

Guidelines to Writing Conformance Clauses for OASIS Specifications

Rev 28

12 December, 2016

Guideline URIs:

Editors):

Patrick Durusau

Ashok Malhotra

Jacques Durand

Abstract:

This document provides guidelines on how to write conformance statements for OASIS specifications. The target audience is primarily specification writers and TC members.

Status:

This document is not yet approved. The template is only being used as an editing and review convenience, and will not be used when publishing to the wiki.

Interested parties should send comments on this specification to the TAB by using the “Send A Comment” button on the TAB’s [web page](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tab.)

**Table of Contents**

[1 Non-Normative References 3](#_Toc468969216)

[2 Introduction 4](#_Toc468969217)

[3 Terms and Definitions 5](#_Toc468969218)

[4 Normative Content 6](#_Toc468969219)

[4.1 About Normative Content 7](#_Toc468969220)

[4.1.1 Examples of Normative Content 9](#_Toc468969221)

[4.2 Writing Normative Content 10](#_Toc468969222)

[4.3 Conformance Clauses vs. Normative Content 12](#_Toc468969223)

[4.4 Referring to Normative Content External to the Specification 13](#_Toc468969224)

[4.5 Considerations for Specifications that are not Implemented in Computer Code 14](#_Toc468969225)

[5 Conformance Section and Clauses 15](#_Toc468969226)

[5.1 Identification 16](#_Toc468969227)

[5.2 Conformance Target(s) 17](#_Toc468969228)

[5.3 References to Normative Content 18](#_Toc468969229)

[5.4 Related Conformance Clauses 19](#_Toc468969230)

[5.5 Variable Conformance Clauses 21](#_Toc468969231)

[6 Ten Rules for writing Conformance Clauses 23](#_Toc468969232)

[7 Examples of Conformance Clauses 25](#_Toc468969233)

[7.1 A Simple Example of Good Conformance Clauses: The Baseball Standard 25](#_Toc468969234)

[7.1.1 Bases/Home Plate 25](#_Toc468969235)

[7.1.3.2 Ninety Foot Base Paths 26](#_Toc468969236)

[7.1.4 Baseball Diamond 26](#_Toc468969237)

[7.1.5.2 College Baseball Diamond 26](#_Toc468969238)

[7.2 Good Conformance Clauses from the MQTT OASIS Standard 27](#_Toc468969239)

[7.3 Good Conformance Clauses from the XML standard 29](#_Toc468969240)

[7.4 Examples of Bad Conformance Clauses 32](#_Toc468969241)

[7.5 Examples of Related Conformance Clauses 34](#_Toc468969242)

[8 Appendix A: Conformance Clause Tutorial 35](#_Toc468969243)

# Non-Normative References

[ISO/IEC Part 2] [ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards (2011, 6th edition)](http://isotc.iso.org/livelink/livelink?func=ll&objId=10562502&objAction=Open&nexturl=%2Flivelink%2Flivelink%3Ffunc%3Dll%26objId%3D4230456%26objAction%3Dbrowse%26sort%3Dsubtype)

[OASIS] [Conformance Requirements for Specifications v1.0](https://www.oasis-open.org/apps/org/workgroup/ioc/download.php/305/conformance_requirements-v1.pdf)

[QAFRAME-SPEC] QA Framework: Specification Guidelines , K. Dubost, L. Rosenthal,

D. Hazaël-Massieux, L. Henderson, Editors, W3C Recommendation, 17 August 2005,

<http://www.w3.org/TR/2005/REC-qaframe-spec-20050817/> .

Latest version available at <http://www.w3.org/TR/qaframe-spec/>

[RFC2119] Bradner, S.,[Key words for use in RFCs to Indicate Requirement Levels,](http://www.ietf.org/rfc/rfc2119.txt) S., BCP 14, RFC 2119, March 1997.

[WSI-Profile] [Basic Profile Version 1.0](http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html)

# Introduction

As of 1 June 2007, the [OASIS TC Process](http://www.oasis-open.org/committees/process.php) requires that each specification contain a separate conformance section containing “numbered conformance clauses, to which any implementation of the specification must adhere in order to claim conformance to the specification (or any optional portion thereof)." This document provides guidelines on how to write conformance clauses (statements) for OASIS specifications. While it is not a requirement to follow these guidelines, it is recommended that TC adopt the advice herein in order to achieve consistency across OASIS specifications. The topic of conformance testing, covering validation of implementations with respect to a specification, is not covered in these guidelines**.**

This document describes the purpose and scope of conformance clauses, and provides guidelines for writing conformance clauses. Wherever possible, sample text and examples are provided.

Sample text has a pale yellow background

and

recommended text has a blue-grey background

The information contained is produced as the result of extensive experience by OASIS Staff and TAB Members in the writing and reviewing of specifications, and draws upon guidelines and requirements from [ISO/IEC Part 2], [OASIS], [QAFRAME-SPEC], [RFC2119], [WSI-Profile] which are included in the [References](#_References) section.

# Terms and Definitions

For the purposes of this document and specifications following its guidance, the following terms and definitions apply:

**CONFORMANCE:** The fulfillment of specified requirements by an implementation – i.e. a product, document, process, or service.

**CONFORMANCE CLAIM:** – A declaration that a product or artifact meets the requirements of one or more conformance clauses defined in the standard.

**CONFORMANCE CLAUSE:** A statement in the Conformance section of a specification that that defines precisely how an implementation can conform to a specification and references one or more portions of normative content, directly or indirectly, and may refer to other conformance clauses.

**CONFORMANCE TARGET:** An implementation which is the subject of at least one conformance clause.

**IMPLEMENTATION (of a specification):** – A concrete realization of a specification such as:

* an artifact, such as a protocol, document, process definition or service interface, or
* a processor that interprets such an artifact, or
* a performance that involves the interpretation or
* a combination thereof.

The implementation is expected to realize the requirements of some or all of the applicable normative content of a specification

*For example, the implementation of a messaging specification could be a message handler that will be a commercial product or simply a recognized software component. This messaging specification could also define two kinds of implementation: message receiver and message sender. These have to be understood as two roles with different requirements from the same specification. An actual software component could support both roles and therefore qualify as an implementation of both kinds.*

NORMATIVE CONTENT: Content of a specification that specifies prescriptive requirements or options for an implementation. Content of any nature - textual or not - qualifies as “normative” if stated so by a specification.

