



## RIF Combination with XML data

W3C Editor's Draft 29 September 2009

**This version:**

<http://www.w3.org/2005/rules/wg/draft/ED-rif-xml-data-20090929/>

**Latest editor's draft:**

<http://www.w3.org/2005/rules/wg/draft/rif-xml-data/>

**Previous version:**

<http://www.w3.org/2005/rules/wg/draft/ED-rif-xml-data-20090904/> (color-coded diff)

**Editors:**

Christian de Sainte Marie, IBM

This document is also available in these non-normative formats: [PDF version](#).

---

Copyright © 2009 W3C<sup>®</sup> (MIT, ERCIM, Keio), All Rights Reserved. W3C [liability](#), [trademark](#) and [document use](#) rules apply.

---

### Abstract

This document, developed by the [Rule Interchange Format \(RIF\) Working Group](#), specifies how a RIF document can be combined with XML data sources.

### Status of this Document

#### May Be Superseded

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index](#) at <http://www.w3.org/TR/>.*

#### Set of Documents

This document is being published as one of a set of 10 documents:

1. [RIF Overview](#)

2. [RIF Core Dialect](#)
3. [RIF Basic Logic Dialect](#)
4. [RIF Framework for Logic Dialects](#)
5. [RIF RDF and OWL Compatibility](#)
6. [RIF Datatypes and Built-Ins 1.0](#)
7. [RIF Production Rule Dialect](#)
8. [RIF Test Cases](#)
9. [RIF Combination with XML data](#) (this document)
10. [OWL 2 RL in RIF](#)

### First Public Working Draft

This new draft specifies how RIF rules can be used with XML data. Although conceptually similar to [RIF RDF and OWL Compatibility](#), and serving a parallel function, it is being developed later in the life of the Working Group and is not expect to reach Recommendation at the same time as the other Rec-Track RIF documents.

### Please Comment By 27 October 2009

The [Rule Interchange Format \(RIF\) Working Group](#) seeks public feedback on this First Public Working Draft. Please send your comments to [public-rif-comments@w3.org](mailto:public-rif-comments@w3.org) ([public archive](#)). If possible, please offer specific changes to the text that would address your concern. You may also wish to check the [Wiki Version](#) of this document and see if the relevant text has already been updated.

### No Endorsement

*Publication as a Editor's Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.*

### Patents

*This document was produced by a group operating under the [5 February 2004 W3C Patent Policy](#). The group does not expect this document to become a W3C Recommendation. W3C maintains a [public list of any patent disclosures](#) made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains [Essential Claim\(s\)](#) must disclose the information in accordance with [section 6 of the W3C Patent Policy](#).*

---

## Table of Contents

- [1 Overview](#)
- [2 Importing XML data sources and schemas in RIF](#)
- [3 A simple data model for XML documents](#)
  - [3.1 Definitions](#)
  - [3.2 Element information items](#)
  - [3.3 Attribute information items](#)
  - [3.4 Character information items](#)
  - [3.5 Resolution of references](#)
  - [3.6 Example of a data model instance](#)
- [4 RIF as an implementation of the data model](#)
  - [4.1 Constants](#)
    - [4.1.1 DM-Names](#)
    - [4.1.2 Class names](#)
    - [4.1.3 Slot names](#)
  - [4.2 Variables](#)
  - [4.3 Class membership: rif:Member](#)
  - [4.4 Sub-class relationship: rif:Subclass](#)
  - [4.5 Frames: rif:Frame](#)
  - [4.6 Atoms: rif:Atom](#)
- [5 Conformance](#)
- [6 The special case of RDF and OWL data sources](#)
- [7 References](#)
- [8 Appendix A: Glossary \(non-normative\)](#)
- [9 Appendix B: Extract from the XQuery 1.0 and XPath 2.0 Data Model \(non-normative\)](#)
  - [9.1 Element and attribute node type names \(from \[XDM\]\)](#)
  - [9.2 Typed value determination \(from \[XDM\]\)](#)
- [10 Appendix C: Embedding imported data sources as RIF facts \(non-normative\)](#)

## Overview

This document specifies how a RIF document can be combined with XML data sources. It defines a data model for XML instance documents, that is a simplified version of the XQuery 1.0 and XPath 2.0 Data Model [XDM], and it specifies how the RIF condition language can be interpreted with respect to an XML data source, and what is the associated semantics, in accordance with that data model.

The [XQuery 1.0 and XPath 2.0 Data Model](#) (XDM) specifies what information is accessible in a collection of XML documents, but it does not specify the language used to represent or access the data: this document specifies an implementation, using the RIF condition language, of a simplified version of the XDM. This makes the RIF condition language comparable to other implementations of the XDM, such as [XPath 2.0](#) and [XQuery 1.0](#).

Essentially, this document specifies:

- how XML data sources, possibly associated with XML schemas, are imported in a RIF document;
- how sets of elements with given name or type properties, in an XML instance document, can be represented in RIF; and
- how the relation between parent elements and children elements and attributes, in an XML instance document, can be represented in RIF, when the children have given name and type properties.

Like the [XQuery 1.0 and XPath 2.0 Data Model](#), the simplified version used in this document supports the following classes of XML documents:

- Well-formed documents conforming to [Namespaces in XML] or [Namespaces in XML 1.1].
- DTD-valid documents conforming to [Namespaces in XML] or [Namespaces in XML 1.1], and
- W3C XML Schema-validated documents.

Accordingly, this document specifies how a RIF document is combined with well-formed XML data sources and, where available, the corresponding XML schemas, identified using the `rif:Import` construct.

However, an instance of the data model can also be constructed from non-XML sources such as relational tables in a database or object instances. In this case, the data is represented, in RIF, in conformance with the serialization of the source according to an XML schema that **MUST** be imported in the RIF document.

An XML schema can also be combined with a RIF document without the associated data source being specified: in that case, the selection of the data source to be combined with the RIF document is left to the RIF document consumer. This provides a way to communicate the data model that is intended, in a RIF document, for the data source, without specifying an actual data source.

**Editor's Note:** This section will be completed, e.g. with typical usage scenarios, in a future draft.

## Importing XML data sources and schemas in RIF

In RIF, the `Import` directive is used to communicate the location of an external document to be combined with the RIF document that contains the directive and, optionally, a profile that governs the combination.

In [\[RIF-Core\]](#), [\[RIF-PRD\]](#) and [\[RIF-BLD\]](#), the use of the `Import` directive is limited to identifying an imported RIF document, or an RDF graph or an OWL ontology to

be combined with a RIF document. An optional profile that governs the combination of a RIF document with an RDF graph or an OWL ontology can also be provided, as specified in [\[RIF-RDF-OWL\]](#).

