Abstract

This document, developed by the Rule Interchange Format (RIF) Working Group, specifies the Basic Logic Dialect, RIF-BLD, a format that allows logic rules to be exchanged between rule systems. The RIF-BLD presentation syntax and semantics are specified both directly and as specializations of the RIF Framework for Logic Dialects, or RIF-FLD. The XML serialization syntax of RIF-BLD is specified via a mapping from the presentation syntax. A normative XML schema is also provided.

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May Be Superseded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 6 documents:

1. RIF Core Dialect
2. RIF Basic Logic Dialect (this document)
3. RIF Framework for Logic Dialects
4. RIF RDF and OWL Compatibility
5. RIF Datatypes and Built-Ins 1.0
6. RIF Production Rule Dialect

Summary of Changes

The changes since the previous version (which was also a "last call" draft) are largely editorial; see the changelog for details. Most of the changes to RIF during this time have been to DTB and to the other dialects.

(Second) Last Call

The Working Group believes it has completed its design work for the technologies specified in this document, so this is a "Last Call" draft. The design is not expected to change significantly going forward, and now is the key time for external review, before the implementation phase. (This is the second Last Call draft of this document.)

Please Comment By 31 July 2009

The Rule Interchange Format (RIF) Working Group seeks public feedback on this Working Draft. Please send your comments to 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 Working 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. 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 Direct Specification of RIF-BLD Presentation Syntax
  ◦ 2.1 Alphabet of RIF-BLD
  ◦ 2.2 Terms
  ◦ 2.3 Formulas
  ◦ 2.4 RIF-BLD Annotations in the Presentation Syntax
  ◦ 2.5 Well-formed Formulas
  ◦ 2.6 EBNF Grammar for the Presentation Syntax of RIF-BLD (Informative)
    - 2.6.1 EBNF for the Condition Language
    - 2.6.2 EBNF for the Rule Language
    - 2.6.3 EBNF for Annotations
• 3 Direct Specification of RIF-BLD Semantics
  ◦ 3.1 Truth Values
  ◦ 3.2 Semantic Structures
  ◦ 3.3 RIF-BLD Annotations in the Semantics
  ◦ 3.4 Interpretation of Non-document Formulas
  ◦ 3.5 Interpretation of Documents
  ◦ 3.6 Logical Entailment
• 4 XML Serialization Syntax for RIF-BLD
  ◦ 4.1 XML for the Condition Language
  ◦ 4.2 XML for the Rule Language
  ◦ 4.3 Mapping from the Presentation Syntax to the XML Syntax
    - 4.3.1 Mapping of the Condition Language
    - 4.3.2 Mapping of the Rule Language
    - 4.3.3 Mapping of Annotations
• 5 Conformance Clauses
• 6 RIF-BLD as a Specialization of the RIF Framework for Logic Dialects [RIF-FLD]
  ◦ 6.1 The Presentation Syntax of RIF-BLD as a Specialization of RIF-FLD
  ◦ 6.2 The Semantics of RIF-BLD as a Specialization of RIF-FLD
  ◦ 6.3 The XML Serialization of RIF-BLD as a Specialization of RIF-FLD
  ◦ 6.4 RIF-BLD ConFORMANCE as a Specialization of RIF-FLD
• 7 Acknowledgements
• 8 References
1 Overview

This specification develops **RIF-BLD** (the **Basic Logic Dialect** of the **Rule Interchange Format**). From a theoretical perspective, RIF-BLD corresponds to the language of definite Horn rules with equality and a standard first-order semantics [CL73]. Syntactically, RIF-BLD has a number of extensions to support features such as objects and frames as in F-logic [KLW95], internationalized resource identifiers (or IRIs, defined by [RFC-3987]) as identifiers for concepts, and XML Schema datatypes [XML-SCHEMA2]. In addition, RIF RDF and OWL Compatibility [RIF-RDF+OWL] defines semantics for the integrated RIF-BLD/RDF and RIF-BLD/OWL languages. These features make RIF-BLD a Web-aware language. However, it should be kept in mind that RIF is designed to enable interoperability among rule languages in general, and its uses are not limited to the Web.

While rule interchange (and not, e.g., execution) is the principle design goal for RIF-BLD, the design clearly indicates a decision to avoid solving the (probably impossible) problem of rule interchange in general. Instead, the design of RIF reflects the rationale of identifying specific kinds of rules within existing rule systems, called **RIF dialects**, that can be translated into other rule systems without changing their meaning. RIF-BLD is just the first in a series of such dialects. In particular, RIF-BLD has the RIF-Core dialect [RIF-Core] as a subset. It is not expected that most rule systems will be able to translate all their rules into RIF-BLD, rather it is expected that only certain kinds of rules will be translatable. Since there are many existing rule languages with useful features that are not supported in RIF-BLD, it is expected that RIF-BLD translators will not translate rules that use such features. This could drive the design of "BLD-specific" rule sets in which rules are specifically written by the implementor to be within the BLD dialect and thus be portable between many rule system implementations.

Among its many influences, RIF shares certain characteristics with ISO Common Logic (CL) [ISO-CL], itself an evolution of KIF [KIF] and Conceptual Graphs [CG]. Like CL, RIF employs XML as its primary normative syntax, uses IRIs as identifiers, specifies integrated RIF-BLD/RDF and RIF-BLD/OWL languages for Semantic Web Compatibility [RIF-RDF+OWL], and provides a rich set of datatypes and built-ins that are designed to be well aligned with Web-aware rule system implementations [RIF-DTB]. Unlike CL, RIF-BLD was designed to be a **simple dialect** with limited expressiveness that lies within the intersection of first-order and logic-programming systems. This is why RIF-BLD does not support negation. More generally, RIF-BLD is part of a coherent array of RIF rule dialects, which encompasses both logic rules -- through the RIF framework for logic dialects [RIF-FLD] also including a variety of rule languages based on non-monotonic theories -- and production rules, as
defined in [RIF-PRD]. CL, on the other hand, is strictly first-order; it does not account for non-monotonic semantics (e.g. negation as failure, defaults, priorities, etc.). For rule interchange between CL and RIF dialects, it is expected that partial RIF-CL mappings will be defined.

RIF-BLD also bears some similarity to SPARQL, in particular with respect to RDF Compatibility [RIF-RDF+OWL]. As with the well-known correspondence between a fragment of SQL and Datalog, SPARQL can be partially mapped to Datalog (and thus to the RIF-Core subset of RIF-BLD), see [AP07] and [AG08] for details. A full mapping of SPARQL would need constructs beyond RIF-BLD, such as non-monotonic negation. Likewise, not all of SPARQL’s FILTER functions are expressible in RIF-DTB built-in predicates. Not all of RIF-BLD is expressible in SPARQL either, for instance recursive rules over RDF Data are not expressible as SPARQL CONSTRUCT statements.

RIF-BLD is defined in two different ways -- both normative:

- As a direct specification, independently of the RIF framework for logic dialects [RIF-FLD], for the benefit of those who desire a direct path to RIF-BLD, e.g., as prospective implementers, and are not interested in extensibility issues. This version of the RIF-BLD specification is given first.
- As a specialization of the RIF framework for logic dialects [RIF-FLD], which is part of the RIF extensibility framework. Building on RIF-FLD, this version of the RIF-BLD specification is comparatively short and is presented in Section RIF-BLD as a Specialization of the RIF Framework at the end of this document. This is intended for the reader who is already familiar with RIF-FLD and does not need to go through the much longer direct specification of RIF-BLD. This section is also useful for dialect designers, as it is a concrete example of how a non-trivial RIF dialect can be derived from the RIF framework for logic dialects.

Logic-based RIF dialects that specialize or extend RIF-BLD in accordance with the RIF framework for logic dialects [RIF-FLD] include RIF-Core as a specialization of RIF-BLD. It is expected that other specifications will develop further logic dialects based on RIF-BLD such as a RIF-BLD extension capturing uncertainty [URD08].

As a preview, here is a simple complete RIF-BLD example deriving a ternary relation from its inverse.

**Example 1** (An introductory RIF-BLD example).

A rule can be written in English to derive the buy relationships (rather than store them) from the sell relationships that are stored as facts (e.g., as exemplified by the English statement below):

- A buyer buys an item from a seller if the seller sells the item to the buyer.
- John sells LeRif to Mary.
Intuitively, the fact Mary buys LeRif from John should be logically derivable from the above premises. Assuming Web IRIs for the predicates buy and sell, as well as for the individuals John, Mary, and LeRif, the above English text can be represented in the RIF-BLD Presentation Syntax as follows.

Document(
  Prefix(cpt <http://example.com/concepts#>)
  Prefix(ppl <http://example.com/people#>)
  Prefix(bks <http://example.com/books#>)

  Group
  {   
    Forall ?Buyer ?Item ?Seller ( 
    )

    cpt:sell(ppl:John bks:LeRif ppl:Mary)
  } 
)

Whenever a RIF-BLD document falls into the Core subset or can be translated to it, the document should be produced in RIF-Core to allow its interchange with a maximum number of RIF consumers. For instance, the Datalog-like RIF document in Example 1 is also a RIF-Core example.

For the interchange of documents containing RIF-BLD rules (and facts), a concrete RIF-BLD XML Syntax is given in this specification. To formalize their meaning, a model-theoretic RIF-BLD Semantics is specified.

2 Direct Specification of RIF-BLD Presentation Syntax

This section specifies the presentation syntax of RIF-BLD directly, without relying on RIF-FLD. In the first five (normative) subsections, the presentation syntax is defined using "mathematical English," a special form of English for communicating mathematical definitions, examples, etc. In the non-normative subsection EBNF Grammar for the Presentation Syntax of RIF-BLD, a grammar for a superset of the presentation syntax is given using Extended Backus–Naur Form (EBNF). Neither the mathematical English nor the EBNF is intended to be a concrete syntax for RIF-BLD. The mathematical English deliberately leaves out details such as the delimiters of the various syntactic components, escape symbols, parenthesizing, precedence of operators, and the like. The EBNF does not specify context-sensitive syntactic constraints. Since RIF is an interchange format, it uses XML as the only concrete syntax, which will be defined in XML Serialization Syntax for RIF-BLD. Hence RIF-BLD conformance is described in terms of semantics-preserving transformations.
Note to the reader: this section depends on Section Constants, Symbol Spaces, and Datatypes of [RIF-DTB].

2.1 Alphabet of RIF-BLD

Definition (Alphabet). The alphabet of the presentation language of RIF-BLD consists of

- a countably infinite set of constant symbols Const
- a countably infinite set of variable symbols Var (disjoint from Const)
- a countably infinite set of argument names, ArgNames (disjoint from Const and Var)
- connective symbols And, Or, and :-
- quantifiers Exists and Forall
- the symbols =, #, ##, ->, External, Import, Prefix, and Base
- the symbols Group and Document
- the symbols for representing lists: List and OpenList.
- the auxiliary symbols (, ), [ , ], < , >, and ^^

The set of connective symbols, quantifiers, =, etc., is disjoint from Const and Var. The argument names in ArgNames are written as Unicode strings that must not start with a question mark, "?". Variables are written as Unicode strings preceded with the symbol "?".

Constants are written as "literal"^symspace, where literal is a sequence of Unicode characters and symspace is an identifier for a symbol space. Symbol spaces are defined in Section Constants, Symbol Spaces, and Datatypes of [RIF-DTB].

The symbols =, #, and ## are used in formulas that define equality, class membership, and subclass relationships. The symbol -» is used in terms that have named arguments and in frame formulas. The symbol External indicates that an atomic formula or a function term is defined externally (e.g., a built-in) and the symbols Prefix and Base enable compact representations of IRIs [RFC-3987].

The symbol Document is used to specify RIF-BLD documents, the symbol Import is an import directive, and the symbol Group is used to organize RIF-BLD formulas into collections. ☐

The language of RIF-BLD is the set of formulas constructed using the above alphabet according to the rules given below.
2.2 Terms

RIF-BLD defines several kinds of terms: constants and variables, positional terms, terms with named arguments, plus equality, membership, subclass, frame, and external terms. The word "term" will be used to refer to any of these constructs.

To simplify the next definition, we will use the phrase base term to refer to simple, positional, or named-argument terms, or to terms of the form External(t), where t is a positional or a named-argument term.

Definition (Term).

1. Constants and variables. If \( t \in \text{Const} \) or \( t \in \text{Var} \) then \( t \) is a simple term.

2. Positional terms. If \( t \in \text{Const} \) and \( t_1, \ldots, t_n, n \geq 0 \), are base terms then \( t(t_1 \ldots t_n) \) is a positional term.

Positional terms correspond to the usual terms and atomic formulas of classical first-order logic [Enderton01, Mendelson97].

3. Terms with named arguments. A term with named arguments is of the form \( t(s_1 \rightarrow v_1 \ldots s_n \rightarrow v_n) \), where \( n \geq 0, t \in \text{Const} \) and \( v_1, \ldots, v_n \) are base terms and \( s_1, \ldots, s_n \) are pairwise distinct symbols from the set ArgNames.

The constant \( t \) here represents a predicate or a function; \( s_1, \ldots, s_n \) represent argument names; and \( v_1, \ldots, v_n \) represent argument values. The argument names, \( s_1, \ldots, s_n \), are required to be pairwise distinct. Terms with named arguments are like positional terms except that the arguments are named and their order is immaterial. Note that a term of the form \( f() \) is, trivially, both a positional term and a term with named arguments.

Terms with named arguments are introduced to support exchange of languages that permit argument positions of predicates and functions to be named (in which case the order of the arguments does not matter).

4. List terms. There are two kinds of list terms: open and closed.
   - A closed list has the form \( \text{List}(t_1 \ldots t_m) \), where \( m \geq 0 \) and \( t_1, \ldots, t_m \) are terms.
   - An open list (or a list with a tail) has the form \( \text{OpenList}(t_1 \ldots t_m \mid t) \), where \( m > 0 \) and \( t_1, \ldots, t_m, t \) are terms. Open lists are usually written using the following: \( \text{List}(t_1 \ldots t_m \mid t) \).

The last argument, \( t \), represents the tail of the list and so it is normally a list as well. However, the syntax does not restrict \( t \) in any way: it could be an integer, a variable, another list, or, in fact,
any term. An example is \texttt{List(1 2 | 3)}. This is not an ordinary list, where the last argument, 3, would represent the tail of a list (and thus would also be a list, which 3 is not). Such general open lists correspond to Lisp's dotted lists [Steele90]. Note that they can be the result of instantiating an open list with a variable in the tail, hence are hard to avoid. For instance, \texttt{List(1 2 | 3)} is \texttt{List(1 2 | ?X)}, where the variable \?X is replaced with 3.

A closed list of the form \texttt{List()} (i.e., a list in which \(m = 0\), corresponding to Lisp's \texttt{nil}) is called the \textbf{empty list}.

5. \textit{Equality terms}. \(t = s\) is an \textbf{equality term}, if \(t\) and \(s\) are base terms.

6. \textit{Class membership terms} (or just \textbf{membership terms}). \(t \# s\) is a \textbf{membership term} if \(t\) and \(s\) are base terms.

7. \textit{Subclass terms}. \(t \## s\) is a \textbf{subclass term} if \(t\) and \(s\) are base terms.

8. \textit{Frame terms}. \(t[p_1->v_1 \ldots p_n->v_n]\) is a \textbf{frame term} (or simply a \textbf{frame}) if \(t, p_1, \ldots, p_n, v_1, \ldots, v_n, n \geq 0\), are base terms.

Membership, subclass, and frame terms are used to describe objects and class hierarchies.

9. \textit{Externally defined terms}. If \(t\) is a positional or a named-argument term then \texttt{External(t)} is an \textbf{externally defined term}.

External terms are used for representing built-in functions and predicates as well as "procedurally attached" terms or predicates, which might exist in various rule-based systems, but are not specified by RIF.