NON-NORMATIVE CONTENT: Content of a specification that does not specify prescriptive requirements or options for an implementation. Content of any nature - textual or not - qualifies as “non-normative” if stated so by a specification. Examples are generally non-normative.

# Normative Content

## About Normative Content

A specification broadly consists of non-normative and normative content. The non-normative content provides background information, context and examples. Normative content defines what an implementation – or some part of it – does in order to fulfill a part of the specification.

Normative Statements use the RFC2119 keywords or the ISO keywords if those have been chosen instead). Descriptive text MUST NOT use these reserved words in order to avoid confusing the reader as to what text is descriptive and what text is prescriptive.

The following example is taken from the [WS-BPEL specification](http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html#_Toc164738482):

*WS-BPEL supports extensibility by allowing namespace-qualified attributes to appear on any WS-BPEL element and by allowing elements from other namespaces to appear within WS-BPEL defined elements. This is allowed in the XML Schema specifications for WS-BPEL.*

*Extensions are either mandatory or optional (see section14). … In the case of mandatory extensions not supported by a WS-BPEL implementation, the process definition MUST be rejected. Optional extensions not supported by a WS-BPEL implementation MUST be ignored.*

The first paragraph in the sample is non-normative and provides background information on how to extend the WS-BPEL language.

The second paragraph contains Normative content and two Normative Statements that direct implementers on what a WS-BPEL implementation must do with unknown extensions. *If* an extension is mandatory and it is not supported by the implementation, *then* the implementation MUST reject the process definition. *If* the extension is optional and it is not supported, *then* the implementation MUST ignore the extension. These two statements provide clear, unambiguous, testable directions to implementers of the specification.

Normative content forms the core of a specification. It is essential that it be clear, concise, and unambiguous. Good normative content should enable yes-or-no questions about whether an implementation satisfies the requirements expressed by the content. A single specification can define multiple implementations with different capabilities or limitations. Where multiple implementations are defined, it must be clear which requirements apply to each implementation. (Different implementations may share some of the same requirements.)

Because implementations are the target of requirements implementations should be identified before normative content defining requirements for those implementations in a specification. This is especially important when a specification defines more than one kind of implementation. For example, the WS-BPEL example above identifies two implementations: a processor and the artifact that this processor interprets. A brief functional definition of the WS-BPEL processor and the artifact should be given early on in the specification.

Another example is a specification addressing a client / server architecture. The specification will likely define one implementation type to be a server and another to be a client (even if both can be found in the same software component).

Normative content must be referenceable so that it can be referred to from other parts of a specification, from the Conformance Clauses, and from outside the specification. There are no hard and fast rules for division of normative content but at a minimum, the normative content referred to by each conformance clause should be unambiguously identifiable from such clauses.

Normative content may be inconsistent, such as when defining alternative capabilities or limitations of different implementations, but it should clearly separate inconsistencies for the convenience of readers and implementers.

### Examples of Normative Content

The following example is taken from the Emergency management specification: <http://docs.oasis-open.org/emergency/edxl-de/v1.0/EDXL-DE_Spec_v1.0.pdf>

In the discussion on representing longitude and latitude the following Normative Statement is made:

*Latitudes north of the equator MAY be specified by a plus sign (+), or by the absence of a minus sign (-), preceding the designating degrees. Latitudes south of the Equator MUST be designated by a minus sign (-) preceding the digits designating degrees. Latitudes on the Equator MUST be designated by a latitude value of 0.*

This Normative Statement uses RFC2119 wording. Note that in this example, the implementation that is the target of this statement is not clearly identified in the Normative Content. We assume that it is clear to the reader that the implementation is known from context in which the Statement is made. For example, this Statement may be in a section explicitly labeled with the name of the implementation.

The following example is an example of poorly written Normative Content.

*When processing a document some features can be ignored and not displayed.*

It is not clear what features can be ignored. If the list were not made clear elsewhere in the context where the Content was written, it would need to be qualified.Also, the implementation is not defined. A better phrasing for the Normative Content would be:

*A word processor MAY ignore the following features contained within a document and MUST NOT display those features it ignores: …list of features.*

## Writing Normative Content

Now that we can distinguish normative from non-normative content, the next question is: How do we write normative content?

Many people, following the RFC tradition, insert in the body of their specification, requirements material.

Here are some examples of rewriting normative text so it can be referenced by

a conformance clause.

Each Client Registry Exchange (CLIENT-REGISTRY) repository MAY contain *original records* and *clone records*.

The above normative statement appears to be weak and to describe an optional feature, while, in fact, the statement is meant to say that the repository MUST contain either kind of records. “MAY” is actually meant as “is (only) allowed to”, i.e. a strong requirement, not an option.

This is a typical example of misuse of the “MAY” keyword. The use of MAY prevents separate conformance clauses for repositories that contain only original records, only clone records or than can contain both. Finally, the use of ‘and’ to indicate alternatives covered by a same normative keyword, is imprecise: do we mean here that both *original records* and *clone records* must be found in a repository? Or either one?

Suggest:

A Client Registry Exchange (CLIENT-REGISTRY) repository contains *original records,* *clone records,* or both.

Note that the rewriting takes away the implied requirement that repositories contain both original and clone records.

Whenever a CLIENT-REGISTRY repository adds a new original record or clone record, it MUST set the entry\_date field to the current time. This time value MUST never decrease as records are added. A client can incrementally update its copy of a repository by querying for all records with an entry\_date greater than or equal to the entry\_date of the last received record.

Suggest:

When a CLIENT-REGISTRY repository adds a new original record or clone record, it sets the entry\_date field to the current time.

Note: A client can incrementally update its copy of a repository by querying for all records with an entry\_date greater than or equal to the entry\_date of the last received record.

The final example reads:

If present, the expiry\_date field indicates when a record should be deleted to preserve the privacy of the personal information it contains. Conforming CLIENT-REGISTRY implementations MUST meet the following requirements:

Here “should” is not a keyword; it is in lower case, and because of that we can consider the statement as non-normative and should be clearer about that – e.g. put it in a note.

Suggest:

Note: If present, the expiry\_date field indicates when a record should be deleted to preserve the privacy of the personal information it contains.

and remove “Conforming CLIENT-REGISTRY implementations MUST meet the following requirements:” because it makes part of the conformance clause appear separate from other conformance clauses in the document.