This specification extends the `Import` directive in two ways:

1. a new allowed value and a new set of allowable values are specified for the profile of an import, in addition to the values specified in [\[RIF-RDF-OWL\]](#):
  - the new value is the URI: `rif:xml-data`;
  - in addition, any URI that identifies an XML schema document is allowed as a profile.
2. the `location` sub-element, that contains the IRI that identifies the imported data document, is made optional as well.

The [BNF-style pseudo-schema](#) for the modified syntax is as follows:

```
<Import>
  <location> xs:anyURI </location>?
  <profile> xs:anyURI </profile>?
</Import>
```

The following constraints must be satisfied:

- Although all the sub-elements are optional, an `Import` directive must contain at least a data document `location` or a `profile`;
- If an `Import` directive contains no data document `location`, the `profile` must identify an XML Schema;
- When an `Import` directive contains both a data document `location` and a `profile`, if the `profile` identifies an XML Schema and if the data document provides a `schema-location`, the `profile` must identify the same XML schema as the `schema-location` URI;
- If the `profile` is `rif:xml-data`, the `location` URI must be present and it must identify an XML instance document.

This specification does not prescribe the behaviour of a conformant implementation when one of the above constraint is not satisfied.

This specification does not prescribe the behaviour of a conformant implementation when an `Import` directive contains a `profile` that is neither `rif:xml-data`, nor an URI that identifies an XML schema.

## A simple data model for XML documents

The data model described in this section specifies the information that is accessible in an XML document, possibly in combination with an XML schema, and that is

used to specify the interpretation of the RIF condition language with respect to an XML instance document.

The data model is a stripped down version of the XQuery 1.0 and XPath 2.0 Data Model [XDM]. As a consequence, the RIF condition language can be considered a partial implementation of the XQuery 1.0 and XPath 2.0 Data Model, and the interpretation of the RIF condition language with respect to XML documents can be specified in terms of the XQuery 1.0 and XPath 2.0 Data Model or its implementations, such as XPath 2.0 [XPath 2.0]. This document will reuse definitions from the [XQuery 1.0 and XPath 2.0 Data Model](#) and the [XPath 2.0](#) specifications, as appropriate, and otherwise provide pointers and examples where relevant.

## Definitions

The data model specifies the *information items* from the XML infoset [Infoset] and from the post-schema validation infoset (PSVI), or derived from the infoset and the PSVI, that are required to interpret some RIF constructs with respect to an XML instance document.

**Definition (Information item).** (from [Infoset]) An *information item* is an abstract description of some part of an XML document: each information item has a set of associated named properties. □

In this document, an information item is said to be *constructed from an infoset*, if the data item that it describes is contained in a data source that is not associated with an XML schema when it is imported in RIF. If the data source is associated with an XML schema, all the information items used to describe the content of that data source are said to be *constructed from a PSVI*.

If an information item is constructed from an infoset, all general and external parsed entities must be fully expanded before the data model is constructed.

In this specification, the property names are shown in square brackets, **[thus]**.

The data model relies on three types of information items:

- [Element information items](#), that describe the elements in an XML instance document. The properties associated to each element information item are: [namespace name], [local name], [children], [root], [attributes], [type name], [string value], [typed value], [is-id], [is-idrefs];
- [Attribute information items](#), that describe the attributes of elements. The properties associated to the attribute information item are: [namespace name], [local name], [attribute type], [owner element], [type name], [string value], [typed value], [is-id], [is-idrefs];
- [Character information items](#), that describe the data characters that appear in the XML document. The character information item has two properties: [character code] and [element content whitespace].

Given a data source, the relevant set of information items may be created by methods other than parsing and/or schema-validating an XML document. This specification does not describe or prescribe any method for retrieving the required information from a data source, possibly combined with an XML schema.

This specification distinguishes between the data model as a general concept and specific items (information items or atomic values) that are concrete examples of the data model. Some concrete examples are being particularly distinguished by being identified as *instances of the data model*.

**Definition (Instance of the data model).** An *instances of the data model* is a sequence of element information items, in document order. In particular, given an XML document *D*, the *instance of the data model that describes D* is the sequence of all the element information items that describe an element contained in *D*, in document order. □

When there is no ambiguity with respect to *D*, the instance of the data model that describes *D* will be called, simply: *the instance of the data model*.

**Definition (Atomic value).** An *atomic value* is a value in the value space of an atomic type and is labeled with the name of that atomic type. □

**Definition (Atomic type).** An *atomic type* is one of the 20 primitive simple types defined in [Section 3.3 Primitive Datatypes](#) of [\[XSD 1.1 Part 2\]](#) or a type derived by restriction from another atomic type. □

Types derived by list or union are not atomic.

**Definition (Sequence).** A *sequence* is an ordered collection of zero or more information items. □

A sequence cannot be a member of a sequence. An important characteristic of the data model is that there is no distinction between an item (a information item or an *atomic value*) and a singleton sequence containing that item. An item is equivalent to a singleton sequence containing that item and vice versa.

Except when specified otherwise, sequences are ordered according to the *document order*.

**Definition (Document order).** A *document order* is defined among all the element information items that describe a given XML instance document. Document order is a total ordering. Informally, document order is the order in which nodes appear in the XML serialization of a document. □

Within a tree, *document order* satisfies the following constraints:

1. The root node is the first node.
2. Every node occurs before all of its children and descendants.
3. The relative order of siblings is the order in which they occur in the children property of their parent node.

4. Children and descendants occur before following siblings.

XML element, attribute and type names are usually represented as XML qualified names, or QNames. However, `xs:QName` is not a RIF-Core built-in datatype. In the data model, all qualified names, including atomic values, are represented as *expanded QNames*.

**Definition (Expanded QName).** An *expanded QName* is a pair of two values, consisting of a possibly empty namespace URI and a local name. □

## Element information items

There is an *element information item* for each element appearing in the XML document. One of the element information items corresponds to the root of the element tree, and all other element information items are accessible by recursively following its [children] property.