Observe that the argument names of frame terms, \(p_1, \ldots, p_n\), are base terms and so, as a special case, can be variables. In contrast, terms with named arguments can use only the symbols from \texttt{ArgNames} to represent their argument names. They cannot be constants from \texttt{Const} or variables from \texttt{Var}. The reason for not allowing variables for those is to control the complexity of unification, which is used by several inference mechanisms of first-order logic.

\textbf{Example 2 (Terms)}

a. Positional term: "http://example.com/ex1"^^rif:iri(1 "http://example.com/ex2"^^rif:iri(?X 5) "abc")


d. Lists
- Empty list: List()
- Closed list with variable inside: List("a"^^xs:string ?Y "c"^^xs:string)
- Open list with variables: List("a"^^xs:string ?Y "c"^^xs:string | ?Z)
- Equality term with lists inside: List(Head | Tail) = List("a"^^xs:string ?Y "c"^^xs:string)
- Nested list: List("a"^^xs:string List(?X "b"^^xs:string) "c"^^xs:string)

e. Classification terms
- Membership: ?X # ?Y
- Subclass: ?X ## "http://example.com/ex1"^^rif:iri(?Y)
- Membership: "http://example.com/John"^^rif:iri # "http://example.com/Person"^^rif:iri
- Subclass: "http://example.com/Student"^^rif:iri ## "http://example.com/Person"^^rif:iri

f. External term: External(pred:numeric-greater-than(?diffdays 10)))

2.3 Formulas

RIF-BLD distinguishes certain subsets of the set $\text{Const}$ of symbols, including the subset of **predicate symbols** and **function symbols**. Section [Well-formed Formulas](#) gives more details, but we do not need those details yet.

**Definition (Atomic Formula).** Any term (positional or with named arguments) of the form $p(\ldots)$, where $p$ is a predicate symbol, is also an atomic formula.
Equality, membership, subclass, and frame terms are also atomic formulas. An externally defined term of the form $\text{External}(\varphi)$, where $\varphi$ is an atomic formula, is also an atomic formula, called an externally defined atomic formula. □

Note that simple terms (constants and variables) are not formulas.

More general formulas are constructed from atomic formulas with the help of logical connectives.

**Definition (Formula).** A formula can have several different forms and is defined as follows:

1. **Atomic:** If $\varphi$ is an atomic formula then it is also a formula.
2. **Condition formula:** A condition formula is either an atomic formula or a formula that has one of the following forms:
   - **Conjunction:** If $\varphi_1, ..., \varphi_n$, $n \geq 0$, are condition formulas then so is $\text{And}(\varphi_1 \ldots \varphi_n)$, called a conjunctive formula. As a special case, $\text{And}()$ is allowed and is treated as a tautology, i.e., a formula that is always true.
   - **Disjunction:** If $\varphi_1, ..., \varphi_n$, $n \geq 0$, are condition formulas then so is $\text{Or}(\varphi_1 \ldots \varphi_n)$, called a disjunctive formula. As a special case, $\text{Or}()$ is permitted and is treated as a contradiction, i.e., a formula that is always false.
   - **Existentials:** If $\varphi$ is a condition formula and $?V_1, ..., ?V_n$, $n > 0$, are distinct variables then $\text{Exists } ?V_1 \ldots ?V_n(\varphi)$ is an existential formula.

Condition formulas are intended to be used inside the premises of rules. Next we define the notions of rule implications, universal rules, universal facts, groups (i.e., sets of rules and facts), and documents.

3. **Rule implication:** $\varphi : - \psi$ is a formula, called a rule implication, if:
   - $\varphi$ is an atomic formula or a conjunction of atomic formulas,
   - $\psi$ is a condition formula, and
   - none of the atomic formulas in $\varphi$ is an externally defined term (i.e., a term of the form $\text{External}(\ldots)$). Note: external terms can occur in the arguments of atomic formulas in the rule conclusion. For instance, $p(\text{func:numeric-add}(?X, "2"^^xsd:integer)) : - q(?X)$.

**Feature At Risk #2: Equality in the rule conclusion ($\varphi$ in the above)**

Note: This feature is "at risk" and may be removed from this specification based on feedback. Please send feedback to public-rif-comments@w3.org.

4. **Universal rule:** If $\varphi$ is a rule implication and $?V_1, ..., ?V_n$, $n > 0$, are distinct variables then $\text{Forall } ?V_1 \ldots ?V_n(\varphi)$ is a formula, called a universal
rule. It is required that all the free variables in $\varphi$ occur among the variables $?V_1 \ldots ?V_n$ in the quantification part. An occurrence of a variable $?v$ is free in $\varphi$ if it is not inside a substring of the form $Q ?v (\psi)$ of $\varphi$, where $Q$ is a quantifier (Forall or Exists) and $\psi$ is a formula. Universal rules will also be referred to as RIF-BLD rules.

5. Universal fact: If $\varphi$ is an atomic formula and $?V_1, \ldots, ?V_n, n>0$, are distinct variables then Forall $?V_1 \ldots ?V_n(\varphi)$ is a formula, called a universal fact, provided that all the free variables in $\varphi$ occur among the variables $?V_1 \ldots ?V_n$.

Universal facts are often considered to be rules without premises.

6. Group: If $\varphi_1, \ldots, \varphi_n$ are RIF-BLD rules, universal facts, variable-free rule implications, variable-free atomic formulas, or group formulas then Group($\varphi_1 \ldots \varphi_n$) is a group formula. As a special case, the empty group formula, Group(), is allowed and is treated as a tautology, i.e., a formula that is always true.

Non-empty group formulas are used to represent sets of rules and facts. Note that some of the $\varphi_i$’s can be group formulas themselves, which means that groups can be nested.

7. Document: An expression of the form Document($directive_1 \ldots directive_n \Gamma$) is a RIF-BLD document formula (or simply a document formula), if

- $\Gamma$ is an optional group formula; it is called the group formula associated with the document.
- $directive_1, \ldots, directive_n$ is an optional sequence of directives. A directive can be a base directive, a prefix directive or an import directive.

- A base directive has the form Base(<iri>), where $iri$ is a Unicode string in the form of an absolute IRI [RFC-3987].

The Base directive defines a syntactic shortcut for expanding relative IRIs into full IRIs, as described in Section Constants, Symbol Spaces, and Datatypes of [RIF-DTB].

- A prefix directive has the form Prefix($p <v>$), where $p$ is an alphanumeric string that serves as the prefix name and $v$ is an expansion for $p$ — a Unicode sequence of characters that forms an IRI. (An alphanumeric string is a sequence of ASCII characters, where each character is a letter, a digit, or an underscore "_", and the first character is a letter.)
Like the `Base` directive, the `Prefix` directives define shorthands to allow more concise representation of constants that come from the symbol space `rif:iri` (we will call such constants `rif:iri constants`). This mechanism is explained in [RIF-DTB], Section Constants, Symbol Spaces, and Datatypes.

- **An import directive** can have one of these two forms: `Import(<loc>)` or `Import(<loc> <p>)`. Here `loc` is a Unicode sequence of characters that forms an IRI and `p` is another Unicode sequence of characters. The constant `loc` represents the location of another document to be imported; it is called the **locator** of the imported document. The argument `p` is called the **profile of import**; it has the form of a Unicode character sequence in the form of an IRI -- see [RIF-RDF+OWL].

Section Direct Specification of RIF-BLD Semantics of this document defines the semantics for the directive `Import(<loc>)` only. The two-argument directive, `Import(<loc> <p>)`, is intended for importing non-RIF-BLD documents, such as rules from other RIF dialects, RDF data, or OWL ontologies. The profile, `p`, indicates what kind of entity is being imported and under what semantics (for instance, the various RDF entailment regimes have different profiles). The semantics of `Import(<loc> <p>)` (for various `p`) are expected to be given by other specifications on a case-by-case basis. For instance, [RIF-RDF+OWL] defines the semantics for the profiles that are recommended for importing RDF and OWL.

Note that although `Base`, `Prefix`, and `Import` all use symbols of the form `<iri>` to indicate the connection of these symbols to IRIs, these symbols are *not* `rif:iri` constants, as semantically they are interpreted in a way that is quite different from constants.

A document formula can contain at most one `Base` directive. The `Base` directive, if present, must be first, followed by any number of `Prefix` directives, followed by any number of `Import` directives.

In the definition of a formula, the component formulas \( \phi, \phi_1, \psi_i, \) and \( \Gamma \) are said to be **subformulas** of the respective formulas (condition, rule, group, etc.) that are built using these components.
2.4 RIF-BLD Annotations in the Presentation Syntax

RIF-BLD allows every term and formula (including terms and formulas that occur inside other terms and formulas) to be optionally preceded by an annotation of the form (* id φ *), where id is a rif:iri constant and φ is a frame formula or a conjunction of frame formulas. Both items inside the annotation are optional. The id part represents the identifier of the term or formula to which the annotation is attached and φ is the metadata part of the annotation. RIF-BLD does not impose any restrictions on φ apart from what is stated above. This means that it may include variables, function symbols, constants from the symbol space rif:local (often referred to as local or rif:local constants), and so on.

Document formulas with and without annotations will be referred to as RIF-BLD documents.

The following convention is used to avoid a syntactic ambiguity with respect to annotations. The annotation scoping convention associates each annotation to the largest term or formula it precedes. For instance, in (* id φ *) t[w -> v] the metadata annotation could be attributed to the term t or to the entire frame t[w -> v]. The convention specifies that the above annotation is considered to be syntactically attached to the entire frame. Yet, since φ can be a conjunction, some conjuncts can be used to provide metadata targeted to the object part, t, of the frame. For instance, (* And(_foo[meta_for_frame->"this is an annotation for the entire frame"] _bar[meta_for_object->"this is an annotation for t" meta_for_property->"this is an annotation for w"] *) t[w -> v].

We suggest to use Dublin Core, RDFS, and OWL properties for metadata, along the lines of Section 7.1 of [OWL-Reference]—specifically owl:versionInfo, rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs:isDefinedBy, dc:creator, dc:description, dc:date, and foaf:maker.

2.5 Well-formed Formulas

Not all formulas and thus not all documents are well-formed in RIF-BLD: it is required that no constant appear in more than one context. What this means precisely is explained below.

The set of all constant symbols, Const, is partitioned into the following subsets:

- A subset of individuals.
  
  The symbols in Const that belong to the symbol spaces of Datatypes are required to be individuals.

- A subset of plain (i.e., non-external) function symbols.
• A subset for external function symbols.
• A subset of plain predicate symbols.
• A subset for external predicate symbols.

The above subsets do not differentiate between positional and named argument symbols. Also, as seen from the following definitions, these subsets are not specified explicitly but, rather, are inferred from the occurrences of the symbols.

Definition (Context of a symbol). The context of an occurrence of a symbol, \( s \in \text{const} \), in a formula, \( \phi \), is determined as follows:

- If \( s \) occurs as a predicate of the form \( s(...) \) (positional or named-argument) in an atomic subformula of \( \phi \) then \( s \) occurs in the context of a (plain) predicate symbol.
- If \( s \) occurs as a function symbol in a non-subformula term of the form \( s(...) \) then \( s \) occurs in the context of a (plain) function symbol.
- If \( s \) occurs as a predicate in an atomic subformula \( \text{External}(s(...)) \) then \( s \) occurs in the context of an external predicate symbol.
- If \( s \) occurs as a function in a non-subformula term \( \text{External}(s(...)) \) then \( s \) occurs in the context of an external function symbol.
- If \( s \) occurs in any other context (in a frame: \( s[...], [...][s->...], \) or \( [...][...->s] \); or in a positional/named-argument term: \( p(...s...), q(...->s...) \)), it is said to occur as an individual.

Definition (Imported document). Let \( \Delta \) be a document formula and \( \text{Import}(\text{loc}) \) be one of its import directives, where \( \text{loc} \) is a locator of another document formula, \( \Delta' \). We say that \( \Delta' \) is directly imported into \( \Delta \).

A document formula \( \Delta' \) is said to be imported into \( \Delta \) if it is either directly imported into \( \Delta \) or it is imported (directly or not) into some other formula that is directly imported into \( \Delta \).

The above definition deals only with one-argument import directives, since only such directives can be used to import other RIF-BLD documents. Two-argument import directives are provided to enable import of other types of documents, and their semantics are supposed to be covered by other specifications, such as [RIF-RDF+OWL].

Definition (Well-formed formula). A formula \( \phi \) is well-formed iff:

- every constant symbol (whether coming from the symbol space \( \text{rif:local} \) or not) mentioned in \( \phi \) occurs in exactly one context.
- if \( \phi \) is a document formula and \( \Delta'_1, ..., \Delta'_k \) are all of its imported documents, then every non-\( \text{rif:local} \) constant symbol mentioned in \( \phi \) or any of the imported \( \Delta'_1 \)s must occur in exactly one context (in all of the \( \Delta'_1 \)s).
• whenever a formula contains a term or a subformula of the form \( \text{External}(t) \), \( t \) must be an instance of a schema in the coherent set of external schemas (Section Schemas for Externally Defined Terms of [RIF-DTB]) associated with the language of RIF-BLD.

• if \( t \) is an instance of a schema in the coherent set of external schemas associated with the language then \( t \) can occur only as \( \text{External}(t) \), i.e., as an external term or atomic formula. ☐

Definition (Language of RIF-BLD). The language of RIF-BLD consists of the set of all well-formed formulas and is determined by:

- the alphabet of the language and
- a set of coherent external schemas, which determine the available built-ins and other externally defined predicates and functions. ☐

2.6 EBNF Grammar for the Presentation Syntax of RIF-BLD (Informative)

Until now, we have been using mathematical English to specify the syntax of RIF-BLD. Tool developers, however, may prefer EBNF notation, which provides a more succinct view of the syntax. Several points should be kept in mind regarding this notation.

• The syntax of first-order logic is not context-free, so EBNF cannot capture the syntax of RIF-BLD precisely. For instance, it cannot capture the well-formedness conditions, i.e., the requirement that each symbol in RIF-BLD can occur in at most one context. As a result, the EBNF grammar defines a strict superset of RIF-BLD: not all formulas that are derivable using the EBNF grammar are well-formed formulas in RIF-BLD.

• The EBNF grammar does not address all details of how constants (defined in [RIF-DTB]) and variables are represented, and it is not sufficiently precise about the delimiters and escape symbols. White space is informally used as a delimiter, and is implied in productions that use Kleene star. For instance, \( \text{TERM}^{*} \) is to be understood as \( \text{TERM} \text{ TERM} \ldots \text{ TERM} \), where each space abstracts from one or more blanks, tabs, newlines, etc. This is so because RIF's presentation syntax is a tool for specifying the semantics and for illustration of the main RIF concepts through examples. It is not intended as a concrete syntax for a rule language. RIF defines a concrete syntax only for exchanging rules, and that syntax is XML-based, obtained as a refinement and serialization of the presentation syntax.

• For all the above reasons, the EBNF grammar is not normative. Recall from the opening paragraph, however, that the RIF-BLD presentation syntax as specified in mathematical English is normative.
The EBNF for the RIF-BLD presentation syntax is given as follows, showing the entire (top-down) context of its three parts for rules, conditions, and annotations.