With normative text that declares data types, operations, profiles, conformance clauses can then mix and match those declarative sections to define clients, servers, services with multiple levels and the like. Editors and more importantly readers, no longer have to worry about missing a requirement and being incompatible with other conforming implementations.

## Conformance Clauses vs. Normative Content

Normative content is not a substitute for conformance clauses. Conformance clauses are explicit about the features (“what”) a conformance target must implement, while normative content (with or without keywords) is often more focused on “how” a feature must operate or behave, Adding separate conformance clauses to a specification allows the editor to separate the normative logic from practical requirements for implementations, which may vary depending on conformance levels or profiles.

Normative content in the body of a specification must never mention “conformance”, e.g. must not use expressions such as “In order to conform, an implementation MUST….” Or “The following section describes conformance requirements for…”. Only a conformance clause can do this.

An editor can use direct, declarative prose to specify normative content of a work product. Then, in the conformance section the editors can define related conformance clauses to say, for example:

CC1. In order to claim CC1 conformance a server implementation MUST follow the definitions set forth in Section 1

CC2. A conforming server MUST follow CC1 and the server definitions in Section 2.

CC3. A conforming client MUST follow CC1 and the client definitions in Section 3.

The requirement for conformance clauses in Section 2.18 of the OASIS TC Process makes the editor's task easier at OASIS than at other organizations.

First, an editor can use normative versus non-normative text to specify portions of a work product that must be followed by those who claim conformance versus those that are simply commentary.

Second, the editor can simply make declarative statements that define elements, attributes, data types, return values, etc. Those do need to occur in the normative portion of the text but do not necessarily use RFC 2119 keywords. Later, a conformance clause can reference that normative text as the definitions to be followed in order to claim conformance.

RFC 2119 keyword were designed to be used in RFCs which, by their very nature, do not have formal conformance clauses. Therefore, OASIS conformance clause that use RFC 2119 keywords must include additional information in order to create usable conformance clauses.

Third, when an editor or the TC wants to offer some observation or comment that need not be accepted by an implementer, such as: "The TC thinks UML diagrams aid in the understanding of this specification." then it can do so without an implementer thinking that is required for conformance.

## Referring to Normative Content External to the Specification

When some normative content required for implementing a specification (A) belongs to another, pre-existing, specification (B), (B) should be referred to in the "Normative references" section of (A).

In such a case, a conformance clause for (A) that requires implementation of [parts of] an external specification (B), should state the expected level of conformance to (B) either by explicitly referring at the normative content in (B) that is to be complied with, or by referring to conformance clause(s) of (B) if any.

## Considerations for Specifications that are not Implemented in Computer Code

For specifications such as business process descriptions or reference models that are not meant to be implemented in computer code, precision in normative content is even more important. When a human being is partially or fully responsible for carrying out the Normative portions of a specification, even greater care must be taken to ensure that the Normative Content is sufficiently detailed, precise and unambiguous in its description to enable yes-or-no questions about whether the activities of a performer of the specification satisfy the requirements expressed by the content. An auditor should be able to judge unequivocally whether requirements have been satisfied.

Consider the following example of a business process Normative Statement:

*Before the administrator grants access to the system, the administrator MUST perform a risk assessment, determine the level of risk from granting access, and adopt a risk mitigation strategy.*

(In this example, the implementation is an administrative process for granting access to a system or maybe an even broader process of which granting system access is just a part, for example the on-boarding of a new employee. The administrator is just a piece of this process – an agent with the power to grant or deny access to a system, and to which the above requirement applies. The situation at hand is that the administrator has been asked to grant access to a system. Three actions must be taken in order to conform to this part of the specification:

* A risk assessment must be performed;
* The level of risk must be determined; and
* A risk mitigation strategy must be adopted.

In order for this Normative Statement to be clear and yes-or-no testable, the specification needs clear definitions of: risk, risk assessment, and risk mitigation.

It is generally better to err on the side of caution when determining whether or not definition is needed. In the example on entering values for Latitude ([1.1.1](#1.1.1.Examples of Normative Content|outline)), the term Latitude is clear and unambiguous and needs no definition. In other cases, the definitions may be drawn from other sources that are listed in the Normative References section of the specification. For example:

* A risk assessment must be performed **[ISO 31000]**;

If terms are not industry standards or otherwise clearly and unambiguously understood by all the parties who wish to implement the specification, it is critical that they be defined in the specification. If they are not, it becomes impossible to state whether or not someone has complied with the requirements. Put another way, without the definitions, yes-or-no questions may be formulated but they cannot be answered.

This is a common weakness of specifications that are not meant to be implemented in computer code. Some of the notions are under-defined or left to open-interpretation by various parties that may not be consistent.

Consider the following example:

*A conforming program MUST demonstrate Stakeholder Collaboration by establishing and giving high priority and adequate resources to a formal managed stakeholder engagement program.*

In order to assess whether or not such a Normative Statement is satisfied, the terms “stakeholder,” “collaboration,” “high priority” and “adequate resources” must be defined. Either the specification provides the definition itself, or it must identify where or how the definition will be provided, for example by a reference to another specification or to an authoritative external body.

# Conformance Section and Clauses

Conformance clauses must be defined within a separate conformance section of an OASIS specification. The conformance section of a specification must contain at least one conformance clause.

The structure of the conformance section should be as follows (with “N” being the section number, typically the last section in the specification):

N. Conformance

n.1 Conformance targets

[A list of implementation types for which conformance can be claimed – i.e. list of conformance targets. A clear functional definition for each implementation type must be provided, unless such definitions are already provided in the specification body (preferred).]

n.2 Conformance Clause 1: <name>

…

n.3 Conformance Clause 2: <name>

…

n.4 Conformance Clause 3: <name>

…

## Identification

Each conformance clause must be uniquely labeled.

There may be more than one conformance clause in a specification, and like normative statements they must be clear, concise, and unambiguous. The [OASIS TC Process](https://www.oasis-open.org/policies-guidelines/tc-process) requires conformance clauses to be numbered in addition to having a meaningful name that identifies the conformance level or profile.

