.txt	
application/pdf	http://www.rfc-editor.org/rfc/rfc3778.txt	
application/postscript	http://www.rfc-editor.org/rfc/rfc2046.txt	
application/rdf+xml	http://www.ietf.org/rfc/rfc3870.txt	application/xml
application/rss+xml		application/xml
application/soap+fastinfoset	http://www.iana.org/assignments/media-types/application/soap+fastinfoset	application/fasti
application/soap+xml	http://www.rfc-editor.org/rfc/rfc3902.txt	application/xml
application/vnd.hal+json	http://www.iana.org/assignments/media-types/application/vnd.hal+json	application/json
application/vnd.hal+xml	http://www.iana.org/assignments/media-types/application/vnd.hal+xml	application/xml
application/vnd.osgi.bundle	http://www.iana.org/assignments/media-	application/zip

	types/application/vnd.osgi.bundle	
application/vnd.xmi+xml	http://www.iana.org/assignments/media-types/application/vnd.xmi+xml	application/xml
application/x-www-form-urlencoded		
application/xhtml+xml	http://www.rfc-editor.org/rfc/rfc3236.txt	application/xml
application/xml	http://www.rfc-editor.org/rfc/rfc3023.txt	
application/xml-dtd	http://www.rfc-editor.org/rfc/rfc3023.txt	
application/xop+xml		application/xml
application/zip	http://www.iana.org/assignments/media-types/application/zip	application/octe stream
multipart/form-data		
text	http://www.iana.org/assignments/media-types/text	
text/calendar	http://www.rfc-editor.org/rfc/rfc5545.txt	text/plain
text/cmd		
text/css	http://www.rfc-editor.org/rfc/rfc2318.txt	text
text/csv		
text/ecmascript	http://www.rfc-editor.org/rfc/rfc4329.txt	text
text/html	http://www.iana.org/assignments/media-types/text/html	text/plain
text/javascript		

text/plain	http://www.rfc-editor.org/rfc/rfc2046.txt	text
text/vcard		
text/xml	http://www.rfc-editor.org/rfc/rfc3023.txt	text/plain

0 Comments