**Rule Language:**

Base ::= 'Base' '(' ANGLEBRACKIRI ')'
Prefix ::= 'Prefix' '(' Name ANGLEBRACKIRI ')
Import ::= IRIMETA? 'Import' '(' LOCATOR PROFILE? ')
Group ::= IRIMETA? 'Group' '(' (RULE | Group)* ')
RULE ::= (IRIMETA? 'Forall' Var+ '(' CLAUSE ')') | CLAUSE
CLAUSE ::= Implies | ATOMIC
Implies ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* '))' ':-' FORMULA
LOCATOR ::= ANGLEBRACKIRI
PROFILE ::= ANGLEBRACKIRI

**Condition Language:**

FORMULA ::= IRIMETA? 'And' '(' FORMULA* ')') | IRIMETA? 'Or' '(' FORMULA* ')') | IRIMETA? 'Exists' Var+ '(' FORMULA ')') | ATOMIC | IRIMETA? 'External' '(' Atom ')
ATOMIC ::= IRIMETA? (Atom | Equal | Member | Subclass | Frame)
Atom ::= UNITERM
UNITERM ::= Const '(' (TERM* | (Name -> TERM)* ')')
Equal ::= TERM '=' TERM
Member ::= TERM '#' TERM
Subclass ::= TERM '##' TERM
Frame ::= TERM '[' (TERM | Term -> TERM)* ']
TERM ::= IRIMETA? (Const | Var | Expr | List | 'External' '(' Expr ')
Expr ::= UNITERM
List ::= 'List' '(' (TERM* ')') | 'List' '(' TERM+ '|' TERM ')
Const ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
Name ::= UNICODESTRING
Var ::= '?' UNICODESTRING
SYMSPACE ::= ANGLEBRACKIRI | CURIE

**Annotations:**

IRIMETA ::= '(*' IRICONST? (Frame | 'And' '(' Frame* '))? '*')

The following subsections explain and exemplify these parts, starting with the basic language of positive conditions.
2.6.1 EBNF for the Condition Language

The Condition Language represents formulas that can be used in the premises of RIF-BLD rules (also called rule bodies). The EBNF grammar for a superset of the RIF-BLD condition language is shown in the above conditions part.

The production rule for the non-terminal FORMULA represents RIF condition formulas (defined earlier). The connectives And and Or define conjunctions and disjunctions of conditions, respectively. Exists introduces existentially quantified variables. Here Var+ stands for the list of variables that are free in FORMULA. RIF-BLD conditions permit only existential variables. A RIF-BLD FORMULA can also be an ATOMIC term, i.e. an Atom, External Atom, Equal, Member, Subclass, or Frame. A TERM can be a constant, variable, Expr, List, or External Expr.

The RIF-BLD presentation syntax does not commit to any particular vocabulary and permits arbitrary Unicode strings in constant symbols, argument names, and variables. Constant symbols can have this form: "UNICODESTRING"^^SYMSPACE, where SYMSPACE is an ANGLEBRACKIRI or CURIE that represents the identifier of the symbol space of the constant, and UNICODESTRING is a Unicode string from the lexical space of that symbol space. ANGLEBRACKIRI and CURIE are defined in Section Shortcuts for Constants in RIF’s Presentation Syntax of [RIF-DTB]. Constant symbols can also have several shortcut forms, which are represented by the non-terminal CONSTSHORT. These shortcuts are also defined in the same section of [RIF-DTB]. One of them is the CURIE shortcut, which is extensively used in the examples in this document. Names are Unicode character sequences. Variables are composed of UNICODESTRING symbols prefixed with a ?-sign.

Equality, membership, and subclass terms are self-explanatory. An Atom and Expr (expression) can either be positional or have named arguments. A frame term is a term composed of an object identifier and a collection of attribute-value pairs. The term External(Atom) is a call to an externally defined predicate. Likewise, External(Expr) is a call to an externally defined function.

Example 3 (RIF-BLD conditions).

This example shows conditions that are composed of atoms, expressions, equalities with lists, frames, and existentials. In frame formulas, variables are shown in the positions of object identifiers, object properties, and property values. For brevity, we use the shortcut CURIE notation prefix:suffix for constant symbols defined in [RIF-DTB]. This is understood as a shorthand for an IRI obtained by concatenation of the prefix definition and suffix. Thus, if bks is a prefix that expands into http://example.com/books# then bks:LeRif is an abbreviation for "http://example.com/books#LeRif"^^rif:iri.
Prefix(bks <http://example.com/books#>)
Prefix(auth <http://example.com/authors#>)
Prefix(cpt <http://example.com/concepts#>)

Positional terms:

cpt:book(auth:rifwg bks:LeRif)
Exists ?X (cpt:book(?X bks:LeRif))

Terms with named arguments:


Equalities with list terms:

?L = List(?X ?Y ?X)
List(?Head | ?Tail) = List("a"^^rif:local ?Y "c"^^rif:local)

Frames:


2.6.2 EBNF for the Rule Language

The presentation syntax for RIF-BLD rules is based on the syntax in Section EBNF for RIF-BLD Condition Language with the productions shown in the above rules part.

A RIF-BLD Document consists of an optional Base, followed by any number of Prefixes, followed by any number of Imports, followed by an optional Group. Base and Prefix serve as shortcut mechanisms for IRIs. IRI has the form of an internationalized resource identifier as defined by [RFC-3987]. An Import indicates the location of a document to be imported and an optional profile. A RIF-
BLD Group is a collection of any number of RULE elements along with any number of nested Groups.

Rules are generated using CLAUSE elements. The RULE production has two alternatives:

- In the first, a CLAUSE is in the scope of the Forall quantifier. In that case, all variables mentioned in CLAUSE are required to also appear among the variables in the Var+ sequence.
- In the second alternative, CLAUSE appears on its own. In that case, CLAUSE cannot have variables.

Var, ATOMIC, and FORMULA were defined as part of the syntax for positive conditions in Section EBNF for RIF-BLD Condition Language. In the CLAUSE production, an ATOMIC is what is usually called a fact. An Implies rule can have an ATOMIC or a conjunction of ATOMIC elements as its conclusion; it has a FORMULA as its premise. Note that, by the definition of formulas, externally defined atoms (i.e., formulas of the form External(Atom)) are not allowed in the conclusion part of a rule (ATOMIC does not expand to External).

Example 4 (RIF-BLD rules).

This example shows a business rule borrowed from the document RIF Use Cases and Requirements:

- If an item is perishable and it is delivered to John more than 10 days after the scheduled delivery date then the item will be rejected by him.

As before, for better readability we use the compact URI notation defined in [RIF-DTB], Section Constants, Symbol Spaces, and Datatypes. Again, prefix directives are assumed in the preamble to the document. Then, two versions of the main part of the document are given.

```xml
Prefix(ppl  <http://example.com/people#>)
Prefix(cpt  <http://example.com/concepts#>)
Prefix(func <http://www.w3.org/2007/rif-built-in-function#>)
Prefix(pred <http://www.w3.org/2007/rif-built-in-predicate#>)
```

a. Universal form:

```
cpt:reject(ppl:John ?item) :-  
  And(cpt:perishable(?item)  
    cpt:delivered(?item ?deliverydate ppl:John)  
    cpt:scheduled(?item ?scheduledate))
```
b. Universal-existential form:

\[
\text{forall } \text{?item} \rightarrow
\text{cpt:reject(ppl:John } \text{?item } \rightarrow-
\right. \\
\left. \text{exists } \text{?deliverydate } \text{?scheduledate } \text{?diffduration } \text{?diffdays} \rightarrow
\left. \begin{array}{l}
\text{And(cpt:perishable(?item)} \\
\text{cpt:delivered(?item ?deliverydate ppl:John)} \\
\text{cpt:scheduled(?item ?scheduledate)} \\
\text{?diffduration = External(func:subtract-dateTimes(?deliverydate ?scheduledate)} \\
\text{?diffdays = External(func:days-from-duration(?diffduration)} \\
\text{External(pred:numeric-greater-than(?diffdays 10)))}
\end{array}
\right)
\]

2.6.3 EBNF for Annotations

The EBNF grammar production for RIF-BLD annotations is shown in the above annotations part.

As explained in Section RIF-BLD Annotations in the Presentation Syntax, RIF-BLD formulas and terms can be prefixed with optional annotations, IRIMETA, for identification and metadata. IRIMETA is represented using (*...*)-brackets that contain an optional rif:iri constant, IRICONST, as identifier followed by an optional Frame or conjunction of Frames as metadata.

An IRICONST is rif:iri constant, again permitting the shortcut forms defined in [RIF-DTB]. One such specialization is "'IRI"^^' 'rif:iri' from the Const production, where IRI is a sequence of Unicode characters that forms an internationalized resource identifier as defined by [RFC-3987].

Example 5 (A RIF-BLD document containing an annotated group).

This example shows a complete document containing a group formula that consists of two RIF-BLD rules. The first of these rules is copied from Example 4a. The group is annotated with an IRI identifier and metadata represented using Dublin Core vocabulary.
3 Direct Specification of RIF-BLD Semantics

This normative section specifies the semantics of RIF-BLD directly, without relying on [RIF-FLD].

Recall that the presentation syntax of RIF-BLD allows shorthand notation, which is specified via the Prefix and Base directives, and various shortcuts for integers, strings, and rif:local symbols. The semantics, below, is described using the full syntax, i.e., we assume that all shortcuts have already been expanded as defined in [RIF-DTB], Section Constants, Symbol Spaces, and Datatypes.