Example:

Conformance Clause 1: “basic”

“A server implementation satisfies “basic” conformance level if.…”

Conformance Clause 2: “advanced”

“…”

## Conformance Target(s)

A conformance clause must clearly identify its conformance target. There is just one conformance target for each conformance clause. A specification may have several types of implementations for which it is desirable to claim conformance – e.g. a document, a server, a client… - and to each one of these implementation categories should correspond at least one conformance clause.

Example:

Conformance Clause 1: “reliable server”

“A server implementation satisfies “reliable server” conformance profile if.…”

In the example immediately above, “server implementation” is the conformance target that may or may not claim *reliable server* conformance profile. The notion of server (or server implementation) must have been functionally defined outside the conformance clause and in the body of the specification.

## References to Normative Content

A conformance clause identifies that to which a conformance target must conform and this is done by reference(s) to normative content in the specification. A conformance clause therefore must identify a sub-set of the normative content of a specification. There are several ways to identify and reference normative content:

* *Statement tagging*: If the normative content is made of statements, sentences or small paragraphs that can easily be construed as requirements, then these could be tagged by some ID and referred to by this ID. Such tagged statements should be non-ambiguous when taken out of context, or else should explicitly refer to their context. Thought should be put into the appropriate granularity of references to normative statements. Once tagged, it is easy to group many such statements into normative units that could be referenced in the conformance clauses instead.
* *Sections titles and numbers*: If the normative content that concerns a conformance target is well delineated by sections, it is then possible to directly refer to such sections in a conformance clause.

Example:

A server implementation satisfies “reliable server” conformance profile if it fulfills all mandatory requirements in sections 2 and 3 that concern server implementations.”

When referring to normative content, a clear statement must be made as to how optional normative statements (i.e. those using the MAY, SHOULD, SHOULD NOT, RECOMMENDED keywords) must be handled. This decision relates to the type of conformance target and the use of the specifications. For example a document that claims conformance to a schema does not have to use any optional features. However, in another scenario, a protocol implementation target should implement optional features in case another party using the protocol makes use of the optional features. In deciding how to dispose of optional features, issues that effect interoperability and portability need to be considered.

## Related Conformance Clauses

There may be several conformance clauses in a specification. They may be unrelated to each other, either because they affect different conformance targets, or because they concern very different functions of the specification. For example:

Case 1: **Distinct Functional Roles**. A specification may define distinct possible roles for an implementation that relate to distinct parts of the specification. For example, a messaging protocol specification may define (message) Sender and Receiver roles. An implementation may decide to act in just one role. Consequently, two distinct conformance profiles should be defined, one for each role, that are associated with two conformance clauses:

Conformance Clause 1: “Sender”

“A message handler implementation satisfies “Sender” conformance profile if…”

Conformance Clause 2: “Receiver”

“A message handler implementation satisfies “Receiver” conformance profile if…”

Such conformance clauses are independent from each other. Yet a single implementation may claim conformance to both, as it can be a conformance target for both – here acting as both Sender and Receiver.

However, it may be that the conformance clauses relate to each other. Some typical patterns are illustrated by the following use cases:

Case 2: **Aggregate clauses.** In the simplest case, a conformance clause aggregates other clauses

that have the same conformance target. For example, clause A is the union of clauses B and C.

Case 3: **Levels of conformance**. A conformance clause may build on top of another one. For example, clause B requires A and extends it with additional requirements (e.g. references to new normative statements) in addition to those defined by A. These clauses must have the same conformance target.

For example, the conformance section defines three levels of conformance: basic, standard, and advanced for an implementation. These levels typically build on top of each other by adding more requirements: “standard” adds more requirements than “basic”. Consequently, conforming at a “standard” level also implies conformance at “basic” level, and conformance at “advanced” level implies conformance to all lower levels (basic, standard). In such a case, the conformance clauses can refer to each other:

Conformance Clause 1: “basic”

“A server implementation satisfies “basic” conformance level if.…”

Conformance Clause 2: “standard”

“A server implementation satisfies “standard” conformance level if (a) it conforms at “basic” level, and (b)…”

Conformance Clause 3: “advanced”

“A server implementation satisfies “advanced” conformance level if (a) it conforms at “standard” level, and (b)…”

Case 4: **Combination of Roles**. Building on the Sender and Receiver conformance profiles illustrated in Case 1: it may be useful to define a conformance profile that combines both roles – e.g. “Sender-Receiver” - , so that an implementation can claim conformance to this single profile, instead of having to claim conformance to several profiles:

Conformance Clause 3: “Sender-Receiver”

“A message handler implementation satisfies “Sender-Receiver” conformance profile if it satisfies both the “Sender” profile and the “Receiver” profile.”

Case 5: **Accessory Qualification**. A conformance clause may use another clause in an accessory manner, i.e. not directly on its conformance target but instead on an accessory artifact e.g. to define a pre-condition to its applicability. These clauses typically have different conformance targets. For example, clause A has “processor” as conformance target, but relies on clause B with “document” conformance target, in order to qualify the documents expected to be processed by “processor” according to clause A.

A specification may define different implementations or conformance targets yet these targets may relate functionally to each other so that one target may need to assume that the other target is conforming. For example, a messaging protocol specification defines very different targets such as “processor” and “message”. The conformance requirements for “processor” include the ability to process messages correctly, but this only if these “messages” are themselves conforming. In the example below, “Clause 1 is used by Clause 2 in an accessory manner”

Conformance Clause 1: “Message”

“A data artifact satisfies “Message” conformance profile if…”

Conformance Clause 2: “Processor”

“An implementation satisfies “Processor” conformance profile if (a) it fulfills all normative statements in Sections… and if (b) it is able to process correctly according to semantics described in … any message that conforms to the “Message” profile (Clause 1).”

Case 6: **Alternate Clauses**. Clause A uses alternate clauses if it requires the conformance target to satisfy either clause B or clause C. These clauses must have the same conformance target.

For example, building on Case 3 (levels of conformance), a new conformance clause could be defined, called “German market ready”, defined as:

Conformance Clause 4: “German market ready”

“A user interaction module satisfies “German market ready” conformance level if:

* It is conforming at either “basic” (see Conformance Clause 1) or “standard” (see Conformance Clause 2) or “advanced” (see Conformance Clause 3) level.
* All text constants that appear in the GUI as well as in error messages are in German.