An element information item has the following properties:

1. **[namespace name]** The namespace name, if any, of the element type. If the element does not belong to a namespace, this property has no value.
2. **[local name]** The local part of the element-type name. This does not include any namespace prefix or following colon;
3. **[children]** An ordered list of child information items, in document order. This list contains only *element information items* and *character information items*, one for each element and data character appearing immediately within the current element. It does not contain other kinds of information items, such as *attribute information items*, even if the XML element described by the information item has attributes. If the element is empty, this list has no members;
4. **[root]** The element information item that describes the root element in the XML document. The [root] property all the element information items that describe elements contained in the same XML instance document point to the same root element information item, including that root element information item itself. If the [root] property of an element information item points to that element information item itself, the [root] properties of all the element information items that are accessible by following its [children] property, recursively, must point to that same root element information item;
5. **[attributes]** An unordered set of attribute information items, one for each of the attributes of this element. This includes all of the "special" attributes (`xml:lang`, `xml:space`, `xsi:type`, etc.) but does not include namespace declarations (because they are not attributes). Default and fixed attributes, e.g. provided by XML Schema processing, are added to the [attributes]. If the element has no attributes, this set has no members;
6. **[type name]** The [type name] property of an element information item is empty if, and only if, the element information item is constructed from an infoset. If the element information item is constructed from a PSVI, the type name is represented by an expanded QName. It is determined as described in [Section 3.3.1.1. Element and Attribute Node Type Names](#)

- from [\[XDM\]](#) (reproduced in [APPENDIX TYPE NAME](#) for the reader's convenience);
7. **[string value]** The normalised representation of the content of the element as a string. The string value is calculated as follows:
    - If the element is empty or if it does not have any character information item children nor descendants, its string value is the zero length string;
    - Else, if the [type name] property of the element information item is empty, or if the element has a complex type with element-only content, or a complex type with mixed content, its string value is the string comprised of characters that correspond to the [character code] properties of each of the character information item children of the element and all its descendants, in document order. If the resulting string consists entirely of whitespace and the [element content whitespace] property of the character information items used to construct it are true, the string value is the zero-length string;
    - Else, if the element has a simple type or a complex type with simple content: its string value is the [schema normalized value](#) of the element;
    - Note that, if the element has a typed value, any valid lexical representation of the typed value can be used to determine the [string value] property;
  8. **[typed value]** The typed-value is calculated as follows:
    - If the element is empty or if it has element-only children, its typed value is the element itself; more precisely, if an element information item has no character information items in its [children] property, the value of its [typed value] property is the element information item itself. This is a deviation from the [XQuery 1.0 and XPath 2.0 Data Model](#);
    - Else, if the [type name] property of the element information item is empty, or if the element has a complex type with mixed content (including xs:anyType), its typed value is the same as its string value;
    - Otherwise, the element must have a simple type or a complex type with simple content. Its typed value is computed as described in [Section 3.3.1.2 Typed Value Determination](#), in [\[XDM\]](#) (reproduced in [Appendix B](#) for the reader's convenience). The result is a sequence of zero or more atomic values. The relationship between the values of the [type name], [typed value], and [string value] properties of an element information item is consistent with XML Schema validation. Note that in the case of `xs:QNames` and `xs:NOTATIONS`, the prefix is not preserved, and the typed values are represented as expanded QNames. This deviates from the [\[#ref-xdm\]XQuery 1.0 and XPath 2.0 Data Model](#);
  9. **[is-id]** If the [type name] property of an element information item is empty, or if the element has a complex type with element-only content, the [is-id] property is false. Else, if the typed value of the element consists of exactly

one atomic value, that value is of type `xs:ID`, or a type derived from `xs:ID`, the `[is-id]` property is true, otherwise it is false.

10. `[is-idrefs]` If the `[type name]` property of an element information item is empty, or if the element has a complex type with element-only content, the `[is-idrefs]` property is false. Else, if any of the atomic values in the typed-value of the element is of type `xs:IDREF` or `xs:IDREFS`, or a type derived from one of those types, the `[is-idrefs]` property is true, otherwise it is false.

## Attribute information items

There is an attribute information item for each attribute (specified or defaulted) of each element in the document, excluding those which are namespace declarations (because they are not attributes).

Attributes declared in the DTD with no default value and not specified in the element's start tag are not represented by attribute information items.

An attribute information item has the following properties:

1. **[namespace name]** The namespace name, if any, of the attribute. Otherwise, this property has no value;
2. **[local name]** The local part of the attribute name. This does not include any namespace prefix or following colon;
3. **[attribute type]** An indication of the type declared for this attribute in the DTD. The only values that are relevant to this specification are `ID`, `IDREF` and `IDREFS`. If there is no declaration for the attribute or if no declaration has been read, this property has no value. The value of this property is not affected by the validity of the attribute value;
4. **[owner element]** The element information item which contains this information item in its `[attributes]` property;
5. **[type name]** Empty if the attribute information item is constructed from an infoset. If the attribute information item is constructed from a PSVI, the type name is represented as an expanded QName, and it is determined as described in [Section 3.3.1.1. Element and Attribute Node Type Names](#) from [\[XDM\]](#) (reproduced in [Appendix B](#) for the reader's convenience);
6. **[string value]** The [schema normalized value](#) PSVI property if that exists; otherwise, the normalized attribute value according to [Section 3.3.3 Attribute-Value Normalization](#) in [\[XML\]](#)). Note that, if the attribute has a typed value, any valid lexical representation of the typed value can be used to determine the string value;
7. **[typed value]** The typed-value is calculated as follows:
  - If the `[type name]` property of an attribute information item is empty, its typed value is the same as its string value;
  - Otherwise, a sequence of zero or more atomic values as described in [Section 3.3.1.2 Typed Value Determination](#), in [\[XDM\]](#) (reproduced in [Appendix B](#) for the reader's convenience). The relationship between the values of the `[type name]`, `[typed`

- value], and [string value] properties of an attribute information item is consistent with XML Schema validation.
8. **[is-id]** If the attribute is named `xml:id` and its [attribute type] property does not have the value `ID` and its [type name] property does not have the value `xs:ID`, then [\[xml:id\]](#) processing is performed. This will assure that the value does have the type `ID` or `xs:ID` (if the attribute information item is constructed from an infoset or from a PSVI, respectively) and that it is properly normalized. The [is-id] property is always true for attributes named `xml:id`. Else, if the [attribute type] property has the value `ID`, or if the type name is `xs:ID` or a type derived from `xs:ID`, the [is-id] property is true; otherwise, it is false. This specification does not prescribe the behaviour of a RIF consumer application if an error is encountered during [\[xml:id\]](#) processing.
  9. **[is-idrefs]** True if the value of the [attribute type] property is `IDREF` or `IDREFS`, or if any of the atomic values in the typed-value of the attribute is of type `xs:IDREF` or `xs:IDREFS`, or a type derived from one of those types. Otherwise, false.