3.1 Truth Values

The set $TV$ of truth values in RIF-BLD consists of two values, $t$ and $f$. 

```xml
Document(
  Prefix(ppl <http://example.com/people#>)
  Prefix(cpt <http://example.com/concepts#>)
  Prefix(dc <http://purl.org/dc/terms/>)
  Prefix(func <http://www.w3.org/2007/rif-built-in-function#>)
  Prefix(pred <http://www.w3.org/2007/rif-built-in-predicate#>)
  Prefix(xs <http://www.w3.org/2001/XMLSchema#>)

dc:date -> "2008-04-04"^^xs:date] * )

  Group
      And(cpt:perishable(?item)
        cpt:delivered(?item ?deliverydate ppl:John)
        cpt:scheduled(?item ?scheduledate)
        ?diffduration = External(func:subtract-dateTimes(?deliverydate
        ?scheduledate))
        ?difffdays = External(func:days-from-duration(?diffduration))
        External(pred:numeric-greater-than(?difffdays 10)))

    Forall ?item (cpt:reject(ppl:Fred ?item) :- cpt:unsolicited(?item)
    )
  )
  )
)
3.2 Semantic Structures

The key concept in a model-theoretic semantics of a logic language is the notion of a *semantic structure* [Enderton01, Mendelson97]. The definition is slightly more general than what is strictly necessary for RIF-BLD alone. This lays the groundwork for extensions to RIF-BLD and makes the connection with the semantics of the RIF framework for logic-based dialects [RIF-FLD] more obvious.

**Definition (Semantic structure).** A semantic structure, $I$, is a tuple of the form $\langle TV, DTS, D, D_{\text{ind}}, D_{\text{func}}, I_C, I_V, I_F, I_{\text{list}}, I_{\text{fail}}, I_{\text{frame}}, I_{\text{sub}}, I_{\text{isa}}, I_{\text{=}}, I_{\text{external}}, I_{\text{truth}} \rangle$. Here $D$ is a non-empty set of elements called the domain of $I$, and $D_{\text{ind}}, D_{\text{func}}$ are nonempty subsets of $D$. $D_{\text{ind}}$ is used to interpret the elements of $\text{Const}$ that occur as individuals and $D_{\text{func}}$ is used to interpret the elements of $\text{Const}$ that occur in the context of function symbols. As before, $\text{Const}$ denotes the set of all constant symbols and $\text{Var}$ the set of all variable symbols. $DTS$ denotes a set of identifiers for datatypes (please refer to Section **Datatypes** of [RIF-DTB] for the semantics of datatypes).

The other components of $I$ are total mappings defined as follows:

1. $I_C$ maps $\text{Const}$ to $D$.
   
   This mapping interprets constant symbols. In addition:
   
   - If a constant, $c \in \text{Const}$, is an *individual* then it is required that $I_C(c) \in D_{\text{ind}}$.
   - If $c \in \text{Const}$, is a *function symbol* (positional or with named arguments) then it is required that $I_C(c) \in D_{\text{func}}$.

2. $I_V$ maps $\text{Var}$ to $D_{\text{ind}}$.
   
   This mapping interprets variable symbols.

3. $I_F$ maps $D$ to functions $D_{\text{ind}}^* \rightarrow D$ (here $D_{\text{ind}}^*$ is a set of all finite sequences over the domain $D_{\text{ind}}$).
   
   This mapping interprets positional terms. In addition:
   
   - If $d \in D_{\text{func}}$ then $I_F(d)$ must be a function $D_{\text{ind}}^* \rightarrow D_{\text{ind}}$.
   - This means that when a function symbol is applied to arguments that are individual objects then the result is also an individual object.

4. $I_{\text{NF}}$ maps $D$ to the set of total functions of the form $\text{SetOfFiniteSets(} \text{ArgNames} \times D_{\text{ind}}\text{)} \rightarrow D$.
   
   This mapping interprets function symbols with named arguments. In addition:
• If $d \in D_{\text{func}}$ then $I_{\text{INF}}(d)$ must be a function $\text{SetOfFiniteSets}(\text{ArgNames} \times D_{\text{ind}}) \rightarrow D_{\text{ind}}$.
• This is analogous to the interpretation of positional terms with two differences:
  ▪ Each pair $<s,v> \in \text{ArgNames} \times D_{\text{ind}}$ represents an argument/value pair instead of just a value in the case of a positional term.
  ▪ The arguments of a term with named arguments constitute a finite set of argument/value pairs rather than a finite ordered sequence of simple elements. So, the order of the arguments does not matter.

5. $I_{\text{list}}$ and $I_{\text{tail}}$ are used to interpret lists. They are mappings of the following form:
• $I_{\text{list}} : D_{\text{ind}}^* \rightarrow D_{\text{ind}}$
• $I_{\text{tail}} : D_{\text{ind}}^+ \times D_{\text{ind}} \rightarrow D_{\text{ind}}$

In addition, these mappings are required to satisfy the following conditions:
• The function $I_{\text{list}}$ is injective (one-to-one).
• The set $I_{\text{list}}(D_{\text{ind}})$, henceforth denoted $D_{\text{list}}$, is disjoint from the value spaces of all data types in $DTS$.
• $I_{\text{tail}}(a_1, ..., a_k, I_{\text{list}}(a_{k+1}, ..., a_{k+m})) = I_{\text{list}}(a_1, ..., a_k, a_{k+1}, ..., a_{k+m})$.

Note that the last condition above restricts $I_{\text{tail}}$ only when its last argument is in $D_{\text{list}}$. If the last argument of $I_{\text{tail}}$ is not in $D_{\text{list}}$, then the list is a general open one and there are no restrictions on the value of $I_{\text{tail}}$ except that it must be in $D_{\text{ind}}$.

6. $I_{\text{frame}}$ maps $D_{\text{ind}}$ to total functions of the form $\text{SetOfFiniteBags}(D_{\text{ind}} \times D_{\text{ind}}) \rightarrow D$.

This mapping interprets frame terms. An argument, $d \in D_{\text{ind}}$, to $I_{\text{frame}}$ represents an object and the finite bag $\{<a_1,v_1>, ..., <a_k,v_k>\}$ represents a bag of attribute-value pairs for $d$. We will see shortly how $I_{\text{frame}}$ is used to determine the truth valuation of frame terms.

Bags (multi-sets) are used here because the order of the attribute/value pairs in a frame is immaterial and pairs may repeat. Such repetitions arise naturally when variables are instantiated with constants. For instance, $o[\text{?A->?B ?C->?D}]$ becomes $o[\text{a->b a->b}]$ if variables $\text{?A}$ and $\text{?C}$ are instantiated with the symbol $a$ while $\text{?B}$ and $\text{?D}$ are instantiated with $b$. (We shall see later that $o[\text{a->b a->b}]$ is equivalent to $o[\text{a->b}]$.)

7. $I_{\text{sub}}$ gives meaning to the subclass relationship. It is a mapping of the form $D_{\text{ind}} \times D_{\text{ind}} \rightarrow D$. 


\[ I_{\text{sub}} \] will be further restricted in Section Interpretation of Formulas to ensure that the operator \( \# \) is transitive, i.e., that \( c_1 \# c_2 \) and \( c_2 \# c_3 \) imply \( c_1 \# c_3 \).

8. \[ I_{\text{sa}} \] gives meaning to class membership. It is a mapping of the form \( D_{\text{ind}} \times D_{\text{ind}} \rightarrow D \).

\[ I_{\text{sa}} \] will be further restricted in Section Interpretation of Formulas to ensure that the relationships \( \# \) and \( \## \) have the usual property that all members of a subclass are also members of the superclass, i.e., that \( \circ \# c_1 \) and \( c_1 \# \text{ scl} \) imply \( \circ \# \text{ scl} \).

9. \[ I_{=\text{}} \] is a mapping of the form \( D_{\text{ind}} \times D_{\text{ind}} \rightarrow D \).

It gives meaning to the equality operator.

10. \[ I_{\text{truth}} \] is a mapping of the form \( D \rightarrow TV \).

It is used to define truth valuation for formulas.

11. \[ I_{\text{external}} \] is a mapping from the coherent set of schemas for externally defined functions to total functions \( D^* \rightarrow D \). For each external schema \( \sigma = (\text{?X}_1 \ldots \text{?X}_n; \tau) \) in the coherent set of external schemas associated with the language, \( I_{\text{external}}(\sigma) \) is a function of the form \( D^n \rightarrow D \).

For every external schema, \( \sigma \), associated with the language, \( I_{\text{external}}(\sigma) \) is assumed to be specified externally in some document (hence the name external schema). In particular, if \( \sigma \) is a schema of a RIF built-in predicate or function, \( I_{\text{external}}(\sigma) \) is specified in [RIF-DTB] so that:

- If \( \sigma \) is a schema of a built-in function then \( I_{\text{external}}(\sigma) \) must be the function defined in [RIF-DTB].
- If \( \sigma \) is a schema of a built-in predicate then \( I_{\text{truth}} \circ (I_{\text{external}}(\sigma)) \) (the composition of \( I_{\text{truth}} \) and \( I_{\text{external}}(\sigma) \), a truth-valued function) must be as specified in [RIF-DTB].

We also define the following mapping from terms to \( D \), which we denote using the same symbol \( I \) as the one used for semantic structures. This overloading is convenient and creates no ambiguity.

- \( I(\text{k}) = I_{\text{C}}(\text{k}) \), if \( \text{k} \) is a symbol in \( \text{Const} \).
- \( I(\text?v}) = I_{\text{V}}(\text?v}) \), if \( \text?v} \) is a variable in \( \text{Var} \).
- \( I(f(t_1 \ldots t_n)) = I_{\text{F}}(I(f))(I(t_1),\ldots,I(t_n)) \).
- \( I(f(s_1->v_1 \ldots s_n->v_n)) = I_{\text{NF}}(I(f))(\langle<s_1,I(v_1)>,\ldots,<s_n,I(v_n)>\rangle) \).

Here we use \( {...} \) to denote a set of argument/value pairs.

- For list terms, the mapping is defined as follows:
  - \( I(\text{List}()) = I_{\text{list}}(<>). \)
Here \(<>\) denotes an empty list of elements of \(D_{\text{ind}}\). (Note that the domain of \(I_{\text{list}}\) is \(D_{\text{ind}}\), so \(D_{\text{ind}}^0\) is an empty list of elements of \(D_{\text{ind}}\).)

- \(I(\text{List}(t_1 \ldots t_n)) = I_{\text{list}}(I(t_1), \ldots, I(t_n))\), if \(n > 0\).
- \(I(\text{List}(t_1 \ldots t_n | t)) = I_{\text{tail}}(I(t_1), \ldots, I(t_n), I(t))\), if \(n > 0\).

- \(I(o[a_1 -> v_1 \ldots a_k -> v_k]) = I_{\text{frame}}(I(o))\{<I(a_1), I(v_1)>, \ldots, <I(a_n), I(v_n)>\}\)

Here \{…\} denotes a bag of attribute/value pairs. Jumping ahead, we note that duplicate elements in such a bag do not affect the value of \(I_{\text{frame}}(I(o))\) -- see Section Interpretation of Non-document Formulas. For instance, \(I(o[a->b \ a->b]) = I(o[a->b])\).

- \(I(c1##c2) = I_{\text{sub}}(I(c1), I(c2))\)
- \(I(o#c) = I_{\text{isa}}(I(o), I(c))\)
- \(I(x=y) = I_{=}(I(x), I(y))\)
- \(I(\text{External}(t)) = I_{\text{external}}(I(s_1), \ldots, I(s_n))\), if \(t\) is an instance of the external schema \(\sigma = (?X_1 \ldots ?X_n; \ t)\) by substitution \(?X_1/s_1 \ldots ?X_n/s_1\).

Note that, by definition, \(\text{External}(t)\) is well-formed only if \(t\) is an instance of an external schema. Furthermore, by the definition of coherent sets of external schemas, \(t\) can be an instance of at most one such schema, so \(I(\text{External}(t))\) is well-defined.

### The effect of datatypes.

The set \(DTS\) must include the datatypes described in Section Datatypes of [RIF-DTB].

The datatype identifiers in \(DTS\) impose the following restrictions. Given \(dt \in DTS\), let \(LS_{dt}\) denote the lexical space of \(dt\), \(VS_{dt}\) denote its value space, and \(L_{dt}: LS_{dt} \rightarrow VS_{dt}\) the lexical-to-value-space mapping (for the definitions of these concepts, see Section Datatypes of [RIF-DTB]). Then the following must hold:

- \(VS_{dt} \subseteq D_{\text{ind}}\); and
- For each constant "lit"\(^{\text{dt}}\) such that \(\text{lit} \in LS_{dt}\), \(I_C("\text{lit}"^{\text{dt}}) = L_{dt}(\text{lit})\).

That is, \(I_C\) must map the constants of a datatype \(dt\) in accordance with \(L_{dt}\).

RIF-BLD does not impose restrictions on \(I_C\) for constants in symbol spaces that are not datatypes included in \(DTS\). \(\Box\)
3.3 RIF-BLD Annotations in the Semantics

RIF-BLD annotations are stripped before the mappings that constitute RIF-BLD semantic structures are applied. Likewise, they are stripped before applying the truth valuation, $TVal_I$, defined in the next section. Thus, identifiers and metadata have no effect on the formal semantics.

Note that although identifiers and metadata associated with RIF-BLD formulas are ignored by the semantics, they can be extracted by XML tools. The frame terms used to represent RIF-BLD metadata can then be fed to other RIF-BLD rules, thus enabling reasoning about metadata. RIF-BLD does not define any particular semantics for metadata, however.

3.4 Interpretation of Non-document Formulas

This section defines how a semantic structure, $I$, determines the truth value $TVal_I(\varphi)$ of a RIF-BLD formula, $\varphi$, where $\varphi$ is any formula other than a document formula. Truth valuation of document formulas is defined in the next section.

We define a mapping, $TVal_I$, from the set of all non-document formulas to $TV$. Note that the definition implies that $TVal_I(\varphi)$ is defined only if the set $DTS$ of the datatypes of $I$ includes all the datatypes mentioned in $\varphi$ and $I_{\text{external}}$ is defined on all externally defined functions and predicates in $\varphi$.

**Definition (Truth valuation).** *Truth valuation* for well-formed formulas in RIF-BLD is determined using the following function, denoted $TVal_I$:

1. **Positional atomic formulas:** $TVal_I(r(t_1 \ldots t_n)) = h_{\text{truth}}(l(r(t_1 \ldots t_n)))$

2. **Atomic formulas with named arguments:** $TVal_I(p(s_1->v_1 \ldots s_k->v_k))$
   $= h_{\text{truth}}(l(p(s_1->v_1 \ldots s_k->v_k)))$.

3. **Equality:** $TVal_I(x = y) = h_{\text{truth}}(l(x = y))$.
   - To ensure that equality has precisely the expected properties, it is required that:
     - $h_{\text{truth}}(l(x = y)) = t$ if $l(x) = l(y)$ and that $h_{\text{truth}}(l(x = y)) = f$ otherwise.
     - This is tantamount to saying that $TVal_I(x = y) = t$ if and only if $l(x) = l(y)$.

4. **Subclass:** $TVal_I(sc ## cl) = h_{\text{truth}}(l(sc ## cl))$.
   - To ensure that the operator ## is transitive, i.e., $c_1 ## c_2$ and $c_2 ## c_3$ imply $c_1 ## c_3$, the following is required:
5. **Membership**: $TVaI(o \ # cl) = l_{truth}(l(o \ # cl))$.

To ensure that all members of a subclass are also members of the superclass, i.e., $o \ # cl$ and $cl \ # scl$ imply $o \ # scl$, the following is required:

- For all $o, cl, scl \in D$, if $TVaI(o \ # cl) = TVaI(cl \ # scl) = t$ then $TVaI(o \ # scl) = t$.

6. **Frame**: $TVaI(o[a_1->v_1 \ldots a_k->v_k]) = l_{truth}(l(o[a_1->v_1 \ldots a_k->v_k]))$.

Since the bag of attribute/value pairs associated with an object $o$ represents the conjunction of pairs each separately associated with $o$, the following is required, if $k > 0$:

- $TVaI(o[a_1->v_1 \ldots a_k->v_k]) = t$ if and only if $TVaI(o[a_1->v_1]) = \ldots = TVaI(o[a_k->v_k]) = t$.

7. **Externally defined atomic formula**: $TVaI(\text{External}(t)) = l_{truth}(\text{External}(c)(l(s_1), \ldots, l(s_n)))$, if $t$ is an atomic formula that is an instance of the external schema $\sigma = (\exists X_1 \ldots ?X_n; t)$ by substitution $?X_1/s_1 \ldots ?X_n/s_1$.

Note that, by definition, External($t$) is well-formed only if $t$ is an instance of an external schema. Furthermore, by the definition of coherent sets of external schemas, $t$ can be an instance of at most one such schema, so $l(\text{External}(t))$ is well-defined.

8. **Conjunction**: $TVaI(\text{And}(c_1 \ldots c_n)) = t$ if and only if $TVaI(c_1) = \ldots = TVaI(c_n) = t$. Otherwise, $TVaI(\text{And}(c_1 \ldots c_n)) = f$.

The empty conjunction is treated as a tautology, so $TVaI(\text{And}()) = t$.

9. **Disjunction**: $TVaI(\text{Or}(c_1 \ldots c_n)) = f$ if and only if $TVaI(c_1) = \ldots = TVaI(c_n) = f$. Otherwise, $TVaI(\text{Or}(c_1 \ldots c_n)) = t$.

The empty disjunction is treated as a contradiction, so $TVaI(\text{Or}()) = f$.

10. **Quantification**:

- $TVaI(\exists X_1 \ldots ?X_n (\varphi)) = t$ if and only if for some $l^*$, described below, $TVaI(\varphi) = t$.
- $TVaI(\forall X_1 \ldots ?X_n (\varphi)) = t$ if and only if for every $l^*$, described below, $TVaI(\varphi) = t$.

Here $l^*$ is a semantic structure of the form $<TV, DTS, D, D_{\text{ind}}, D_{\text{func}}, IC, \Gamma_V, \Gamma_F, \Gamma_{\text{list}}, \Gamma_{\text{frame}}, \Gamma_{\text{sub}}, \Gamma_{\text{sa}}, \Gamma, l_{\text{external}}, l_{\text{truth}}>$, which is exactly like
I, except that the mapping \( I^*_V \) is used instead of \( I_V \). \( I^*_V \) is defined to coincide with \( I_V \) on all variables except, possibly, on \(?v_1, \ldots, ?v_n\).

11. **Rule implication:**
   - \( TVal_I(\text{conclusion} \leftarrow \text{condition}) = t \), if either \( TVal_I(\text{conclusion}) = t \) or \( TVal_I(\text{condition}) = t \).
   - \( TVal_I(\text{conclusion} \leftarrow \text{condition}) = f \) otherwise.

12. **Groups of rules:**

   If \( \Gamma \) is a group formula of the form \( \text{Group}(\phi_1 \ldots \phi_n) \) then
   - \( TVal_I(\Gamma) = t \) if and only if \( TVal_I(\phi_1) = t, \ldots, TVal_I(\phi_n) = t \).
   - \( TVal_I(\Gamma) = f \) otherwise.

   This means that a group of rules is treated as a conjunction. In particular, the empty group is treated as a tautology, so \( TVal_I(\text{Group}()) = t \).

### 3.5 Interpretation of Documents

Document formulas are interpreted using *semantic multi-structures*, which are sets of closely related semantics structures. The need for multi-structures arises due to the fact that a RIF-BLD document can import other documents and thus is essentially a multi-document object. One interesting aspect of the multi-document semantics is that rif:local symbols that belong to different documents can have different meanings.

**Definition (Semantic multi-structure).** A *semantic multi-structure* \( \tilde{I} \) is a set of semantic structures of the form \( \{ J, I; I^1, I^2, \ldots \} \), where

- \( I \) and \( J \) are RIF-BLD semantic structures; and
- \( I^1, I^2, \ldots \) are semantic structures adorned with the locators of distinct RIF-BLD formulas (one can think of these adorned structures as locator-structure pairs).

All the structures in \( \tilde{I} \) (adorned and non-adorned) are identical in all respects except for the following:

- The mappings \( J_C, I_C, I^{1}_C, I^{2}_C, \ldots \) may differ on the constants in Const that belong to the rif:local symbol space.

As will be seen from the next definition, the structure \( I \) in the above is used to interpret document formulas, and the adorned structures of the form \( I^{i}_k \) are used to interpret imported documents. The structure \( J \) is used in the definition of entailment to interpret non-document formulas.

The semantics of RIF documents is now defined as follows.
Definition (Truth valuation of document formulas). Let $\Delta$ be a document formula and let $\Delta_1, \ldots, \Delta_n$ be all the RIF-BLD document formulas that are imported (directly or indirectly, according to Definition Imported document) into $\Delta$. Let $\Gamma, \Gamma_1, \ldots, \Gamma_n$ denote the respective group formulas associated with these documents. Let $I = \{I_1, \ldots, I_n\}$ be a semantic multi-structure that contains the semantic structures adorned with the locators $i_1, \ldots, i_n$ of the documents $\Delta_1, \ldots, \Delta_n$. Then we define:

- $TVal(I(\Delta)) = t$ if and only if $TVal(I(\Gamma)) = TVal^{i_1}(I(\Gamma_1)) = \ldots = TVal^{i_k}(I(\Gamma_n)) = t$.

Note that this definition considers only those document formulas that are reachable via the one-argument import directives. Two argument import directives are not covered here. Their semantics is defined by the document RIF RDF and OWL Compatibility [RIF-RDF+OWL].

Also note that some of the $\Gamma_1$ above may be missing since all parts in a document formula are optional. In this case, we assume that $\Gamma_1$ is a tautology, such as $\text{And}()$, and every $TVal$ function maps such a $\Gamma_1$ to the truth value $t$.

For non-document formulas, we extend $TVal(I(\phi))$ from regular semantic structures to multi-structures as follows. Let $I = \{I, I; \ldots\}$ be a semantic multi-structure. Then $TVal(I(\phi)) = TVal(I(\phi))$.

The above definitions make the intent behind the rif:local constants clear: occurrences of such constants in different documents can be interpreted differently even if they have the same name. Therefore, each document can choose the names for the rif:local constants freely and without regard to the names of such constants used in the imported documents.

For the relationship between rif:local and RDF blank nodes readers are referred to Section Symbols in RIF Versus RDF/OWL (Informative) of [RIF-RDF+OWL].

3.6 Logical Entailment

We now define what it means for a set of RIF-BLD rules (embedded in a group or a document formula) to entail another RIF-BLD formula. In RIF-BLD we are mostly interested in entailment of RIF condition formulas, which can be viewed as queries to RIF-BLD documents. Entailment of condition formulas provides formal underpinning to RIF-BLD queries.

Definition (Models). A multi-structure $I$ is a model of a formula, $\phi$, written as $I \models \phi$, iff $TVal(I(\phi)) = t$. Here $\phi$ can be a document or a non-document formula.
Definition (Logical entailment). Let \( \varphi \) and \( \psi \) be (document or non-document) formulas. We say that \( \varphi \) entails \( \psi \), written as \( \varphi \models \psi \), if and only if for every multi-structure, \( \mathcal{I} \), for which both \( \text{TVaI}(\varphi) \) and \( \text{TVaI}(\psi) \) are defined, \( \mathcal{I} \models \varphi \) implies \( \mathcal{I} \models \psi \).

Note that one consequence of the multi-document semantics of RIF-BLD is that local constants specified in one document cannot be queried from another document. For instance, if one document, \( \Delta' \), has the fact "http://example.com/ppp"^^rif:iri("abc"^^rif:local) while another document formula, \( \Delta \), imports \( \Delta' \) and has the rule "http://example.com/qqq"^^rif:iri(?X) :- "http://example.com/ppp"^^rif:iri(?X), then \( \Delta \models "http://example.com/qqq"^^rif:iri("abc"^^rif:local) \) does not hold. This is because the symbol "abc"^^rif:local in \( \Delta' \) and \( \Delta \) is treated as different constants by semantic multi-structures.

This behavior of local symbols should be contrasted with the behavior of \( \text{rif:iri} \) symbols. Suppose, in the above scenario, \( \Delta' \) also has the fact "http://example.com/ppp"^^rif:iri("http://cde.example.org"^^rif:iri). Then \( \Delta \models "http://example.com/qqq"^^rif:iri("http://cde.example.org"^^rif:iri) \) does hold.

4 XML Serialization Syntax for RIF-BLD

The RIF-BLD XML serialization defines

- a normative mapping from the RIF-BLD presentation syntax to XML (Section Mapping from the Presentation Syntax to the XML Syntax), and
- a normative XML schema for the XML syntax (Appendix XML Schema for BLD).

Recall that the syntax of RIF-BLD is not context-free and thus cannot be fully captured by EBNF or XML Schema. Still, validity with respect to XML Schema can be a useful test. To reflect this state of affairs, we define two notions of syntactic correctness. The weaker notion checks correctness only with respect to XML Schema, while the stricter notion represents "true" syntactic correctness.

Definition (Valid BLD document in XML syntax). A valid BLD document in the XML syntax is an XML document that is valid with respect to the XML schema in Appendix XML Schema for BLD.

Definition (Admissible BLD document in XML syntax). An admissible BLD document in the XML syntax is a valid BLD document in XML syntax that is the
The XML serialization for RIF-BLD is based on an alternating or striped syntax [ANF01]. A striped serialization views XML documents as objects and divides all XML tags into class descriptors, called type tags, and property descriptors, called role tags [TRT03]. We follow the tradition of using capitalized names for type tags and lowercase names for role tags.

The all-uppercase classes in the presentation syntax, such as FORMULA, become XML Schema groups in Appendix XML Schema for BLD. They are not visible in instance markup. The other classes as well as non-terminals and symbols (such as Exists or =) become XML elements with optional attributes, as shown below.

RIF-BLD uses [XML1.0] for its XML syntax.

4.1 XML for the Condition Language

XML serialization of RIF-BLD in Section EBNF for RIF-BLD Condition Language uses the following elements.

- And (conjunction)
- Or (disjunction)
- Exists (quantified formula for 'Exists', containing declare and formula roles)
- declare (declare role, containing a Var)
- formula (formula role, containing a FORMULA)
- Atom (atom formula, positional or with named arguments)
- External (external call, containing a content role)
- content (content role, containing an Atom, for predicates, or Expr, for functions)
- Member (member formula)
- Subclass (subclass formula)
- Frame (Frame formula)
- object (Member/Frame role, containing a TERM or an object description)
- op (Atom/Expr role for predicates/functions as operations)
- args (Atom/Expr positional arguments role, with ordered="yes" attribute)
- instance (Member instance role)
- class (Member class role)
- sub (Subclass subclass role)
- super (Subclass super-class role)
- slot (Atom/Expr or Frame slot role, with ordered="yes" attribute, containing a TERM)
- Equal (prefix version of term equation ' = ')
- left (Equal left-hand side role)
- right (Equal right-hand side role)
The name of a prefix is not associated with an XML element, since it is handled via preprocessing as discussed in Section Mapping of the Condition Language.

The id and meta elements, which are expansions of the IRIMETA element, can occur optionally as the initial children of any Class element.

For the XML Schema definition of the RIF-BLD condition language see Appendix XML Schema for BLD.

The XML syntax for symbol spaces uses the type attribute associated with the XML element Const. For instance, a literal in the xs:dateTime datatype is represented as follows:

```xml
<Const type="&xs;dateTime">2007-11-23T03:55:44-02:30</Const>
```

RIF-BLD also uses the ordered="yes" attribute to indicate that the children of args and slot elements are ordered.

Example 6 (A RIF condition and its XML serialization).

This example illustrates XML serialization for RIF conditions. As before, the compact URI notation is used for better readability. Assume that the following prefix directives are found in the preamble to the document, whose XML form will be illustrated in Example 8:

```
Prefix(bks <http://example.com/books#>)
Prefix(cpt <http://example.com/concepts#>)
Prefix(curr <http://example.com/currencies#>)
Prefix(rif <http://www.w3.org/2007/rif#>)
Prefix(xs <http://www.w3.org/2001/XMLSchema#>)
```

RIF condition:

```
And (Exists ?Buyer (cpt:purchase(?Buyer ?Seller
cpt:book(?Author bks:LeRif)
current:USD(49)))
?Seller=?Author )
```
XML serialization:

```
<And>
  <formula>
    <Exists>
      <declare><Var>Buyer</Var></declare>
      <formula>
        <Atom>
          <op><Const type="&rif;iri">&cpt;purchase</Const></op>
          <args ordered="yes">
            <Var>Buyer</Var>
            <Var>Seller</Var>
            <Expr>
              <op><Const type="&rif;iri">&cpt;book</Const></op>
              <args ordered="yes">
                <Var>Author</Var>
                &bks;LeRif</Var>
              </Expr>
              <Expr>
                <op><Const type="&rif;iri">&curr;USD</Const></op>
                <args ordered="yes">49</Const></args>
              </Expr>
            </Expr>
          </args>
        </Atom>
      </formula>
    </exists>
  </formula>
  <Equal>
    <left><Var>Seller</Var></left>
    <right><Var>Author</Var></right>
  </Equal>
</And>
```

**Example 7** (An XML serialization of a RIF condition with a frame and a named-argument term).

This example illustrates XML serialization of RIF conditions that involve terms with named arguments. As in Example 6, we assume the following prefix directives, whose XML form will be illustrated in Example 8:

- `Prefix(bks <http://example.com/books#>)`
- `Prefix(cpt <http://example.com/concepts#>)`
- `Prefix(curr <http://example.com/currencies#>)`
- `Prefix(rif <http://www.w3.org/2007/rif#>)`
- `Prefix(xs <http://www.w3.org/2001/XMLSchema#>)`
RIF condition:


XML serialization:

<And>
<formula>
<Exists>
<declare><Var>Buyer</Var></declare>
<declare><Var>P</Var></declare>
<formula>
<And>
<formula>
<Member>
<instance><Var>P</Var></instance>
<class><Const type="&rif;iri">&cpt;purchase</Const></class>
</Member>
</formula>
<formula>
<Frame>
<object>
<Var>P</Var>
</object>
<slot ordered="yes">
<Const type="&rif;iri">&cpt;buyer</Const>
<Var>Buyer</Var>
</slot>
<slot ordered="yes">
<Const type="&rif;iri">&cpt;seller</Const>
<Var>Seller</Var>
</slot>
<slot ordered="yes">
<Const type="&rif;iri">&cpt;item</Const>
<Expr>
<op><Const type="&rif;iri">&cpt;book</Const></op>
</Expr>
</formula>
</Frame>
</formula>
</And>
4.2 XML for the Rule Language

We now extend the set of RIF-BLD serialization elements from Section XML for RIF-BLD Condition Language by including rules, along with their enclosing groups and documents, as described in Section EBNF for RIF-BLD Rule Language. The extended set includes the tags listed below. While there is a RIF-BLD element tag for the Import directive, there are none for the Prefix and Base directives: they are handled as discussed in Section Mapping of the RIF-BLD Rule Language.

- Document (document, containing optional directive and payload roles)
- directive (directive role, containing Import)
- payload (payload role, containing Group)
- Import (importation, containing location and optional profile)
- location (location role, containing ANYURICONST)
- profile (profile role, containing PROFILE)
Example 8 (Serializing a RIF-BLD document containing an annotated group).

This example shows a serialization for the document from Example 5. For convenience, the document is reproduced at the top and then is followed by its serialization.

Presentation syntax:

```
Document(
  Prefix(ppl  <http://example.com/people#>)
  Prefix(cpt  <http://example.com/concepts#>)
  Prefix(dc   <http://purl.org/dc/terms/>)
  Prefix(rif  <http://www.w3.org/2007/rif#>)
  Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
  Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
  Prefix(xs   <http://www.w3.org/2001/XMLSchema#>)

dc:date -> "2008-04-04"^^xs:date] *)

  Group
    And(cpt:perishable(?item)
      cpt:delivered(?item ?deliverydate ppl:John)
      cpt:scheduled(?item ?scheduledate)
      ?diffduration = External(func:subtract-dateTimes(?deliverydate
?scheduledate)
      ?diffdays = External(func:days-from-duration(?diffduration))
    External(pred:numeric-greater-than(?diffdays 10))))
  )

  Forall ?item ( cpt:reject(ppl:Fred ?item) :- cpt:unsolicited(?item)
  )

} )
```
XML syntax:

```xml
<!DOCTYPE Document [ 
  <!ENTITY ppl  "http://example.com/people#"> 
  <!ENTITY cpt  "http://example.com/concepts#"> 
  <!ENTITY dc   "http://purl.org/dc/terms/"> 
  <!ENTITY rif  "http://www.w3.org/2007/rif#"> 
  <!ENTITY func "http://www.w3.org/2007/rif-builtin-function#"> 
  <!ENTITY pred "http://www.w3.org/2007/rif-builtin-predicate#"> 
  <!ENTITY xs   "http://www.w3.org/2001/XMLSchema#"> 
]

<Document 
  xmlns="http://www.w3.org/2007/rif#" 
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 
  xmlns:xs="http://www.w3.org/2001/XMLSchema#"> 
  <payload> 
    <Group> 
      <id> 
        <Const type="&rif;iri">http://sample.org</Const> 
      </id> 
      <meta> 
        <Frame> 
          <object> 
            <Const type="&rif;local">pd</Const> 
          </object> 
          <slot ordered="yes"> 
            <Const type="&rif;iri">&dc;publisher</Const> 
          </slot> 
          <slot ordered="yes"> 
            <Const type="&rif;iri">http://www.w3.org/</Const> 
          </slot> 
        </Frame> 
      </meta> 
      <sentence> 
        <Forall> 
          <declare><Var>item</Var></declare> 
          <declare><Var>deliverydate</Var></declare> 
          <declare><Var>scheduledate</Var></declare> 
          <declare><Var>diffduration</Var></declare> 
          <declare><Var>diffdays</Var></declare> 
          <formula> 
            <Implies> 
              <if> 
                <And> 
                  <formula> 
                    