## Variable Conformance Clauses

In some cases, there are different ways to conform to the same conformance clause. Consider the following situations:

**Example 1**: A processor may have the option of generating different serialization formats for an artifact – e.g. XML, JSON, or YAML. In this case it would be impractical to define a distinct conformance clause for each option, let alone each combination of these.

**Example 2**: A specification for a server may define a set of services that an implementation can support in any combination. In this case, it may not be realistic to define a conformance clause for each one of the possible combinations of services.

In the above cases, the specification editor can define a conformance clause with variable requirements that specify the subset of options or features that must be implemented according to the specification.

In example 1, the conformance clause could be of the form:

*Conformance Clause 15: Biometric matching REST server*

*Variable: “serializationFormat”, values in { “XML”, “JSON”, “YAML”}.*

*“A server implementation conforms to this clause if it fulfills all mandatory requirements in sections 2.1 and 2.5 that concern server implementations, AND if it generates responses with a format within “serializationFormat.”*

*“If serializationFormat=”XML” or contains “XML” then the XML responses generated by the server MUST satisfy the XML schema <…>”.*

*“If serializationFormat=”JSON” or contains “JSON” then the JSON responses generated by the server MUST satisfy the rules defined in section 4 for JSON”.“If serializationFormat=”YAML” or contains “YAML” then the YAML responses generated by the server MUST satisfy the YAML schema <…>”.*

Note that “serializationFormat” is just a name for this variability dimension used in these conformance clauses. The name is just used in the conformance clause, not in the normative part of the specification.

A conformance claim for this conformance clause will be of the form:

*Our server product XYZ conforms as a “Biometric matching REST server”, with serializationFormat = {”JSON”}*

In example 2, the conformance clause would be of the form:

*Conformance Clause 22: ProjectServer*

*Variable: “services”, subsets in { “project”, “issue”, “riskAssessmment”, “reporter”, “assigned”, “due”, “status”… }.*

*“A server implementation conforms to this clause if it fulfills all mandatory requirements in sections 3.7and 5.1 that concern server implementations, AND if it supports a set of services within “services.”*

*“If services contains ”issue” then the server MUST satisfy the mandatory requirements in Section 3.7.1”.*

*“If services contains “reporter” or “assigned” then the server MUST satisfy the mandatory requirements in Section 5.1.2”.*

*…*

A conformance claim for this conformance clause will be of the form:

*Our server product XYZ conforms as a “ProjectServer”, with services = {project, issues, reporter, assigned }*

Variable conformance clauses often imply that there may be several variants of the same type of implementation depending on what features are supported (as specified by the variables). In such a case it is often important that:

The normative content of the specification defines a “discovery” feature that allows other interacting implementations or other interacting components, to discover at run-time (or when in operation) what are the supported features.

The normative content of the specification defines a proper error handling mechanism, that allows other interacting implementations or other interacting components, to get an acceptable notice that a feature is not supported, when attempting or expecting the use of such feature at run-time (or when in operation).

# Ten Rules for Writing Conformance Clauses

**Rule 1**: Each conformance clause must clearly identify its conformance target(s), i.e. the type of implementations it applies to.

**Do** refer to the conformance target, e.g. in the style: “A conforming Sender implementation is an implementation that satisfies normative statements….”, or “In order to conform to the profile ‘Sender’, a message handler MUST…”

**Do not** make abstraction of the implementation, e.g. by writing “conformance to this specification requires satisfying normative statements X, Y and Z… ”.

**Rule 2:** The different conformance targets – i.e. the types of implementation that are candidates for conformance – must be clearly defined ***prior*** to their association with conformance clauses.

**Do** define a conformance target in a functional way in the specification body, based on the role it plays or as a component that is well understood in the domain. (e.g. “*a server in this specification is a component able to respond to messages defined as requests*”)

**Do not** refer to the conformance clause when defining a conformance target, e.g. “*a server is an implementation that satisfies this conformance clause*”.

**Rule 3**: Each Conformance Clause must only concern one type of conformance target, or conformance targets types that are clearly related.

**Do** define a conformance clause for each conformance target, even if the clauses are very similar.

**Do not** define a conformance clause that conflates together targets that are functionally very different (e.g. as stating conformance requirements for both a “server” and a “client”, or for a “processor” and the processed “document”). Instead, use different conformance clauses specific to each target.

**Rule 4**: Conformance Clauses must be numbered within a conformance section, and associated with a named conformance profile or level - unless there is just one profile or level.

**Do** introduce a conformance clause with both a number and a name, e.g. as : “conformance clause 3: advanced server”, so that it is possible to either refer to a specific clause by its name, or to a group of related clauses by their numbers such as “Clauses from N to M”.

**Do not** give a name that is the same as a target name, e.g. “server” as this would lead to wordings such as “a server will conform as a server if…”. Also do not give to a clause a name that includes another clause name, e.g. “advanced large server” and “large server” unless an advanced large server is also conforming as a large server.

**Rule 5**: A Conformance Clause must refer precisely and unambiguously to normative statements and content in the specification main body.

**Do** refer to specific sections by their title AND number. Do refer to tables and figures by both name and number.

**Do not** use line numbers which are not reliable and may change from one format of the specification to the other. Do not use vague or relative references as “the third paragraph” or “the second bullet”.

**Rule 6:** A Conformance Clause must NOT introduce new normative statements that could instead have been part of the specification main body.

**Do** add new requirements in a clause only if they clearly pertain to conformance conditions – e.g. practical matters about an implementation such as some range of values that characterize a class of products, or other arbitrary requirements that clearly could change in another conformance clause.

**Do not** put in a conformance clause requirements that are essentially part of the specification logic and should be in the specification main body, because it is useful for developers to understand and design the core parts of an implementation regardless of conformance concerns.

**Rule 7**: A Conformance Clause must NOT refer to content in the specification main body that was qualified as non-normative.

**Do** always make sure that the referred material has clear normative indicators, such as normative keywords or a section introduced by a qualifying statement (“the tables and definitions in this section are normative…”)

**Do not** use or refer to examples in a conformance clause (like “ as illustrated in…”), nor refer to detailed material in non-normative Appendices.

**Rule 8:** A Conformance Clause may refer to a normative statement of optional nature (SHOULD, MAY), or group of these, and make it recommended or mandatory, but cannot do the reverse.