## Character information items

There is a character information item for each data character that appears in the document, whether literally, as a character reference, or within a CDATA section.

Each character is a logically separate information item, but applications are free to chunk characters into larger groups as necessary or desirable.

A character information item has the following properties:

1. **[character code]** The ISO 10646 character code (in the range 0 to `#x10FFFF`, though not every value in this range is a legal XML character code) of the character.
2. **[element content whitespace]** A boolean indicating whether the character is white space appearing within element content (see [\[XML\]](#), Section 2.10. White Space Handling). Note that validating XML processors are required to provide this information. If there is no declaration for the containing element, or if there are multiple declarations, or if no declaration has been read, this property has no value for white space characters. It is always false for characters that are not white space.

## Resolution of references

**Definition (Reference information item).** Given an information item, *I*, whose [is-id] is true, and given an atomic value, *id*, of type `ID`, `IDREF`, `xs:ID`, or `xs:IDREF`, that matches one of the atomic values in *I*'s [typed value] property, the **reference information item** identified by *id* is the element information item, *R*, such that

- its [root] property has the same value as the [root] property of *I*, if *I* is an element information item, or as the [root] property of the element

- information item pointed to by the [owner element] property of *I*, if *I* is an attribute information element;
- and
    - either the [is-id] property of *R* is true and its [typed value] property matches *id*;
    - or the [is-id] property of one of the attribute information items in *R*'s [attributes] property is true, and the [typed value] property of that attribute information item matches *id*. □

Note that the reference information item is always an *element*.

Where this specification states, with respect to a set of information items, that the *references are resolved*, the following processing is applied to every information item in the set whose [id-refs] property is true:

- If the information item is an element information item, *E*, and
  - if its typed value is a single atomic value of type `xs:IDREF` or `xs:IDREFS` or a type derived from one of those types, *E* itself is replaced with its [reference information item](#);
  - or, if its typed value is a sequence that contains more than one atomic values, each atomic value that is of type `xs:IDREF` or `xs:IDREFS` or a type derived from one of those types is replaced with the typed value of its [reference information item](#), after the references in that typed value have been resolved, if the [id-refs] property of the reference information item is true;
- otherwise, the information item must be an attribute information item, *A*, and every atomic value in the typed value of *A* is replaced with its [reference information item](#).

No property value is changed in any information item: the references are resolved only on a "need-to-resolve" basis, where the specification of [RIF as an implementation of the data model](#) requires them to be resolved.

This specification does not prescribe a behavior if an error is encountered during reference resolution processing (such as `IDREFS` without corresponding `IDs`, invalid or duplicate `ID`, etc).

### Example of a data model instance

Consider the following XML instance document, representing data about customers, that can be retrieved from the IRI: <http://example.org/customertable/customers.xml>:

```
<CustomerTable xmlns="http://example.org/customertable"
  xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <Customer xml:lang="en">
    <Name> John </Name>
    <Account> 111 </Account>
  </Customer>
```

```

    <Customer xml:lang="fr">
      <Name> Jane </Name>
      <Account> 222 </Account>
      <Id> 222 </Id>
    </Customer>
  </CustomerTable>

```

Consider, further, the following XML schema, available from <http://example.org/customertable/customers.xsd>:

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  targetNamespace="http://example.org/customertable">

  <xs:element name="Name" type="xs:string"/>
  <xs:element name="Account" type="xs:integer"/>
  <xs:element name="Id" type="xs:integer"/>

  <xs:element name="Customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="Account"/>
        <xs:element ref="Id" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    <xs:attribute ref="xml:lang"/>
  </xs:element>

  <xs:element name="CustomerTable">
    <xs:complexType>
      <xs:all>
        <xs:element ref="Customer" minOccurs="0"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

</xs:schema>

```

The instance of the data model that describes the XML document associated with the schema, is a sequence that contains the following eight element information items, in that order:

1. an element information item representing the `CustomerTable` element, with the following values for its properties
  - [namespace name]: `http://example.org/customertable`
  - [local name]: `CustomersTable`
  - [children]: a sequence of two element information items, which are the two `Customer` element information items in the data

- model instance, in the same order; that is, items 2 and 5 in the data model instance, in that order;
- [root]: the element information item itself;
  - [attributes]: empty;
  - [type name]: `http://example.org/customertable` *"some locally unique identifier for the anonymous type of the CustomerTable element"*
  - [string value]: "John 111 Jane 222 222"
  - [typed value]: the element information item itself;
  - [is-id]: false;
  - [is-idrefs]: false;
2. an element information item that describes the first `Customer` element in the document, with the following values for its properties
    - [namespace name]: `http://example.org/customertable`
    - [local name]: `Customer`
    - [children]: a sequence of two element information items, which are items 3 and 4 in the data model instance, describing the `Name` and `Account` sub-elements, respectively, in that same order;
    - [root]: same value as the [root] property of the previous element information item;
    - [attributes]: a attribute information item representing the `xml:lang` attribute. The attribute information item has the following values for its attributes:
      - [namespace name]: <http://www.w3.org/XML/1998/namespace>
      - [local name]: `lang`
      - [Attribute type]: empty;
      - [owner element]: the previous element information item;
      - [type name]: <http://www.w3.org/2001/XMLSchema> language
      - [string value]: "en"
      - [typed value]: `en`
      - [is-id]: false;
    - [type name]: `http://example.org/customertable` *"some locally unique identifier for the anonymous type Customer"*
    - [string value]: "John 111"
    - [typed value]: the element information item itself;
    - [is-id]: false;
    - [is-idrefs]: false;
  3. an element information item representing the `Name` sub-element of the first `Customer` element in the document, with the following values for its properties
    - [namespace name]: `http://example.org/customertable`
    - [local name]: `Name`
    - [children]: a sequence of four character information items, spelling *j o h n*, in that order;
    - [root]: same value as the [root] property of the previous element information item;
    - [attributes]: empty;

- [type name]: <http://www.w3.org/2001/XMLSchema> string
  - [string value]: "John"
  - [typed value]: "John"
  - [is-id]: false;
  - [is-idrefs]:false;
4. an element information item representing the `Account` sub-element of the first `Customer` element in the document, with the following values for its properties
    - [namespace name]: `http://example.org/customertable`
    - [local name]: `Account`
    - [children]: a sequence of three character information items, spelling `1 1 1`, in that order;
    - [root]: same value as the [root] property of the previous element information item;
    - [attributes]: empty;
    - [type name]: <http://www.w3.org/2001/XMLSchema> integer
    - [string value]: "111"
    - [typed value]: `111`
    - [is-id]: false;
    - [is-idrefs]:false;
  5. an element information item that describes the second `Customer` element in the document;
  6. an element information item representing the `Name` sub-element of the second `Customer` element in the document;
  7. an element information item representing the `Account` sub-element of the second `Customer` element in the document;
  8. an element information item representing the `Id` sub-element of the second `Customer` element in the document;

## RIF as an implementation of the data model

This section specifies, for some constructs of the RIF condition language, an interpretation with respect to the instances of the above data model that represent the imported data sources.

The specification provides, therefore, a means to combine RIF documents with XML data sources, with or without an associated XML schema, as well as a mechanism to associate with a RIF document, the data model that is assumed, in the interchanged rules, for the data sources, in the form of an XML schema (or a set of XML schemas).

### Constants

Any constant in a RIF document can be interpreted with respect to atomic values in a data model instance. Some RIF constant, those with a *DM-Name*, can also be interpreted with respect to the expanded QNames that represent element, attribute and type names in a data model instance.

## DM-Names

The [data model](#) relies on [expanded-QNames](#) to represent qualified names. However, [RIF-Core](#) has no built-in datatype for qualified names. In order to specify the interpretation of RIF constructs with respect to data model instances that describe imported data sources, we define a mapping, *DM-Names*, from `xs:string` constants to expanded QNames, and we use that mapping to define the *DM-Name* of constants.

**Definition (DM-Names).** *DM-Names* is a mapping from `xs:string` constants to [expanded QNames](#). An `xs:string` constant of the form `[URI '#']? NAME` is mapped to an expanded QName where the optional URI is the, possibly empty, namespace URI and `NAME` is the local name, if and only if

- the substring `URI`, if present, is an Universal Resource Identifier as defined in [\[RFC 3986\]](#) and extended in [\[RFC 3987\]](#) with a new name "IRI"; and
- the substring `NAME` is of the form:

```
NAME := [ xs:NCName | 'type(' xs:NCName ')' | 'list(' xs:NCName ')' | 'attr
```

For all the other `xs:string` constants, the mapping is undefined. □

**Definition (DM-Name).** Given a RIF constant, *c*, of any type that can be cast into an `xs:string`, *s*, for which the *DM-Names* mapping is defined, the *DM-Name* of *c* is the expanded QName onto which *DM-Names* maps *s*. RIF constants that cannot be cast into an `xs:string` for which the *DM-Names* mapping is defined do not have a *DM-Name*. □

By extension, given a constant, *c* for which a *DM-Name* is defined, we will write the *namespace URI of c* and the *local name of c* to refer to the namespace URI and local name components of *c*'s *DM-Name*, respectively.

### Example 4.1.

- The *DM-Name* of the `rif:iri` constant `<http://example.org/customertable#Customer>` is an expanded QName where the namespace URI is `http://example.org/customertable` and the local name is `Customer`;
- The *DM-Name* of the `xs:string` constant `<myElement>` is an expanded QName where the namespace URI is empty and the local name is `myElement`;
- The namespace URI of the `rif:iri` constant `<http://www.w3.org/2001/XMLSchema#type(date)>` is `http://www.w3.org/2001/XMLSchema`, and its local name is `type(date)`;
- The namespace URI of the `xs:string` constant `<attribute(myAttribute)>` is empty, and its local name is `attribute(myAttribute)`;

- The DM-Name of the `xs:string` constant "`http://example.org/customertable#Customer`" is an expanded QName where the namespace URI is `http://example.org/customertable` and the local name is `Customer`;
- The `rif:iri` constant `<http://example.org/>` has no DM-Name.

□

### Class names

Constants that occur in a position where the identifier of a class is expected, e.g. in RIF class membership and subclass relationship formulas, can be interpreted as identifying subsets of the element information items in the [instances of the data model](#) that describe the imported data sources.

Given a constant,  $C$ , for which the DM-Name is defined, and an [instance of the data model](#),  $I_{DM}$ , let us call:  $C$ -set, the set that contains an element information item,  $N \in I_{DM}$ , if and only if

- the local name in  $C$ 's DM-Name has the form `type(NAME)`, and,
  - either the namespace URI and `NAME` in the DM-Name of  $C$  matches the namespace URI and local name in  $N$ 's [type name] property, respectively;
  - or the type identified by the namespace URI and local name in  $N$ 's [type name] property is derived by restriction from the type identified by the namespace URI and `NAME` in  $C$ 's DM-Name;
- or the namespace URI and the local name in the DM-Name of  $C$  match the [namespace name] and the [local name] properties of  $N$ , respectively;
- or  $N$  has been constructed from a PSVI, and the element that it describes belongs to a substitution group, occurs in a position where the schema requires the head element, and the head element's namespace URI and local name match the namespace URI and the local name in the DM-Name of  $C$ , respectively.

This specification does not prescribe anything with respect to the interpretation as class identifiers, of constants for which the DM-Name is not defined.

Given an XML document,  $D$ , the sets of element information items selected by a constant  $C$  from the data model instance that describes  $D$  corresponds to the sequences of elements selected, in  $D$ , by the following XPath 2.0 expressions, where `prefix` is bound to `URI`:

1. `/descendant::element(*, [prefix:]NAME)` if  $C$  has form `[URI#]type(NAME)`;
2. or `/descendant::[prefix:]NAME` (or `/descendant::schema-element([prefix:]NAME)` if the namespace is that of an in-scope schema definition), if  $C$  has form `[URI#]NAME`.

**Example 4.2.** With respect to the data model instance that describes the XML document from [Section 3.6. Example of a data model instance](#), above,

- If *C* stands for the `rif:iri` constant `<<http://www.w3.org/2001/XMLSchema#type\(integer\)>>`, the *C-set* contains the element information items that represent the following elements,

```
<Account>111</Account>
<Account>222</Account>
<Id>222</Id>
```

- If *C* stands for the `rif:iri` constant `<<http://example.org/customertable#Name>>`, the *C-set* will contain the element information items that describe the following elements:

```
<Name>John</Name>
<Name>Jane</Name>
```

- In the schema-less case, that is, if the data model instance has been constructed from the infoset, without the XML schema being associated to the XML document,
  - in the first example, above, the *C-set* would be empty: the information about an element's type comes from the schema, and the [type name] properties would be empty for all the element information items in the data model instance;
  - in the second example, the *C-set* would be unchanged, because the [namespace name] and [local name] properties of an element information item depend on the XML instance element that it describes, only.

□

### Slot names

Constants that occur in a position where a slot's name is expected, e.g. in RIF frame formulas, can be interpreted as identifying subsets of an element information item's [children] or [attributes] properties, in the [instances of the data model](#) that describe the imported data sources.

Given a frame  $o$  [ *slot*  $\rightarrow$  *v* ], where *o* identifies an element information item in a data model instance, and where *slot* is a RIF constant for which the DM-Name is defined, let us call: *o/slot-sequence*, the sequence constructed as follows, [after references have been resolved](#):

- if the local name in *slot*'s DM-Name has the form `attribute(NAME)`, the *o/slot-sequence* is the, possibly empty, subset of the [attributes] property of the element information item identified by *o*, that contains only the attribute information items whose [namespace name] property

- matches the namespace URI of `slot`'s DM-Name, and whose [local name] property matches `NAME`;
- else, the *`o/slot-sequence`* is the possibly empty sub-sequence, in document order, of the [children] property of the element information item identified by `o`, that contains only
    - element information items whose [namespace name] property match the namespace URI in `slot`'s DM-Name, and whose [local name] property matches either the local name in `slot`'s DM-Name, or `NAME`, if the local name in `slot`'s DM-Name is of form `list(NAME)`;
    - and element information items constructed from a PSVI, and such that the element that they represent belongs to a substitution group, occur in a position where the schema requires the head element, and the head element's namespace URI matches the namespace URI in `slot`'s DM-Name and its name matches either the local name in `slot`'s DM-Name, or `NAME`, if the local name in `slot`'s DM-Name is of form `list(NAME)`.

**Editor's Note:** Shall we add a property that contains the expanded QNames of the relevant substitution groups' heads in the element information item part of the data model?

This specification does not prescribe anything with respect to the interpretation of slot names for which the DM-Name is not defined.

Given a XML instance element, *E*, the element information item that describes it in a data model instance, *e*, and a constant, *C*, for which the DM-Name is defined, the sequence of information items selected from *e* by a constant *C*, that is, the *e/C-sequence* as defined above, corresponds, in the absence of references, to the sequences of nodes selected by the following XPath 2.0 expressions, where *prefix*, if present, is bound to *URI*, and where the context element is *E*:

1. `attribute::[prefix:]NAME`, if the, optional, namespace URI in *C*'s DM-Name is *URI* and if the local name has the form `attribute(NAME)`;
2. `child::[prefix:]NAME`, resp. `child::schema-element([prefix:]NAME)`, if the, optional, namespace URI in *C*'s DM-Name is *URI* and if the local name is *NAME*.

**Example 4.3.** In the context of the data model instance that describes the XML document from [Section 3.6. Example of a data model instance](#), above, consider a RIF frame expression of the form `o[slot->v]`:

- if the object, `o`, is the element information item that describes the root `CustomerTable` element, and if `slot` stands for the `rif:iri` constant: `<http://example.org/customertable#Customer>`, the *`o/slot-sequence`* contains two ordered items, items 2 and 5 in the data model instance as described in [Section 3.6](#), that is, in document order:

1. the element information item that describes the first `Customer` element in the document;
  2. the element information item that describes the second `Customer` element;
- if `o` stands for the second one of these two element information items (the one about the `Customer` element that describes the customer named "Jane"), and
    - if `slot` stands for the `xs:string` constant: `"http://example.org/customertable#Name"`, the `o/slot-sequence` contains one single element information item from the data model instance, namely: the information item that describes the `Name` element that contains the string: "Jane" (item 6 in the data model instance represented in [Section 3.6](#));
  - if `slot` stands for the `rif:iri` constant: `<http://www.w3.org/XML/1998/namespace#attribute(lang)>`, the `o/slot-sequence` contains, from the `[attribute]` property of `o`, the attribute information item that describes the `xml:lang` attribute of the `Customer` element that represents the data about "Jane". Note that that information item is not part of the data model instance represented in [Section 3.6](#), since data model instances contain only element information items: attribute information items are, therefore, only accessible through the `[attributes]` properties of the element information items in a data model instance.

Notice that, in this example, the `o/slot-sequences` remain unchanged in the schema-less case introduced in [example 4.2](#): indeed, there are no substitution groups involved. □

## Variables

For the purpose of interpreting RIF constructs that occur in a RIF document,  $D$ , with respect to the imported data sources, the domain of all the variables declared in  $D$  must include, at least:

- the data model instances that describe all the statically imported data sources, that is, the data sources that are identified in the `location` part of an `Import` directive in  $D$ . The data model instances are constructed from the PSVI for the data sources that are associated with an XML schema in the `Import` directive, or from the infoset, otherwise;
- the data model instances that describe all the dynamically imported data source, in any; that is, data sources that are imported at runtime by the consumer application, without being identified in the `location` part of an `Import` directive in  $D$ . The data model instances are constructed from the PSVI if  $D$  contains at least one `Import` directive where the `location` URI is missing and the `profile` identifies an XML schema; or from the infoset, otherwise;
- the value space of all the data types mentioned in the `[type name]` property of any of the information items in the data model instances identified above.

One way to guarantee that this requirement is satisfied is to embed the imported data sources as RIF facts in  $D$ , as specified (non-normatively) in the [Appendix C: Embedding imported data sources as RIF facts](#).

#### Class membership: `rif:Member`

A class membership atom,  $o \# C$ , where  $C$  stands for a RIF constant whose DM-Name is defined, is true if  $o$  identifies one of the element information items in the  $C$ -set constructed from the union of the imported data model instances.

**Example 4.4.** Continuing with the example XML instance document from [Section 3.6](#), and the data model instance that describes it, and assuming that the following RIF class membership atoms occur in a RIF document that imports the XML document, associated with the XML schema from the example:

- `?customer#"http://example.org/customertable#Customer"` is true if the variable `?customer` is bound to an element information item that describes one of the two `Customer` elements in the imported XML document;
- `111#<http://www.w3.org/2001/XMLSchema#type(integer)>` is always false, as far as this specification is concerned: indeed, the atom does not test whether `111` is an integer, but whether `111` identifies an element information item that describes an element of type `xs:integer` in the data model instance that describes an imported data source.

□

#### Sub-class relationship: `rif:Subclass`

A sub-class relationship atom,  $s \## C$ , where  $s$  and  $C$  stand for RIF constants whose DM-Name are defined, is true if, the  $s$ -set is a subset of the  $C$ -set for any given instance of the data model.

Note that the sub-class relationship atom is defined in [RIF-BLD](#) and [RIF-PRD](#) only, not in [RIF-Core](#).

**Example 4.5.** Assuming that the following RIF subclass relationship atoms occur in a RIF document that imports the XML document from the example in [Section 3.6](#):

- Per [example 4.2](#), above, the subclass relationship atom that is represented below in RIF-PRD/XML (or in RIF-BLD/XML) is true if the data model instance is constructed from the PSVI, that is if the XML data source is imported in association with the XML schema from the example. It is false in the schema-less case introduced in [example 4.2](#), however, since, in that case, the super-class is always empty:

```

<Subclass>
  <sub>
    <Const type="xs:string">http://example.org/customertable#Account</Const>
  </sub>
  <super>
    <Const type="rif:iri"><type\(integer\)</type></Const>
  </super>
</Subclass>

```

- The subclass relationship atom that is represented below in RIF-PRD/XML (or in RIF-BLD/XML) is false, as far as this specification is concerned, if the data model instance is constructed from the PSVI. However, it is true if the data model instance has been constructed from the infoset (and no other data source is imported), although an `xs:integer` element can never be an `xs:string` element as well: that is because both classes are empty, in that case. Indeed, the subclass relationship tests that the set of the `xs:integer` elements in the imported data sources is contained in the set of the `xs:string` elements in the same imported data sources; not whether `xs:integer` is a subtype of `xs:string`:

```

<Subclass>
  <sub>
    <Const type="xs:string"><type\(integer\)</type>
  </sub>
  <super>
    <Const type="rif:iri"><type\(string\)</type></Const>
  </super>
</Subclass>

```

□

### Frames: `rif:Frame`

A frame `o [ slot -> v ]`, where `o` identifies an element information item in an imported data model instance, and where `v` identifies a constant whose DM-Name is defined, is true

- either if the local name in `slot` DM-Name has the forme `list (NAME)` and `v` is a RIF list that contains the typed value of all the information items in the `o/slot-sequence`, in document order;
- or if `v` matches the [typed value] property of, at least, one of the information items in the `o/slot-sequence`.

**Example 4.6.** Continuing with the [previous example](#):

- The frame `?x[<http://example.org/customertable#Customer> -> ?y]` is true if `?x` is bound to the

element information item that represents the root `CustomerTable` element, in the imported data source, and if `?y` is bound to one of the two element information items that describe its two `Customer` sub-elements, in the data model instance. That is because the element `Customer` has a complex type with element only content: the [typed value] property of the information item that describes it points, therefore, to the information item itself. For the same reason (element-only children), the value of the frame is the same in the schema-less case, introduced in [example 4.2](#);

- The frame `?x[<http://example.org/customertable#list(Customer)> -> ?y]` is true, for the same binding of `?x`, if `?y` is bound to the sequence that contains, in document order, the two element information items that describe the two `Customer` sub-elements, in the data model instance;
- The frame `?y[<http://example.org/customertable#Account> -> 111]` is true if and only if `?y` is bound to the element information item that represents the `Customer` element with `Name` "John", in the imported data source. This holds in the schema-less case as well;

**Editor's Note:** The typed-value, in the schema-less case, is the string-value, that is, a string. So, the `xs:integer` 111 is not equal to the typed-value of the `Account` node. But it seems reasonable to keep the equality, in the schema-less case, because the type of the RIF constant gives a hint about the intended data model to the consumer. If we agree on this, should it be made explicit in the interpretation of frames?

- For the same binding of `?y` as above, the frame `?y[<http://example.org/customertable#list(Account)> -> List(222 111)]` is false, as far as this specification is concerned, although the RIF list contains the value of all the account elements in the imported document, because it does not follow document order;
- The frame `?y[<http://example.org/customertable#Account> -> "111"]` is false in the example, as far as this specification is concerned, because "111" is an `xs:string`, whereas the schema type of the `Account` element is `xs:integer`. However, the frame would be true, in the schema-less case, since only the string-value of the `Account` element would be known, not its type;
- The frame `?y[<http://www.w3.org/XML/1998/namespace#attribute(lang)> -> "en"^^xs:language]` is true, in the example, if `?y` is bound to the element information item that represents the `Customer` element with `Name` "John", in the imported XML document. This holds in the schema-less case as well. □

**Example 4.7.** Consider the following element with a mixed-content:

```
<letterBody>
  Thank you for ordering the <item>widget</item>.
```

```

    It should arrive by <arrivalDate>09-09-09</arrivalDate>
  </letterBody>

```

Assume that the element is contained in a data source that is imported in a RIF document, and assume that the RIF variable  $?v$ , occurring in that RIF document, is bound to an element information item that describes, in the data model instance, the parent element of the above `letterBody` element.

The value of the frame  $?x["letterBody" \rightarrow ?y]$  will be true is and only if  $?y$  is bound to a string constant that contains the following text: *"Thank you for ordering the Widget. It should arrive by 09-09-09"*; that is, the string value of the information item that describes the `letterBody` element.

Indeed, if an element has a complex type with mixed content (including `xs:anyType`), its typed value is its string value.  $\square$

**Atoms:** `rif:Atom`

An atom,  $P(a_1 \dots a_n)$ ,  $n \geq 0$ , where  $P$  stands for a RIF constant whose DM-Name is defined, is true if there is, at least, one element information item,  $e$ , in the [P-set](#) constructed from the union of the imported data model instances, such that

- the [typed value] property of  $e$  points to  $e$  itself;
- and  $n$  is the number of items in the [children] property of  $e$ ;
- and the  $a_i$  match, respectively, the [typed value] properties of each of the element information items in the [children] property of  $e$ , in document order, [after the references have been resolved](#).

**Editor's Note:** Is that one useful? If we decide to keep it, it needs some more work. E.g., dealing with missing children elements (when `minOccur = 0`); dealing with the fact that  $P$  can occur both in the position of a class identifier in a membership or subclass relationship formula, or as the name of a relation in an atom; etc.

A atom with named arguments,  $P((name_1 a_1) \dots (name_n a_n))$ ,  $n \geq 0$ , where  $P$  stands for a RIF constant whose DM-Name is defined and each of the  $name_i$  are RIF names whose DM-Name is defined, is true if there is, at least, one element information item,  $e$ , in the [P-set](#), such that

- the [typed value] property of  $e$  points to  $e$  itself;
- and each  $a_i$  matches the sequence of the typed values of each of the information items in the  $e/name_i$ -sequence, in document order.

Note that the order in which the  $(name_i a_i)$  pairs occur does not affect the truth value of the atom.

Note that atoms with named arguments are defined only in [RIF-BLD](#), not in [RIF-Core](#) nor in [RIF-PRD](#).

**Editor's Note:** If that is deemed useful, it needs some more work, e.g. extending the definition of DM-Name to RIF names, dealing with  $\mathbb{P}$  occurring both in the position of a class identifier in a membership or subclass relationship formula, and as the name of a relation in an atom; etc.

**Example 4.8.** TBC

## Conformance

**Editor's Note:** This section will be completed in a future draft.

## The special case of RDF and OWL data sources

**Editor's Note:** RDF and dOWL data sources can be imported in RIF documents in two different ways: according to the [RIF-RDF-OWL](#) specification, that specifies the combination of RIF documents with RDF and OWL graphs, directly; or as XML document. This section examines how these two ways of importing RDF and OWL documents relate. The section will be completed in a future draft.

## References

### [DTD]

REF DTD tbd

### [InfoSet]

[XML Information Set \(Second Edition\)](#), John Cowan and Richard Tobin, Editors. World Wide Web Consortium, 04 Feb 2004. This version is <http://www.w3.org/TR/2004/REC-xml-infoSet-20040204>. The latest version is available at <http://www.w3.org/TR/xml-infoSet>.

### [RIF-Core]

REF Core tbd

### [RIF-BLD]

REF BLD tbd

### [RIF-PRD]

REF PRD tbd

### [RIF-RDF-OWL]

REF SWC tbd

**[XF&O]**

REF XF&amp;O tbd

**[XML-SCHEMA]**

REF XML-S tbd

**[XMLS-2]**

REF XML-S part 2 tbd

**[XPath 2.0]**

REF XPath 2.0 tbd

## Appendix A: Glossary (non-normative)

**Editor's Note:** This section will be completed in a future draft.

## Appendix B: Extract from the [XQuery 1.0 and XPath 2.0 Data Model](#) (non-normative)

### Element and attribute node type names (from [XDM](#))

**Editor's Note:** This section reproduces the text of [Section 3.3.1.1. Element and Attribute Node Type Names](#) in [XDM](#), for the reader's convenience. Notice that, for the purpose of this specification, the type name is considered unknown, and the [type name] property is left empty, when the [validaty] property does not exist or is "unknown" and the [validation attempted] property does not exist or is "none".

The precise definition of the schema type of an element or attribute information item depends on the properties of the PSVI. In the PSVI, [Schema Part 1] defines a [type definition] property as well as the [type definition namespace], [type definition name] and [type definition anonymous] properties, which are effectively short-cut terms for properties of the type definition. Further, the [element declaration] and [attribute declaration] properties are defined for elements and attributes, respectively. These declarations in turn will identify the [type definition] declared for the element or attribute. To distinguish the [type definition] given in the PSVI for the element or attribute instance from the [type definition] associated with the declaration, the former is referred to below as the actual type and the latter as the declared type of the element or attribute instance in question.

The type depends on the declared type, the actual type, and the [validity] and [validation attempted] properties in the PSVI. If:

- The [validity] and [validation attempted] properties exist and have the values "valid" and "full", respectively, the schema type of an element or attribute information item is represented by an expanded-QName whose

namespace and local name correspond to the first applicable items in the following list:

- If the declared type exists and is a union and the actual type is (not the same as the declared type, and not a type derived from the declared type, but) one of the member types of the union, or derived from one of its member types:
  - If the {name} property of the declared type is present: the {target namespace} and {name} properties of the declared type.
  - If the {name} property of the declared type is absent: the namespace and local name of the anonymous type name supplied for the declared type.
- If there is no declared type, and the actual type is a union, then:
  - If the {name} property of the actual type is present: the {target namespace} and {name} properties of the actual type.
  - If the {name} property of the actual type is absent: the namespace and local name of the anonymous type name supplied for the actual type.
- Otherwise:
  - If [type definition anonymous] is false: the {target namespace} and {name} properties of the actual type.
  - If [type definition anonymous] is true: the namespace and local name of the anonymous type name supplied for the actual type.
- The [validity] property exists and is "invalid", or the [validation attempted] property exists and is "partial", the schema type of an element is xs:anyType and the type of an attribute is xs:anySimpleType.
- The [validity] property exists and is "notKnown", and the [validation attempted] property exists and is "none", the schema type of an element is xs:untyped and the type of an attribute is xs:untypedAtomic.
- The [validity] or [validation attempted] properties do not exist, the schema type of an element is xs:untyped and the type of an attribute is xs:untypedAtomic.

The prefix associated with the type names is implementation-dependent.

### Typed value determination (from [XDM](#))

This section describes how the typed value of an Element or Attribute Node is computed from an element or attribute PSVI information item, where the information item has either a simple type or a complex type with simple content. [...]

The typed value of Attribute Nodes and some Element Nodes is a sequence of atomic values. The types of the items in the typed value of a node may differ from the type of the node itself. This section describes how the typed value of a node is derived from the properties of an information item in a PSVI.

The types of the items in the typed value of a node are determined as follows. The process begins with *T*, the schema type of the node itself, as represented in the PSVI. For each primitive or ordinary simple type *T*, the W3C XML Schema specification defines a function *M* mapping the lexical representation of a value onto the value itself.

**Note.** For atomic and list types, the mapping is the “lexical mapping” defined for *T* in [Schema Part 2]; for union types, the mapping is the lexical mapping defined in [Schema Part 2] modified as appropriate by any applicable rules in [Schema Part 1]. The mapping, so modified, is a function (in the mathematical sense) which maps to a single value even in cases where the lexical mapping proper maps to multiple values.

The typed value is determined as follows:

- If the nilled property of the node in question is true, then the typed value is the empty sequence.
- If *T* is `xs:anySimpleType` or `xs:anyAtomicType`, the typed value is the [schema normalized value] as an instance of `xs:untypedAtomic`.
- Otherwise, the typed value is the result of applying *M* to the string value as an instance of the appropriate value type, where the appropriate value type is the [member type definition] if *T* is a union type, otherwise it is simply *T*.

The typed value determination process is guaranteed to result in a sequence of atomic values, each having a well-defined atomic type. This sequence of atomic values, in turn, determines the typed-value property of the node in the data model.

## Appendix C: Embedding imported data sources as RIF facts (non-normative)

**Editor's Note:** This section will be completed in a future draft.