```
Atom
<op><Const type="&rif;iri">&cpt;perishable</Const></op>
<args ordered="yes"><Var>item</Var></args>
</Atom>

<Atom>
<op><Const type="&rif;iri">&cpt;delivered</Const></op>
<args ordered="yes">
<Var>item</Var>
<Var>deliverydate</Var>
<Const type="&rif;iri">&ppl;John</Const>
</args>
</Atom>

<Atom>
<op><Const type="&rif;iri">&cpt;scheduled</Const></op>
<args ordered="yes">
<Var>item</Var>
<Var>scheduledate</Var>
</args>
</Atom>

<Equal>
<left><Var>diffduration</Var></left>
<right>
<External>
<content>
<Expr>
<op><Const type="&rif;iri">&func;subtract-dateTimes</Const></op>
<args ordered="yes">
<Var>deliverydate</Var>
<Var>scheduledate</Var>
</args>
</Expr>
</content>
</External>
</right>
</Equal>

<Equal>
<left><Var>diffdays</Var></left>
<right>
<External>
<content>
</External>
</right>
</Equal>
\[
\texttt{\textit{Expr}} \\
\quad \texttt{\textit{op}}\texttt{\textit{const}}\texttt{type}\texttt{=\textit{rif:iri}}\texttt{\textit{func;days-from-duration}} \texttt{\textit{args ordered=\textit{yes}}} \\
\quad \texttt{\textit{Var}}\texttt{\textit{diffduration}}\texttt{\textit{Var}} \texttt{\textit{/Var}} \\
\quad \texttt{\textit{/args}} \\
\quad \texttt{\textit{/Expr}} \\
\quad \texttt{\textit{/content}} \\
\quad \texttt{\textit{/External}} \\
\quad \texttt{\textit{/right}} \\
\quad \texttt{\textit{/Equal}} \\
\quad \texttt{\textit{/formula}} \\
\quad \texttt{\textit{formula}} \\
\quad \texttt{\textit{External}} \\
\quad \texttt{\textit{content}} \\
\quad \texttt{\textit{Atom}} \\
\quad \texttt{\textit{op}}\texttt{\textit{const}}\texttt{type}\texttt{=\textit{rif:iri}}\texttt{\textit{pred;numeric-greater-than}} \\
\quad \texttt{\textit{args ordered=\textit{yes}}} \\
\quad \texttt{\textit{Var}}\texttt{\textit{diffdays}}\texttt{\textit{Var}} \\
\quad \texttt{\textit{Const type}\texttt{=\textit{xsd;integer}}\texttt{\textit{10}}\texttt{\textit{/Const}}} \\
\quad \texttt{\textit{/args}} \\
\quad \texttt{\textit{/Atom}} \\
\quad \texttt{\textit{/content}} \\
\quad \texttt{\textit{/External}} \\
\quad \texttt{\textit{/formula}} \\
\quad \texttt{\textit{/And}} \\
\quad \texttt{\textit{/if}} \\
\quad \texttt{\textit{then}} \\
\quad \texttt{\textit{Atom}} \\
\quad \texttt{\textit{op}}\texttt{\textit{const}}\texttt{type}\texttt{=\textit{rif:iri}}\texttt{\textit{&cpt;reject}}\texttt{\textit{/Const}} \\
\quad \texttt{\textit{args ordered=\textit{yes}}} \\
\quad \texttt{\textit{Const type}\texttt{=\textit{rif:iri}}\texttt{\textit{&ppl;John}}\texttt{\textit{/Const}}} \\
\quad \texttt{\textit{Var}}\texttt{\textit{item}}\texttt{\textit{Var}} \\
\quad \texttt{\textit{/Var}} \\
\quad \texttt{\textit{/args}} \\
\quad \texttt{\textit{/Atom}} \\
\quad \texttt{\textit{/then}} \\
\quad \texttt{\textit{Implies}} \\
\quad \texttt{\textit{/formula}} \\
\quad \texttt{\textit{/Forall}} \\
\quad \texttt{\textit{/sentence}} \\
\quad \texttt{\textit{sentence}} \\
\quad \texttt{\textit{Forall}} \\
\quad \texttt{\textit{declare}}\texttt{\textit{Var}}\texttt{\textit{item}}\texttt{\textit{Var}}\texttt{\textit{/Var}} \\
\quad \texttt{\textit{formula}} \\
\quad \texttt{\textit{Implies}} \\
\quad \texttt{\textit{if}} \\
\quad \texttt{\textit{Atom}} \\
\quad \texttt{\textit{op}}\texttt{\textit{const}}\texttt{type}\texttt{=\textit{rif:iri}}\texttt{\textit{&cpt;unsolicited}}\texttt{\textit{/Const}} \\
\quad \texttt{\textit{args ordered=\textit{yes}}} \\
\quad \texttt{\textit{Var}}\texttt{\textit{item}}\texttt{\textit{Var}} \\
\quad \texttt{\textit{/Var}} \\
\quad \texttt{\textit{/args}}
4.3 Mapping from the Presentation Syntax to the XML Syntax

This section defines a normative mapping, $\chi_{bld}$, from the presentation syntax to the XML syntax of RIF-BLD. The mapping is given via tables where each row specifies the mapping of a particular syntactic pattern in the presentation syntax. These patterns appear in the first column of the tables and the **bold-italic** symbols represent metavariables. The second column represents the corresponding XML patterns, which may contain applications of the mapping $\chi_{bld}$ to these metavariables. When an expression $\chi_{bld}(\text{metavar})$ occurs in an XML pattern in the right column of a translation table, it should be understood as a recursive application of $\chi_{bld}$ to the presentation syntax represented by the metavariable. The XML syntax result of such an application is substituted for the expression $\chi_{bld}(\text{metavar})$. A sequence of terms containing metavariables with subscripts is indicated by an ellipsis. For the subscript $m$ it is understood that $m \geq 1$, i.e. the ellipsis indicates at least one term. For the subscript $n$ it is understood that $n \geq 0$, i.e. the ellipsis indicates zero or more terms. A metavariable or a well-formed XML subelement is marked as optional by appending a bold-italic question mark, $?$, on its right.

4.3.1 Mapping of the Condition Language

The $\chi_{bld}$ mapping from the presentation syntax to the XML syntax of the RIF-BLD Condition Language is specified by the table below. Each row indicates a
Since the presentation syntax of RIF-BLD is context sensitive, the mapping must differentiate between the terms that occur in the position of individuals and the terms that occur as atomic formulas. To this end, in the translation table, the positional and named-argument terms that occur in the context of atomic formulas are denoted by expressions of the form \( \text{pred}(...) \) and the terms that occur as individuals are denoted by expressions of the form \( \text{func}(...) \). In the table, each metavariable for an (unnamed) positional argument \( i \) is assumed to be instantiated to values unequal to the instantiations of named arguments \( \text{unicodestring} -\rightarrow \text{filler} \). Regarding the last but first row, we assume that shortcuts for constants [RIF-DTB] have already been expanded to their full form ("..."^^symspace).

<table>
<thead>
<tr>
<th>Presentation Syntax</th>
<th>XML Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>And (</td>
<td>&lt;And&gt;</td>
</tr>
<tr>
<td>( \text{conjunct}_1 )</td>
<td>&lt;formula&gt;\text{xbld}(\text{conjunct}_1)&lt;/formula&gt;</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>&lt;formula&gt;\text{xbld}(\text{conjunct}_n)&lt;/formula&gt;</td>
</tr>
<tr>
<td>( \text{conjunct}_n )</td>
<td>&lt;/And&gt;</td>
</tr>
<tr>
<td>Or (</td>
<td>&lt;Or&gt;</td>
</tr>
<tr>
<td>( \text{disjunct}_1 )</td>
<td>&lt;formula&gt;\text{xbld}(\text{disjunct}_1)&lt;/formula&gt;</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>&lt;formula&gt;\text{xbld}(\text{disjunct}_n)&lt;/formula&gt;</td>
</tr>
<tr>
<td>( \text{disjunct}_n )</td>
<td>&lt;/Or&gt;</td>
</tr>
<tr>
<td>Exists ( variable(_1)</td>
<td>&lt;Exists&gt;</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>&lt;declare&gt;\text{xbld}(\text{variable}_1)&lt;/declare&gt;</td>
</tr>
<tr>
<td>( \text{variable}_n )</td>
<td>&lt;declare&gt;\text{xbld}(\text{variable}_n)&lt;/declare&gt;</td>
</tr>
<tr>
<td>( ( \text{premise} )</td>
<td>&lt;formula&gt;\text{xbld}(\text{premise})&lt;/formula&gt;</td>
</tr>
<tr>
<td>)</td>
<td>&lt;/Exists&gt;</td>
</tr>
<tr>
<td>External ( atomexpr</td>
<td>&lt;External&gt;</td>
</tr>
<tr>
<td>)</td>
<td>&lt;content&gt;\text{xbld}(\text{atomexpr})&lt;/content&gt;</td>
</tr>
<tr>
<td>)</td>
<td>&lt;/External&gt;</td>
</tr>
<tr>
<td>( \text{pred} ) (</td>
<td>&lt;Atom&gt;</td>
</tr>
<tr>
<td>( \text{argument}_1 )</td>
<td>&lt;op&gt;\text{xbld}(\text{pred})&lt;/op&gt;</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>&lt;args ordered=&quot;yes&quot;&gt;\text{xbld}(\text{argument}_1)&lt;/args&gt;</td>
</tr>
<tr>
<td>( \text{argument}_n )</td>
<td>\ldots</td>
</tr>
</tbody>
</table>
\[ \chi_{\text{bld}}(\text{argument}_n) \]
</args>
</Atom>

\[
\text{func} ( \\
\quad \text{argument}_1 \\
\quad \ldots \\
\quad \text{argument}_n \\
\) 
\]

<Expr>
<op>\chi_{\text{bld}}(\text{func})</op>
<args ordered="yes">
\chi_{\text{bld}}(\text{argument}_1) \\
\ldots \\
\chi_{\text{bld}}(\text{argument}_n) 
</args>
</Expr>

\[
\text{List} ( \\
\quad \text{element}_1 \\
\quad \ldots \\
\quad \text{element}_n \\
\) 
\]

<List>
\chi_{\text{bld}}(\text{element}_1) \\
\ldots \\
\chi_{\text{bld}}(\text{element}_n) 
</List>

\[
\text{List} ( \\
\quad \text{element}_1 \\
\quad \ldots \\
\quad \text{element}_m \\
\quad | \\
\quad \text{remainder} \\
\) 
\]

<List>
\chi_{\text{bld}}(\text{element}_1) \\
\ldots \\
\chi_{\text{bld}}(\text{element}_m) \\
<rest>\chi_{\text{bld}}(\text{remainder})</rest> 
</List>

\[
\text{pred} ( \\
\quad \text{unicodestring}_1 \rightarrow \text{filler}_1 \\
\quad \ldots \\
\quad \text{unicodestring}_n \rightarrow \text{filler}_n \\
\) 
\]

<Atom>
<op>\chi_{\text{bld}}(\text{pred})</op>
<slot ordered="yes">
<Name>\text{unicodestring}_1</Name> \\
\chi_{\text{bld}}(\text{filler}_1) \\
</slot>
\ldots \\
<slot ordered="yes">
<Name>\text{unicodestring}_n</Name> \\
\chi_{\text{bld}}(\text{filler}_n) \\
</slot>
</Atom>

\[
\text{func} ( \\
\quad \text{unicodestring}_1 \rightarrow \text{filler}_1 \\
\quad \ldots \\
\) 
\]

<Expr>
<op>\chi_{\text{bld}}(\text{func})</op>
<slot ordered="yes">
</slot>
</Expr>
### RIF Basic Logic Dialect

#### unicodestring and filler

<table>
<thead>
<tr>
<th>unicodestring&lt;sub&gt;n&lt;/sub&gt; -&gt; filler&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Name&gt;unicodestring&lt;/Name&gt;&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>X&lt;sub&gt;bl&lt;/sub&gt;(filler&lt;sub&gt;1&lt;/sub&gt;)</td>
</tr>
<tr>
<td>&lt;/slot&gt;</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>&lt;slot ordered=&quot;yes&quot;&gt;</td>
</tr>
<tr>
<td>&lt;Name&gt;unicodestring&lt;/Name&gt;&lt;sub&gt;n&lt;/sub&gt;</td>
</tr>
<tr>
<td>X&lt;sub&gt;bl&lt;/sub&gt;(filler&lt;sub&gt;n&lt;/sub&gt;)</td>
</tr>
<tr>
<td>&lt;/slot&gt;</td>
</tr>
<tr>
<td>&lt;/Expr&gt;</td>
</tr>
</tbody>
</table>

<p>| inst [ |
|   key&lt;sub&gt;1&lt;/sub&gt; -&gt; filler&lt;sub&gt;1&lt;/sub&gt; |
|   ... |</p>
<table>
<thead>
<tr>
<th>key&lt;sub&gt;n&lt;/sub&gt; -&gt; filler&lt;sub&gt;n&lt;/sub&gt; ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Frame&gt;</td>
</tr>
<tr>
<td>&lt;object&gt;X&lt;sub&gt;bl&lt;/sub&gt;(inst)&lt;/object&gt;</td>
</tr>
<tr>
<td>&lt;slot ordered=&quot;yes&quot;&gt;</td>
</tr>
<tr>
<td>X&lt;sub&gt;bl&lt;/sub&gt;(key&lt;sub&gt;1&lt;/sub&gt;)</td>
</tr>
<tr>
<td>X&lt;sub&gt;bl&lt;/sub&gt;(filler&lt;sub&gt;1&lt;/sub&gt;)</td>
</tr>
<tr>
<td>&lt;/slot&gt;</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>&lt;slot ordered=&quot;yes&quot;&gt;</td>
</tr>
<tr>
<td>X&lt;sub&gt;bl&lt;/sub&gt;(key&lt;sub&gt;n&lt;/sub&gt;)</td>
</tr>
<tr>
<td>X&lt;sub&gt;bl&lt;/sub&gt;(filler&lt;sub&gt;n&lt;/sub&gt;)</td>
</tr>
<tr>
<td>&lt;/slot&gt;</td>
</tr>
<tr>
<td>&lt;/Frame&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inst # class</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Member&gt;</td>
</tr>
<tr>
<td>&lt;instance&gt;X&lt;sub&gt;bl&lt;/sub&gt;(inst)&lt;/instance&gt;</td>
</tr>
<tr>
<td>&lt;class&gt;X&lt;sub&gt;bl&lt;/sub&gt;(class)&lt;/class&gt;</td>
</tr>
<tr>
<td>&lt;/Member&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub ## super</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Subclass&gt;</td>
</tr>
<tr>
<td>&lt;sub&gt;X&lt;sub&gt;bl&lt;/sub&gt;(sub)&lt;/sub&gt;</td>
</tr>
<tr>
<td>&lt;super&gt;X&lt;sub&gt;bl&lt;/sub&gt;(super)&lt;/super&gt;</td>
</tr>
<tr>
<td>&lt;/Subclass&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>left = right</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Equal&gt;</td>
</tr>
<tr>
<td>&lt;left&gt;X&lt;sub&gt;bl&lt;/sub&gt;(left)&lt;/left&gt;</td>
</tr>
<tr>
<td>&lt;right&gt;X&lt;sub&gt;bl&lt;/sub&gt;(right)&lt;/right&gt;</td>
</tr>
<tr>
<td>&lt;/Equal&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&quot;unicodestring&quot;^^symspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Const type=&quot;symspace&quot;&gt;unicodestring&lt;/Const&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?unicodestring</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Var&gt;unicodestring&lt;/Var&gt;</td>
</tr>
</tbody>
</table>
4.3.2 Mapping of the Rule Language

The $\chi_{\text{bld}}$ mapping from the presentation syntax to the XML syntax of the RIF-BLD Rule Language is specified by the table below. It extends the translation table of Section Mapping of the Condition Language. While the Import directive is handled by the presentation-to-XML syntax mapping, the Prefix and Base directives are not. Instead, these directives should be handled by expanding the associated shortcuts (compact URIs). Namely, a prefix name declared in a Prefix directive is expanded into the associated IRI, while relative IRIs are completed using the IRI declared in the Base directive. The mapping $\chi_{\text{bld}}$ applies only to such expanded documents. RIF-BLD also allows other treatments of Prefix and Base provided that they produce equivalent XML documents. One such treatment is employed in the examples in this document, especially Example 8. It replaces prefix names with definitions of XML entities as follows. Each Prefix declaration becomes an ENTITY declaration [XML1.0] within a DOCTYPE DTD attached to the RIF-BLD Document. The Base directive is mapped to the xml:base attribute [XML-Base] in the XML Document tag. Compact URIs of the form prefix:suffix are then mapped to &prefix;suffix.

<table>
<thead>
<tr>
<th>Presentation Syntax</th>
<th>XML Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document(</td>
<td>&lt;Document&gt;</td>
</tr>
<tr>
<td>Import( $loc_1$ prfl$_1$?)</td>
<td>&lt;directive&gt;</td>
</tr>
<tr>
<td>...</td>
<td>&lt;Import&gt;</td>
</tr>
<tr>
<td>Import( $loc_n$ prfl$_n$?)</td>
<td>&lt;location&gt;$\chi_{\text{bld}}(loc_1)$&lt;/location&gt;</td>
</tr>
<tr>
<td>group?</td>
<td>&lt;profile&gt;$\chi_{\text{bld}}(prfl_1)$&lt;/profile&gt;?</td>
</tr>
<tr>
<td>)</td>
<td>&lt;/Import&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/directive&gt;</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td>&lt;directive&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Import&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;location&gt;$\chi_{\text{bld}}(loc_n)$&lt;/location&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;profile&gt;$\chi_{\text{bld}}(prfl_n)$&lt;/profile&gt;?</td>
</tr>
<tr>
<td></td>
<td>&lt;/Import&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/directive&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;payload&gt;$\chi_{\text{bld}}(\text{group})$&lt;/payload&gt;?</td>
</tr>
<tr>
<td></td>
<td>&lt;/Document&gt;</td>
</tr>
<tr>
<td>Group(</td>
<td>&lt;Group&gt;</td>
</tr>
<tr>
<td>clause$_1$</td>
<td>&lt;sentence&gt;$\chi_{\text{bld}}(\text{clause}_1)$&lt;/sentence&gt;</td>
</tr>
<tr>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>clause$_n$</td>
<td>&lt;sentence&gt;$\chi_{\text{bld}}(\text{clause}_n)$&lt;/sentence&gt;</td>
</tr>
<tr>
<td>)</td>
<td>&lt;/Group&gt;</td>
</tr>
</tbody>
</table>
4.3.3 Mapping of Annotations

The $\chi_{bld}$ mapping from RIF-BLD annotations in the presentation syntax to the XML syntax is specified by the table below. It extends the translation tables of Sections Mapping of the Condition Language and Mapping of the Rule Language. The metavariable Typetag in the presentation and XML syntaxes stands for any of the class names And, Or, External, Document, or Group, and Quantifier for Exists or Forall. The dollar sign, $\$, stands for any of the binary infix operator names #, ##, =, or :, while Binop stands for their respective class names Member, Subclass, Equal, or Implies. Again, each metavariable for an (unnamed) positional argument is assumed to be instantiated to values unequal to the instantiations of named arguments unicodestring $\rightarrow$ filler.

<table>
<thead>
<tr>
<th>Presentation Syntax</th>
<th>XML Syntax</th>
</tr>
</thead>
</table>
| (* iriconst? frameconj? *) Typetag ( e₁ . . . eₙ ) | `<Typetag>` 
  `<id>$\chi_{bld}(iriconst)$</id>` 
  `<meta>$\chi_{bld}(frameconj)$</meta>` 
  `<e₁' . . . eₙ' />` 
  `</Typetag>` 
  where e₁', . . . , eₙ' are defined by $\chi_{bld}(Typetag(e₁ . . . eₙ)) = <$Typetag>$e₁' . . . eₙ'$</Typetag> |
| (* iriconst? frameconj? *) Quantifier variable₁ . . . variableₙ ( formula ) | `<Quantifier>` 
  `<id>$\chi_{bld}(iriconst)$</id>` 
  `<meta>$\chi_{bld}(frameconj)$</meta>` 
  `<variable₁ . . . variableₙ />` 
  `<formula>$\chi_{bld}(formula)$</formula>` 
  `</Quantifier>` |

```xml
<Forall> 
  <declare>$\chi_{bld}(variable₁)$</declare> 
  . . . 
  <declare>$\chi_{bld}(variableₙ)$</declare> 
  <formula>$\chi_{bld}(rule)$</formula> 
</Forall>