**Do** strengthen a “SHOULD” statement to a “MUST” or transform a “MAY” statement into a “SHOULD” if the prescription level of the normative statement is too weak for an advanced level of conformance.

**Do not** refer to a “MUST” statement to weaken it as a “SHOULD” statement for a lower level of conformance.

**Rule 9:** A Conformance Clause may add practical restrictions about how the conformance target is supposed to fulfill a normative requirement or function.

**Do** add in a conformance clause, requirements that are about practical limits or conditions - execution time, implementation size, volume of information or throughput – that would be out of place in the normative logic of the specification body because of the expected diversity of implementations.

**Rule 10:** When writing Conformance Clauses about implementations that interoperate with each other, place a higher bar on the ability for the conformance target to process what other implementations produce, even when being more lax on the same features for what the conformance target produces.

**Do** require the conformance target to be able to handle optional features from other implementations that interact with it - even if the production of such a feature remains optional for the conformance target – or at least require the target to “fail gracefully” (i.e. perform appropriate error handling and notification) when not supporting the feature.

# Examples of Conformance Clauses

## A Simple Example of Good Conformance Clauses: The Baseball Standard

**Preamble**: For the purpose of illustrating the role of a conformance clause, the following description of a baseball field can be seen as a normative specification. This specification gives normative definitions of various parts of the field. These normative definitions represent requirements that an actual implementation of the specification must satisfy.

### Those implementations that are of interest to the users of the specification should always be clearly identified. They will be subject to conformance requirements as described in conformance clauses, and called *conformance targets*. Bases/Home Plate

#### 7.1.1.1 Bases

Bases are squares, 15 inches on each side and 3 to 5 inches thick. There are three bases in a baseball diamond.

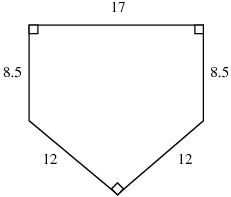
#### 7.1.1.2 Home Plate

Home plate is an irregular pentagon with two parallel sides, each perpendicular to a base.

A home plate is 17 inches in width with the front edge (continuous material out of which the base is constructed) facing the pitcher’s mound.

The sides proceed from the front edge, parallel to each other, for a distance of 8.5 inches.

At 8.5 inches from the front edge, both edges proceed to intersect at the middle of the 17 inch width, forming sides that are 12 inches in length.



*Illustration 1. Non-normative diagram of a Home Plate*

There is one home plate in a baseball diamond.

#### 7.1,2 The Base Path

The base path of a baseball diamond consists of four *segment*s:

1. *Home plate* to *first base*.
2. *First base* to *second base*.
3. *Second base* to *third base*.
4. *Third base* to *home plate*.

(Terms in *italic font* are defined elsewhere in the standard.)

#### Base Path Lengths

#### 7.1.3.1 Sixty Foot Base Paths

A base path composed of sixty foot segments.

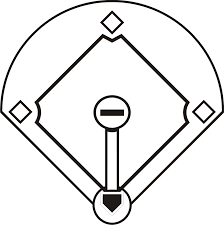
### 7.1.3.2 Ninety Foot Base Paths

A base path composed of ninety foot segments.

### 7.1.4 Baseball Diamond

The baseball diamond is the name given to the baseball field that comprises the infield area, the base path and its elements, and other field items such as the pitcher’s mound and catcher’s box.

An illustrative layout of a baseball infield (non-normative):



*Illustration 2 Baseball Diamiond*

#### 7.1.5 **Conformance**

**Note**: This specification defines several field components and their relationships: base, home plate, base path, baseball diamond. Each one of these has its own requirements and could be considered and built separately. Consequently, the case could be made that each one of these components can be a form of *implementation* of this specification, subject to its own conformance clause.

However as often the authors of a specification make the choice to focus on a subset of the possible implementations, and to worry about conformance of these only. Here, we decide that the **baseball diamond** is the implementation of interest, subject to conformance requirements.

Because there are here two variants (or subtypes) of baseball diamond, with different conformance requirements, we will have one conformance clause for each:

* One for Little league baseball diamonds
* One for College baseball diamonds

These will be our **conformance targets**.

#### **7.1.5.1** Conformance Clause 1: **Little League Baseball Diamond**

(*A Little League Baseball Diamond* ***MUST*** *conform to Sections 7.1.1, 7.1.2, 7.1.3.1 and 7.1.4*.)

A baseball diamond qualifies as a Little League Baseball Diamond if:

* Its base satisfies the requirements in 7.1.1.1 “Bases” section, and
* its home plate satisfies the requirements in 7.1.1.2 “Home Plate” section, and
* Its base path satisfies the requirements in 7.1.2 “Base path”, and
* Its base path has a length as defined in 7.1.3.1 “Sixty Foot Base Paths”, and
* It contains all the elements listed in 7.1.4 “Baseball Diamond”, and its infield area is in proportion with its base path length.

### 7.1.5.2 Conformance Clause 2: College Baseball Diamond

*(A College Baseball diamond* ***MUST*** *conform to Sections 7.1.1, 7.1.2, 7.1.3.2.and 7.1.4.)*

A baseball diamond qualifies as a College Baseball Diamond if:

* Its base satisfies the requirements in 7.1.1.1 “Bases” section, and
* its home plate satisfies the requirements in 7.1.1.2 “Home Plate” section, and
* Its base path satisfies the requirements in 7.1.2 “Base path”, and
* Its base path has a length as defined in 7.1.3.2 “Ninety Foot Base Paths”, and
* It contains all the elements listed in 7.1.4 “Baseball Diamond” and its infield area is in proportion with its base path length.

## Good Conformance Clauses from the MQTT OASIS Standard

Below is good conformance clause material from the conformance section of the MQTT standard extracted from <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.

NOTE: the authors of this guideline took the liberty to reword some of the introductory text for better alignment with the best practices advocated here (the reworded text is in italic, while the original material is regular font).

*Conformance*

*Conformance Targets*

The MQTT specification defines conformance for MQTT Client implementations and MQTT Server

Implementations (*The conformance targets: Client and Server are defined in section 1.2*).

An MQTT implementation MAY conform as both an MQTT Client and MQTT Server implementation.

*Conformance Clause 1: MQTT Server*

*A Server* conforms to this specification *as MQTT Server* only if it satisfies all the statements below:

1. The format of all Control Packets that the Server sends matches the format described in Chapter 2 and Chapter 3.
2. It follows the Topic matching rules described in Section 4.7.
3. It satisfies all of the MUST level requirements in the following chapters that are identified except for those that only apply to the Client

- Chapter 1 - Introduction

- Chapter 2 - MQTT Control Packet format

- Chapter 3 - MQTT Control Packets

- Chapter 4 - Operational behavior

- Chapter 6 - (if MQTT is transported over a WebSocket connection)

- Chapter 7 - Conformance Targets

A conformant Server MUST support the use of one or more underlying transport protocols that provide an ordered, lossless, stream of bytes from the Client to Server and Server to Client.

However conformance does not depend on it supporting any specific transport protocols. A Server MAY support any of the transport protocols listed in Section 4.2, or any other transport protocol that meets the requirements of [MQTT-7.1.11].

A conformant Server implementations MUST NOT require the use of any extensions defined outside of this specification in order to interoperate with any other conformant implementation.

*Conformance Clause 2: MQTT Client*

A *Client* conforms to this specification *as MQTT Client* only if it satisfies all the statements below:

…

Note the following:

* The conformance targets are first distinguished and characterized as *Server* and *Client* . These are defined more precisely at the beginning of the specification - prior to this section, as this allows to more clearly assign normative requirements to either.
* The clause(s) refer to precise sections or subsections, and further narrows these to mandatory requirements (MUST) as well as to only those that concern this target type (here, Server).
* Some normative content is referenced, that is defined as “rules” and expressed without normative keywords (see clause sub-item #2: “It follows the Topic matching rules described in Section 4.7”)
* Some additional requirements are added that are more appropriate here than in the normative body of the specification: they are directly concerned with the notion of conformance, as opposed to conventional normative requirements. For example, an adequate “underlying transport protocol” is not part of MQTT core requirements, but necessary to MQTT deployment in a conforming product. Also notice the statement excluding the need for “extensions”, a concern external to MQTT but of importance for interoperable products claiming conformance to MQTT.

### Good Conformance Clauses from the XML standard

The following is extracted from: <http://www.w3.org/TR/REC-xml/>

What this example illustrates:

* Some normative content, by its nature does not lend itself well to the use of normative keywords.
* How conformance clauses may reuse each other.

Consider the following normative content:

[Definition: A parsed entity contains text, a sequence of characters, which may represent markup or character data.]

[Definition: A character is an atomic unit of text as specified by ISO/IEC 10646 [ISO/IEC 10646]. Legal characters are tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646. The versions of these standards cited in A.1 Normative References were current at the time this document was prepared. New characters may be added to these standards by amendments or new editions. Consequently, XML processors must accept any character in the range specified for Char.]

[2] Char ::= [#x1-#xD7FF] | [#xE000-#xFFFD] |

[#x10000-#x10FFFF] /\* any Unicode character, excluding the surrogate

blocks, FFFE, and FFFF. \*/

[2a] RestrictedChar ::= [#x1-#x8] | [#xB-#xC] | [#xE-#x1F] |

[#x7F-#x84] | [#x86-#x9F]

[4] NameStartChar ::= ":" | [A-Z] | "\_" | [a-z] | [#xC0-#xD6]

| [#xD8-#xF6] | [#xF8-#x2FF] | [#x370-#x37D] | [#x37F-#x1FFF] |

[#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF]

| [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]

[4a] NameChar ::= NameStartChar | "-" | "." | [0-9] | #xB7 |

[#x0300-#x036F] | [#x203F-#x2040]

[5] Name ::= NameStartChar (NameChar)\*

[6] Names ::= Name (#x20 Name)\*

Note that \*none\* of these definitions use normative keywords, MUST/SHALL, etc.

The original conformance clause says:

Conforming XML processors fall into two classes: validating and non-validating. Validating and non-validating processors alike must report violations of this specification's well-formedness constraints in the content of the document entity and any other parsed entities that they read.

In an OASIS specification, this would be written as:

Conforming XML processors fall into two classes: *non-validating* and *validating*, each with its own conformance requirements. Thus, there are two conformance clauses, for Non-validating [XML] Processor and Validating [XML] Processor.

Notes:

The class of implementations concerned by Conformance is here clearly identified: XML processors. The characterization or functional definition of this type of implementation must have been made preferably prior to the conformance section. The conformance clauses only define under which conditions such implementations conform to either validating or non-validating.

The conformance clauses below do not address conformance of XML documents, which is possibly another kind of implementation – and conformance target.

Illustrative Conformance Clauses:

**Conformance Clause 1**: Non-validating Processor

In order to qualify as a Non-validating Processor, a processor MUST:

Accept characters as defined by the Char production [2].

Accept as Name Start Characters, characters as defined by the NameStartChar production [4].

Accept as Name Characters, characters as defined by the NameChar production [4a].

Accept as a single Name, the character strings defined in production [5].

Accept as Names the character strings defined in production [6].

Check the document entity, described in section 4.8, as well as the internal DTD subset for well-formedness.

Report violations, at user option, of this specification's well-formedness constraints (as defined in section 2.1) in the content of the document entity and any other parsed entities that it processes.

**Conformance Clause 2**: Validating Processor

In order to qualify as a Validating Processor, a processor MUST:

Satisfy conformance clause 1 listed above for non-validating processors (i.e. it must qualify as non-validating).

Check whether the constraints expressed in the DTD, as described in section 2.8, are fulfilled and, at user option, report violations.

Notes:

The two above clauses are related as defining levels of conformance (see later): the level “validating” is above of, and includes the level “non-validating”.

These conformance clauses are referring to normative content that do not contain normative keywords (MUST, SHALL, MAY …). This content is however precise enough (productions defined in the XML specification) to be used normatively.

## Examples of Bad Conformance Clauses

The following are examples of sentences from hypothetical conformance clauses that are deficient in some manner or contain language that is inappropriate in a conformance clause:

When processing a document some features can be ignored and not displayed.

First, the word “can” is not a recommended keyword for a Normative Statement and “can” needs to be replaced with MAY or MUST, and should be qualified. Second, it is not clear which features can be ignored; this needs to be qualified. Finally, a Conformance Target has not been defined, so it is not clear what processes a document. A better phrasing would be:

A word processor MAY ignore the following features contained within a document and SHALL choose NOT to display these features: <list of features>.