<Implies> 
  <if>$\chi_{bld}(condition)$</if> 
  <then>$\chi_{bld}(conclusion)$</then> 
</Implies>
```

Forall variable₁ . . . variableₙ ( rule )

conclusion :- condition
(* iriconst? frameconj? *)
pred {
  argument_1
  . . .
  argument_n
}

(* iriconst? frameconj? *)
func {
  argument_1
  . . .
  argument_n
}

(* iriconst? frameconj? *)
pred {
  unicodestring_1 -> filler_1
  . . .
  unicodestring_n -> filler_n
}

(* iriconst? frameconj? *)
func {
  unicodestring_1 -> filler_1
  . . .
unicodestringn \rightarrow \text{fillern} 

\begin{aligned}
\text{inst} &= \\
\text{key}_1 \rightarrow \text{filler}_1 \\
\ldots \\
\text{key}_n \rightarrow \text{filler}_n 
\end{aligned}

\begin{aligned}
\text{e}_1 \$ \text{e}_2 
\end{aligned}

\text{unicodestring}^{\text{\textbackslash symspace}}

\begin{aligned}
\text{?unicodestring} 
\end{aligned}
5 Conformance Clauses

RIF-BLD does not require or expect conformant systems to implement the RIF-BLD presentation syntax. Instead, conformance is described in terms of semantics-preserving transformations between the native syntax of a compliant system and the XML syntax of RIF-BLD.

Let $T$ be a set of datatypes and symbol spaces that includes the datatypes specified in [RIF-DTB], and the symbol spaces rif:iri, and rif:local. Suppose $E$ is a coherent set of external schemas that includes the built-ins listed in [RIF-DTB]. We say that a formula $\phi$ is a $BLD_{T,E}$ formula iff

- it is a well-formed BLD formula,
- all datatypes and symbol spaces used in $\phi$ are in $T$, and
- all externally defined terms used in $\phi$ are instances of external schemas from $E$.

A RIF processor is a conformant $BLD_{T,E}$ consumer iff it implements a semantics-preserving mapping, $\mu$, from the set of all $BLD_{T,E}$ formulas to the language $L$ of the processor ($\mu$ does not need to be an "onto" mapping).

Formally, this means that for any pair $\phi, \psi$ of $BLD_{T,E}$ formulas for which $\phi \models_{BLD} \psi$ is defined, $\phi \models_{BLD} \psi \text{ iff } \mu(\phi) \models_{L} \mu(\psi)$. Here $\models_{BLD}$ denotes the logical entailment in RIF-BLD and $\models_{L}$ is the logical entailment in the language $L$ of the RIF processor.

A RIF processor is a conformant $BLD_{T,E}$ producer iff it implements a semantics-preserving mapping, $\nu$, from the language $L$ of the processor to the set of all $BLD_{T,E}$ formulas ($\nu$ does not need to be an "onto" mapping).

Formally, this means that for any pair $\phi, \psi$ of formulas in $L$ for which $\phi \models_{L} \psi$ is defined, $\phi \models_{L} \psi \text{ iff } \nu(\phi) \models_{BLD} \nu(\psi)$.

An admissible document is one which conforms to all the syntactic constraints of RIF-BLD, including ones that cannot be checked by an XML Schema validator (cf. Definition Admissible BLD document in XML syntax).

The above definitions are specializations to BLD of the general conformance clauses defined in the RIF framework for logic dialects [RIF-FLD]. The following clauses are further restrictions that are specific to RIF-BLD.
RIF-BLD specific clauses

- Conformant BLD producers and consumers are required to support only the entailments of the form $\varphi \models_{BLD} \psi$, where $\psi$ is a closed RIF condition formula, i.e., a RIF condition in which every variable, $\exists V$, is in the scope of a quantifier of the form $\exists V$. In addition, conformant BLD producers and consumers should preserve all annotations where possible.

- A conformant RIF-BLD consumer is a conformant BLD $\Sigma, \Theta$ consumer in which $\Sigma$ consists only of the symbol spaces and datatypes, and $\Theta$ consists only of the external (function and predicate) schemas that are required by RIF-BLD. The required symbol spaces are $\text{rif:iri}$ and $\text{rif:local}$, and the required datatypes and externally defined schemas are those specified in [RIF-DTB]. A conformant RIF-BLD consumer must reject all inputs that do not match the syntax of BLD. If it implements extensions, it may do so under user control -- having a "strict BLD" mode and a "run-with-extensions" mode.

- A conformant BLD producer is a conformant BLD $\Sigma, \Theta$ producer, which produces documents that include only the datatypes and externals that are required by BLD.

Feature At Risk #3: Strictness Requirement

Note: This feature is "at risk" and may be removed from this specification based on feedback. Please send feedback to public-rif-comments@w3.org.

The two preceding clauses are features AT RISK. In particular, the "strictness" requirement is under discussion.

RIF-BLD supports a wide variety of syntactic forms for terms and formulas, which creates infrastructure for exchanging syntactically diverse rule languages. It is important to realize, however, that the above conformance statements make it possible for systems that do not support some of the syntax directly to still support it through syntactic transformations. For instance, disjunctions in rule premises can be eliminated through a standard transformation, such as replacing $p :- \text{Or}(q, r)$ with a pair of rules $p :- q, \ p :- r$. Terms with named arguments can be reduced to positional terms by ordering the arguments by their names and incorporating the ordered argument names into the predicate name. For instance, $p(ba->1 \ aa->2)$ can be represented as $p_{aa_{bb}}(2, 1)$. 
6 RIF-BLD as a Specialization of the RIF Framework for Logic Dialects [RIF-FLD]

This normative section describes RIF-BLD by specializing RIF-FLD. The reader is assumed to be familiar with RIF-FLD as described in RIF framework for logic dialects [RIF-FLD]. The reader who is not interested in how RIF-BLD is derived from the framework can skip this section.

6.1 The Presentation Syntax of RIF-BLD as a Specialization of RIF-FLD

This section defines the precise relationship between the presentation syntax of RIF-BLD and the syntactic framework of RIF-FLD.

The presentation syntax of the RIF Basic Logic Dialect is defined by specialization from the presentation syntax of the RIF Syntactic Framework for Logic Dialects described in [RIF-FLD]. Section Syntax of a RIF Dialect as a Specialization of the RIF Framework in [RIF-FLD] lists the parameters of the syntactic framework in mathematical English, which we will now specialize for RIF-BLD.

1. Extension points.

   All extension points of RIF-FLD are removed (specialized by replacing them with zero objects).

2. Alphabet.

   The alphabet of the RIF-BLD presentation syntax is the alphabet of RIF-FLD with the symbols Dialect, Neg, and Naf excluded.

3. Assignment of signatures to each constant and variable symbol.

   The signature set of RIF-BLD contains the following signatures:

   a. Basic
      - individual{ }
      - atomic{ }

      The signature individual{ } represents the context in which individual objects (but not atomic formulas) can appear.
      The signature atomic{ } represents the context where atomic formulas can occur.

   b. Signatures for lists
      - The signature list for closed lists. It has the following arrow expressions: () ⇒ individual,
c. Signatures for functions, predicates, and external functions and predicates

- Function signature \( f \). This signature has the arrow expressions for positional functions:
  \( () \Rightarrow \text{individual}, \text{(individual)} \Rightarrow \text{individual}, \text{(individual individual)} \Rightarrow \text{individual}, ... \),
  plus the arrow expressions for functions with named arguments:
  \( (s_1->\text{individual} \ldots s_k->\text{individual}) \Rightarrow \text{individual} \), for all \( k > 0 \) and all \( s_1, \ldots, s_k \in \text{ArgNames} \).

- Predicate signature \( p \). This signature has the arrow expressions for positional predicates:
  \( () \Rightarrow \text{atomic}, \text{(individual)} \Rightarrow \text{atomic}, \text{(individual individual)} \Rightarrow \text{atomic}, ... \),
  plus the arrow expressions for predicates with named arguments:
  \( (s_1->\text{individual} \ldots s_k->\text{individual}) \Rightarrow \text{atomic} \), for all \( k > 0 \) and all \( s_1, \ldots, s_k \in \text{ArgNames} \).

- External function signature \( ef \). This signature has the same arrow expressions as the signature \( f \).

- External predicate signature \( ep \). This signature has the same arrow expressions as the signature \( p \).

In the RIF-BLD specialization of RIF-FLD, the argument names \( s_1, \ldots, s_k \) must be pairwise distinct.

d. Every symbol in \( \text{Const} \) has exactly one signature of the form:
\( \text{individual}, f, p, ef, \text{or } ep \).

A constant cannot have the signature \( \text{atomic} \), since only complex terms can have such signatures. Thus, by itself a symbol, \( p \), cannot be a proposition in RIF-BLD, but a term of the form \( p() \) can.

According to the above, each constant symbol in RIF-BLD can be either an individual, a plain function, a plain predicate, an externally defined function, or an externally defined predicate. However, the same function symbol (plain or external) can occur with different numbers of arguments in different places. Similarly, predicate symbols can occur with different numbers of arguments. Note that such polyadic terms with the same function
or predicate symbol but different arities could be replaced by terms with fixed arities: the arity information could be encoded in the function or predicate names. For instance, the polyadic predicate terms \( p(?X) \) and \( p(?X, ?Y) \) could be represented, Prolog-style, as \( p/1(?X) \) and \( p/2(?X, ?Y) \), respectively, regarding the slash as part of the predicate names.

e. The constant symbols that correspond to RIF datatypes (XML Schema datatypes, `rdf:XMLLiteral`, `rdf:PlainLiteral`, etc.) all have the signature `individual` in RIF-BLD.

f. The symbols of type `rif:iri` and `rif:local` can have the following signatures in RIF-BLD: `individual`, `f`, `p`, `ef`, or `ep`.

g. All variables are associated with signature `individual`, so they can range only over individuals.

h. The signature for equality is \( =\{(\text{individual} \\text{individual}) \Rightarrow \text{atomic}\}. \)

This means that equality can compare only those terms whose signature is `individual`; it cannot compare predicate or function symbols. Equality terms are also not allowed to occur inside other terms, since the above signature implies that any term of the form \( t = s \) has signature `atomic` and not `individual`.

i. The frame signature, \( ->\), is \( ->\{(\text{individual} \\text{individual} \\text{individual}) \Rightarrow \text{atomic}\}. \)

Note that this precludes the possibility that a frame term might occur as an argument to a predicate, a function, or inside some other term.

j. The membership signature, \( \#\), is \( \#\{(\text{individual} \\text{individual}) \Rightarrow \text{atomic}\}. \)

Note that this precludes the possibility that a membership term might occur as an argument to a predicate, a function, or inside some other term.

k. The signature for the subclass relationship is \( \#\#\{(\text{individual} \\text{individual}) \Rightarrow \text{atomic}\}. \)

As with frames and membership terms, this precludes the possibility that a subclass term might occur inside some other term.

RIF-BLD uses no special syntax for declaring signatures. Instead, the rule author specifies signatures `contextually`. That is, since RIF-BLD requires that each symbol is associated with a unique signature, the signature is
determined from the context in which the symbol is used. If a symbol is used in more than one context, the parser must treat this as a syntax error. If no errors are found, all terms and atomic formulas are guaranteed to be well-formed. Thus, signatures are not part of the RIF-BLD language, and individual and atomic are not reserved keywords.

4. **Supported types of terms.**
   - RIF-BLD supports the following types of terms defined by the syntactic framework (see the Section Terms of [RIF-FLD]):
     a. constants
     b. variables
     c. positional
     d. with named arguments
     e. equality
     f. frame
     g. membership
     h. subclass
     i. external
   - Compared to RIF-FLD, terms (both positional and with named arguments) have significant restrictions in order to keep BLD relatively simple.
     - The signature for the variable symbols does not permit them to occur in the context of predicates, functions, or formulas. In particular, in the RIF-BLD specialization of RIF-FLD, a variable is not an atomic formula.
     - Likewise, a symbol cannot be an atomic formula by itself. That is, if \( p \in \text{Const} \) then \( p \) is not a well-formed atomic formula. However, \( p() \) can be an atomic formula. Note that Propositional Logic can thus be expressed in RIF-BLD using zero-argument predicate formulas.
     - Signatures permit only constant symbols to occur in the context of functions or predicate. Indeed, RIF-BLD signatures ensure that all variables have the signature \( \text{individual\{}()\}\) and all other terms, except for the constants from \( \text{Const} \), can have either the signature \( \text{individual\{}()\}\) or \( \text{atomic\{}()\}\). Therefore, if \( t \) is a (non-\( \text{Const} \)) term then \( t(...) \) is not a well-formed term.
     - In an externally defined term, \( \text{External}(t) \), \( t \) can be only a positional or a named-argument term. Compared to RIF-FLD, this restricts \( t \) so that it cannot be a constant, a frame, an equality, or a classification term. RIF-FLD's two-argument form of \( \text{External} \) is not supported in RIF-BLD either.

Combined with the fact that in a well-formed term of the form \( \text{External}(t) \) the subterm \( t \) must be an instance
of an external schema (by the definition of well-formed external terms in RIF-FLD), it follows that a predicate or a function symbol, p, that occurs in an external term External(p(...)) cannot also occur as a non-external symbol.

- If a term, t, is an instance of an externally defined schema from the coherent set of external schemas associated with the language, then t can occur only as External(t), i.e., as an external term or atomic formula.

5. No aggregate terms, module terms, or formula terms are allowed. (In RIF-FLD, formula terms correspond to compound formulas that involve logical connectives and quantifiers.)

6. Required symbol spaces.

RIF-BLD requires the symbol spaces defined in Section Constants, Symbol Spaces, and Datatypes of [RIF-DTB].

7. Supported formulas.

RIF-BLD supports the following types of formulas (see Well-formed Terms and Formulas in [RIF-FLD] for the definitions):

- **RIF-BLD condition**

  A RIF-BLD condition is an atomic formula, a conjunctive or disjunctive combination of atomic formulas, or an external atomic formula. All these can optionally have existential quantifiers.

- **RIF-BLD rule**

  A RIF-BLD rule is a universally quantified RIF-FLD rule with the following restrictions:

  - The conclusion of the rule is an atomic formula or a conjunction of atomic formulas.

    Note: This feature (Equality in the rule conclusion) is "at risk". See feature at risk #2

  - None of the atomic formulas mentioned in the rule conclusion is externally defined (i.e., cannot have the form External(...)).

  - The premise of the rule is a RIF-BLD condition.

  - All free (non-quantified) variables in the rule must be quantified with Forall outside of the rule (i.e., Forall ?vars (conclusion :- premise)).

- **Universal fact**
A universal fact is a universally quantified atomic formula with no free variables.

- **RIF-BLD group**

A RIF-BLD group is a RIF-FLD group that contains only RIF-BLD rules, universal facts, variable-free rule implications, variable-free atomic formulas, and RIF-BLD groups.

- **RIF-BLD document**

A RIF-BLD document is a RIF-FLD document that consists of directives and a RIF-BLD group formula. There is no Dialect or Module directives, and the Import(<loc>) directive (with one argument) can import RIF-BLD documents only. Here loc must be a Unicode character sequence that forms an IRI. There are no BLD-specific restrictions on the two-argument directive Import except that the second argument must be a Unicode sequence of characters of the form <loc>, where loc is an IRI.

Negation (whether classical or default) is not allowed in RIF-BLD rules: neither in rule conclusions nor in premises.

### 6.2 The Semantics of RIF-BLD as a Specialization of RIF-FLD

This normative section defines the precise relationship between the semantics of RIF-BLD and the semantic framework of RIF-FLD. Specification of the semantics that does not rely on RIF-FLD is given in Section Direct Specification of RIF-BLD Semantics.

The semantics of the RIF Basic Logic Dialect is defined by specialization from the semantics of the semantic framework for logic dialects of RIF. Section Semantics of a RIF Dialect as a Specialization of the RIF Framework in [RIF-FLD] lists the parameters of the semantic framework that can be specialized. Thus, for RIF-BLD, we need to look at the following parameters:

- **The effect of the syntax.**

  RIF-BLD does not support negation. This is the only obvious simplification with respect to RIF-FLD as far as the semantics is concerned. The restrictions on the signatures of symbols in RIF-BLD do not affect the semantics in a significant way.

- **Truth values.**
The set $TV$ of truth values in RIF-BLD consists of two values, $t$ and $f$, such that $f \preceq t$. The order $\preceq$ is total.

- **Datatypes.**

  RIF-BLD supports the datatypes listed in Section Datatypes of [RIF-DTB].

- **Logical entailment.**

  Recall that logical entailment in RIF-FLD is defined with respect to an unspecified set of intended semantic structures and that dialects of RIF must make this notion concrete. For RIF-BLD, this set is defined as the set of all models.

- **Import directive.**

  The semantics of the two-argument Import directive is given in [RIF-RDF+OWL]. The semantics of the one-argument directive is the same as in RIF-FLD.

### 6.3 The XML Serialization of RIF-BLD as a Specialization of RIF-FLD

Section Mapping from the RIF-FLD Presentation Syntax to the XML Syntax of [RIF-FLD] defines a mapping, $\chi_{\text{fld}}$, from the presentation syntax of RIF-FLD to its XML serialization. When restricted to well-formed RIF-BLD formulas, $\chi_{\text{fld}}$ coincides with the BLD-to-XML mapping $\chi_{\text{bld}}$. In this way, the XML serialization of RIF-BLD is a specialization of the RIF-FLD XML Serialization Framework defined in [RIF-FLD].

### 6.4 RIF-BLD Conformance as a Specialization of RIF-FLD

If $T$ is a set of datatypes and symbol spaces and $E$ a coherent set of external schemas for functions and predicates, then the general definition of conformance in RIF-FLD yields the notion of conformant BLD$_{T,E}$ producers and consumers.

BLD further requires strictness, i.e., that a conformant producer produces only the documents where $T$ are precisely the datatypes/symbol spaces and $E$ are the external schemas specified in [RIF-DTB], and that a conformant consumer consumes only such documents.

*Note: This feature (Strictness requirement) is "at risk". See feature at risk #3*
7 Acknowledgements

This document is the product of the Rules Interchange Format (RIF) Working Group (see below) whose members deserve recognition for their time and commitment. The editors extend special thanks to: Jos de Bruijn, Gary Hallmark, Stella Mitchell, Leora Morgenstern, Adrian Paschke, Axel Polleres, and Dave Reynolds, for their thorough reviews and insightful discussions; the working group chairs, Chris Welty and Christian de Sainte-Marie, for their invaluable technical help and inspirational leadership; and W3C staff contact Sandro Hawke, a constant source of ideas, help, and feedback.

The regular attendees at meetings of the Rule Interchange Format (RIF) Working Group at the time of the publication were: Adrian Paschke (Freie Universitaet Berlin), Axel Polleres (DERI), Changhai Ke (ILOG), Chris Welty (IBM), Christian de Sainte Marie (ILOG), Dave Reynolds (HP), Gary Hallmark (ORACLE), Harold Boley (NRC), Hassan Aït-Kaci (ILOG), Jos de Bruijn (FUB), Leora Morgenstern (IBM), Michael Kifer (Stony Brook), Mike Dean (BBN), Sandro Hawke (W3C/MIT), and Stella Mitchell (IBM). We would also like to thank past members of the working group, Allen Ginsberg, David Hirtle, Igor Mozetic, and Paula-Lavinia Patranjan.

8 References

8.1 Normative References

[RDF-CONCEPTS]  
Resource Description Framework (RDF): Concepts and Abstract Syntax,  
Klyne G., Carroll J. (Editors), W3C Recommendation, 10 February 2004,  

[RFC-3066]  
RFC 3066 - Tags for the Identification of Languages, H. Alvestrand, IETF,  

[RFC-3987]  

[RIF-Core]  
RIF Core Dialect  Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, Axel Polleres, Dave Reynolds, eds. W3C Working Draft, 3 July 2009,  
8.2 Informational References

[ANF01]  

[AG08]  
From SPARQL to rules (and back), Axel Polleres. *WWW 2007: 787-796*


9 Appendix: XML Schema for RIF-BLD

The namespace of RIF is http://www.w3.org/2007/rif#.

XML schemas for the RIF-BLD sublanguages are defined below and are also available here with additional examples.

9.1 Condition Language

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2007/rif#"
 targetNamespace="http://www.w3.org/2007/rif#"
 elementFormDefault="qualified"
 version="Id: BLDCond.xsd, v. 1.4, 2009-06-25, dhirtle/hboley">


<xs:annotation>
  <xs:documentation>
  This is the XML schema for the Condition Language as defined by
  </xs:documentation>
</xs:annotation>
```

[Steele90]  

[TRT03]  

[URD08]  

[vEK76]  
The schema is based on the following EBNF for the RIF-BLD Condition Language:

```
FORMULA ::= IRIMETA? 'And' (FORMULA+ ) |
           IRIMETA? 'Or' (FORMULA+ ) |
           IRIMETA? 'Exists' Var+ (FORMULA+ ) |
           ATOMIC |
           IRIMETA? 'External' (Atom )

ATOMIC ::= IRIMETA? (Atom | Equal | Member | Subclass | Frame)

Atom ::= UNITERM

UNITERM ::= Const '(' (TERM* | (Name -> TERM)* ) ')'

Equal ::= TERM '=' TERM

Member ::= TERM '#' TERM

Subclass ::= TERM '##' TERM

Frame ::= TERM '[' (TERM -> TERM)* ']' 

TERM ::= IRIMETA? (Const | Var | Expr | List | 'External' 'Ex'

Expr ::= UNITERM

List ::= 'List' '(' TERM* ')' | 'List' '(' TERM+ '|' TERM )'

Const ::= '^^' UNICODESTRING SYMSPACE | CONSTSHORT

Name ::= UNICODESTRING

Var ::= '?' UNICODESTRING

SYMSPACE ::= ANGLEBRACKIRI | CURIE

IRIMETA ::= '(' IRICONST? (Frame | 'And' ( Frame+ ) )? ')' 
```

```xml
<xs:documentation>
  <!-- sensitive to FORMULA (Atom) context-->
</xs:annotation>

<xs:group name="FORMULA">
  <!-- sensitive to FORMULA (Atom) context-->
</xs:choice>
</xs:complexType>
```

The Last Call Draft of the RIF Basic Logic Dialect.

http://www.w3.org/TR/2009/WD-rif-bld-20090703/
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element name="content" type="content-FORMULA.type"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="content-FORMULA.type">
<!-- sensitive to FORMULA (Atom) context-->
<xs:sequence>
  <xs:element ref="Atom"/>
</xs:sequence>
</xs:complexType>

<xs:element name="And">
<xs:complexType>
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Or">
<xs:complexType>
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Exists">
<xs:complexType>
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="declare" minOccurs="1" maxOccurs="unbounded"/>
  <xs:element ref="formula"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="formula">
<xs:complexType>
<xs:sequence>
  <xs:group ref="FORMULA"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="declare">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Var"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="ATOMIC">
<!--
ATOMIC ::= IRIMETA? (Atom | Equal | Member | Subclass | Frame)
-->  
  <xs:choice>
    <xs:element ref="Atom"/>
    <xs:element ref="Equal"/>
    <xs:element ref="Member"/>
    <xs:element ref="Subclass"/>
    <xs:element ref="Frame"/>
  </xs:choice>
</xs:group>

<xs:element name="Atom">
<!--
Atom ::= UNITERM
-->  
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="UNITERM" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="op"/>
      <xs:choice>
        <xs:element ref="args" minOccurs="0" maxOccurs="1"/>
        <xs:element name="slot" type="slot-UNITERM.type" minOccurs="0" maxOccurs="unbounded"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="UNITERM">
<!--
UNITERM ::= Const '(' (TERM* | (Name '->' TERM)*) ')' 
-->  
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="op"/>
    <xs:choice>
      <xs:element ref="args" minOccurs="0" maxOccurs="1"/>
      <xs:element name="slot" type="slot-UNITERM.type" minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:sequence>
</xs:group>

<xs:element name="op"/>
<xs:complexType>
  <xs:sequence>
    <xs:element ref="Const"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="args">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
  </xs:complexType>
</xs:element>
<xs:complexType name="slot-UNITERM.type">
  <!-- sensitive to UNITERM (Name) context-->
  <xs:sequence>
    <xs:element ref="Name"/>
    <xs:group ref="TERM"/>
  </xs:sequence>
  <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
</xs:complexType>
<xs:element name="Equal">
  <!-- Equal ::= TERM '=' TERM -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="left"/>
      <xs:element ref="right"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="left">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="right">
  <xs:complexType>
  </xs:complexType>
</xs:element>
<xs:sequence>
  <xs:group ref="TERM"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Member">
  <!--
  Member ::= TERM '#' TERM
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="instance"/>
      <xs:element ref="class"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Subclass">
  <!--
  Subclass ::= TERM '##' TERM
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="sub"/>  
      <xs:element ref="super"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="instance">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="class">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="sub">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="super">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Frame">
  <!--
  Frame ::= TERM '[' (TERM '->' TERM)* ']
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="object"/>
      <xs:element name="slot" type="slot-Frame.type" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="object">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="slot-Frame.type">
  <!-- sensitive to Frame (TERM) context-->
  <xs:sequence>
    <xs:group ref="TERM"/>
    <xs:group ref="TERM"/>
  </xs:sequence>
  <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
</xs:complexType>

<xs:group name="TERM">
TERM ::= IRIMETA? (Const | Var | Expr | List | 'External' '(' Expr ')')

-->  
  <xs:choice>
    <xs:element ref="Const"/>
    <xs:element ref="Var"/>
    <xs:element ref="Expr"/>
    <xs:element ref="List"/>
    <xs:element name="External" type="External-TERM.type"/>
  </xs:choice>
</xs:group>

<xs:element name="List">
  <!--
  List ::= 'List' '(' TERM* ')' | 'List' '(' TERM+ '|' TERM ')' 
  rewritten as
  List ::= 'List' '(' LISTELEMENTS? ')' 
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="LISTELEMENTS" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="LISTELEMENTS">
  <!--
  LISTELEMENTS ::= TERM+ ('|' TERM)?
  -->
  <xs:sequence>
    <xs:group ref="TERM" minOccurs="1" maxOccurs="unbounded"/>
    <xs:element ref="rest" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:group>

<xs:element name="rest">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="External-TERM.type">
  <!-- sensitive to TERM (Expr) context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element name="content" type="content-TERM.type"/>
  </xs:sequence>
</xs:complexType>
</xs:complexType>

<xs:complexType name="content-TERM.type">
  <!-- sensitive to TERM (Expr) context-->
  <xs:sequence>
    <xs:element ref="Expr"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="Expr">
  <!--
  Expr ::= UNITERM
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="UNITERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Const">
  <!--
  Const ::= '"' UNICODESTRING '"' SYMSPACE | CONSTSHORT
  -->
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="type" type="xs:anyURI" use="required"/>
    <xs:attribute ref="xml:lang"/>
  </xs:complexType>
</xs:element>

<xs:element name="Name" type="xs:string">
  <!--
  Name ::= UNICODESTRING
  -->
  </xs:element>

<xs:element name="Var">
  <!--
  Var ::= '?' UNICODESTRING
  -->
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
IRIMETA ::= '(*' IRICONST? (Frame | 'And' '(' Frame* ')')? '*')'

<xs:element name="id">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Const" type="IRICONST.type"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="meta">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="Frame"/>
      <xs:element name="And" type="And-meta.type"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:complexType name="And-meta.type">
  <!-- sensitive to meta (Frame) context-->
  <xs:sequence>
    <xs:element name="formula" type="formula-meta.type" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="formula-meta.type">
  <!-- sensitive to meta (Frame) context-->
  <xs:sequence>
    <xs:element ref="Frame"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="IRICONST.type" mixed="true">
  <!-- sensitive to location/id context-->
  <xs:sequence/>
  <xs:attribute name="type" type="xs:anyURI" use="required" fixed="http://www.w3.org/2007/rif#iri"/>
</xs:complexType>
9.2 Rule Language

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2007/rif#"
targetNamespace="http://www.w3.org/2007/rif#"
elementFormDefault="qualified"
version="Id: BLDRule.xsd, v. 1.4, 2009-06-25, dhirtle/hboley">
<xs:annotation>
<xs:documentation>
This is the XML schema for the Rule Language as defined by the Last Call Draft of the RIF Basic Logic Dialect.

The schema is based on the following EBNF for the RIF-BLD Rule Language:

Base     ::= 'Base' '(' ANGLEBRACKIRI ')
Prefix   ::= 'Prefix' '(' Name ANGLEBRACKIRI ')
Import   ::= IRIMETA? 'Import' '(' LOCATOR PROFILE? ')
Group    ::= IRIMETA? 'Group' '(' (RULE | Group)* ')
RULE     ::= (IRIMETA? 'Forall' Var+ '(' CLAUSE ')') | CLAUSE
CLAUSE   ::= Implies | ATOMIC
Implies  ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* ')') ':-' FORMULA
LOCATOR  ::= ANGLEBRACKIRI
PROFILE  ::= ANGLEBRACKIRI

Note that this is an extension of the syntax for the RIF-BLD Condition Language.</xs:documentation>
</xs:annotation>

<!-- The Rule Language includes the Condition Language from the same directory -->
<xs:include schemaLocation="BLDCond.xsd"/>

<xsl:element name="Document">
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="1" ref="IRIMETA"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="directive"/>
    </xs:sequence>
  </xs:complexType>
</xsl:element>
<xs:element name="directive">
  <!--
  Base and Prefix represented directly in XML
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Import"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="payload">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Group"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Import">
  <!--
  Import ::= IRIMETA? 'Import' '(' LOCATOR PROFILE? ')'  
  LOCATOR ::= ANGLEBRACKIRI 
  PROFILE ::= ANGLEBRACKIRI 
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="location"/>
      <xs:element ref="profile" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="location">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Const" type="IRICONST.type"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="profile">
<xs:complexType>
  <xs:sequence>
    <xs:element name="Const" type="IRICONST.type"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="Group">
  <!--
  Group ::= IRIMETA? 'Group' '(' (RULE | Group)* ')' 
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="sentence" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="sentence">
  <xs:complexType>
    <xs:choice>
      <xs:group ref="RULE"/>
      <xs:element ref="Group"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:group name="RULE">
  <!--
  RULE ::= (IRIMETA? 'Forall' Var+ ' (' CLAUSE ')') | CLAUSE 
  -->
  <xs:choice>
    <xs:element ref="Forall"/>
    <xs:group ref="CLAUSE"/>
  </xs:choice>
</xs:group>

<xs:element name="Forall">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="declare" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:element>
</xs:complexType>
</xs:element>

<xs:group name="CLAUSE">
<!--
CLAUSE ::= Implies | ATOMIC
-->
<xs:choice>
<xs:element ref="Implies"/>
<xs:group ref="ATOMIC"/>
</xs:choice>
</xs:group>

<xs:element name="Implies">
<!--
Implies ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* ')') ':-' FORMULA
-->
<xs:complexType>
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="if"/>
<xs:element ref="then"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="if">
<xs:complexType>
<xs:sequence>
<xs:group ref="FORMULA"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="then">
<xs:complexType>
<xs:choice>
<xs:group ref="ATOMIC"/>
<xs:element name="And" type="And-then.type"/>
</xs:choice>
</xs:complexType>
</xs:element>

<xs:complexType name="And-then.type">
<!-- sensitive to then (ATOMIC) context-->
10 Appendix: Changes Since the Last Call Version (2008-07-30)

This section summarizes the main changes to this document since the "last call" public snapshot of July 30, 2008.

- The definition of entailment in Section Logical Entailment relied on the notion of a "query document," which was not defined. The definitions in Sections Interpretation of Documents and Logical Entailment has been changed to eliminate this and related problems.
- Section EBNF Grammar for the Presentation Syntax of RIF-BLD now presents the EBNF of the entire language in one place. Previously the EBNF was given forst for the condition sublanguage and then repeated again when the entire rule language is defined.
- Symbols can now have many different arities.
- Lists have been added.
- Import: the first argument is now a character sequence that forms and IRI.
- Base, Prefix, Import: IRIs are now delimited with angle brackets.
- Numerous clarifications and explanations were added in response to the public comments.
- A number of typos were found and fixed.