The client should conform to all the normative statements that apply to it in the specification.

This conformance clause has a target, “client”, but is not specific enough about what the client is expected to do. Also, the “should” should be changed to MUST. Then the above requirement makes a very strong assumption that it is clear enough throughout the specification which normative statement applies to “client” and which does not. Finally, the clause should not ask the client to “conform” but to satisfy or fulfill requirements. Conformance only exists in reference to the clause.

An entity that institutes a XXX process that incorporates the principles described in this document may be said to be in conformance with the findings of this TC.

First, the “entity” is too vague as conformance target - unless it is given a specific meaning by a definition in the specification body. Then the "principles" referred to should be better identified in the body of the specification and the referencing from the conformance clause back to the specification body should be more precise (see 5.2.3). They should be worded as normative requirements of the kind: "an … process MUST/SHOULD be structured as … and then the conformance clause can say something like "... is conforming if all mandatory requirements in Section 3 & 4 are satisfied".

An implementation is conformant with this specification if it satisfies all of the MUST and REQUIRED level requirements defined herein for the functions implemented. Normative text within this specification takes precedence over normative outlines, which in turn take precedence over the XML Schema and WSDL descriptions, which in turn take precedence over examples.

Unless defined more clearly in the specification body, “*implementation”* is too vague. Needs to be more specific with regards to exactly which MUST and REQUIRED statements are to be conformed with (“*defined herein*” is too vague). The precedence rules between normative contents should be specified in the body of the specification, not in a conformance clause. Typically, examples should not be normative and never referenced in a conformance clause.

The specification says: Servers SHOULD include the available bindings supported in the xyz Object. Then the conformance clause says: The xyz Object MUST provide a list of the bindings supported.

When a normative statement from the specification body is strengthened in the conformance clause - e.g. from SHOULD to MUST, or MAY to SHOULD - the conformance clause must still refer to the original statement and make explicit mention of its strengthening. Example:

The server MUST provide a list of the bindings supported in the xyz Object. Note that this is a strengthening of the statement in section 1.2.3.

Also, in case there is no other conformance clause that makes use of the original statement as is, then it is preferable that the strengthened requirement is directly stated in the specification and the conformance clause just refers to it.

## Examples of Related Conformance Clauses

The following example is taken from [ebXML Registry Services Specification v2.0](http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrs.pdf), and illustrates how to write multiple conformance clauses and relate them to each other:

5.5 Implementation Conformance

An implementation is a conforming ebXML Registry if the implementation meets the conditions in Section 5.5.1.

An implementation is a conforming ebXML Registry Client if the implementation meets the conditions in Section 5.5.2. An implementation is a conforming ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to the conditions of Section 5.5.1 and Section 5.5.2. An implementation shall be a conforming ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and Registry Client.

5.5.1 Conformance as an ebXML Registry

An implementation conforms to this specification as an ebXML Registry if it meets

the following conditions:

1. Conforms to the ebXML Registry Information Model [ebRIM].

2. Supports the syntax and semantics of the Registry Interfaces and

3. Supports the defined ebXML Registry Schema (Appendix B).

4. Optionally supports the syntax and semantics of Section 8.3, SQL

5.5.2 Conformance as an ebXML Registry Client

An implementation conforms to this specification, as an ebXML Registry

if it meets the following conditions:

1. Supports the ebXML CPA and bootstrapping process.

2. Supports the syntax and the semantics of the Registry Client Interfaces.

3. Supports the defined ebXML Error Message DTD.

4. Supports the defined ebXML Registry Schema (Appendix B).

Section 5.5.1 is a conformance clause for an ebXML Registry conformance target. Section 5.5.2, is a conformance clause for an ebXML Registry Client. Both these clauses reference normative material. The introduction paragraph Section 5.5 defines three top level conformance clauses, references the clauses containing the details (5.5.1 and 5.5.2), and defines the relationship between the clauses. In this case it uses a mix of alternative and combined styles: an implementation is either a Registry, or a Client, or a Registry and a Client.

# Appendix A: Conformance Clause Tutorial

Consider a specification entitled *MyPreferredProtocol* that defines the structure of some messages, and defines the behavior of a Sender as well as of a Receiver for these messages.

The first job of the conformance clause editor is to identify and define the conformance targets (see definition in main document).

Products implementing the *MyPreferredProtocol* specification will be message handlers. It is expected that some message handlers will act in the role of a Sender only, some will act in the role of a Receiver only, and some will act as both.

Consequently, there are at least two ways to conform to *MyPreferredProtocol* using different parts of the specification:

* As a Sender
* As a Receiver

Each one of these roles represents a way to implement (a part of) the specification, and identifies a conformance target. The conformance section will define precisely what is expected from an implementation (the conformance target) to qualify either as a *MyPreferredProtocol* Sender or as a *MyPreferredProtocol* Receiver (or as both).

So, the first thing to do is to define the conformance targets. It is not necessary for the definitions to be very detailed and precise – just sufficient to give the reader a general understanding of their function and role.

For example, in our *MyPreferredProtocol* specification, the Sender and Receiver may be introduced as follows:

This specification assumes two roles for its implementations – or two types of implementations: Sender and Receiver.

* A Sender generates messages - monitoring notices and alarms - according to its configuration. The Sender also may dynamically re-configure itself based on received specific configuration messages.
* A Receiver receives and interprets monitoring and alarm messages according to some static configuration, and generates a log for such messages.

The above conformance targets – or types of implementations – should be defined at the beginning of a specification, so that throughout the rest of the specification, normative statements can relate to the relevant implementation types by their names. For example, once Sender and Receiver have been defined as above, one can write normative statements that are specific to them, such as:.

The header of all messages sent by a Sender MUST be structured as defined in this section.

* *A Sender SHOULD NOT send more repeats of the same alarm message than what its configuration requires.*
* *A Sender MAY bundle several monitoring messages together in the same message envelope as defined below.*
* *A Receiver SHOULD accept and process bundled messages, with the same effect as if they were separate.*
* *A Receiver MUST order the received messages in a log according to their origination timestamp.*

Then, at the end of the specification (but before any Appendicies), there should be a Conformance section containing the following sorts of clauses:

Conformance Clause 1: Basic Sender

A Sender implementation is said to conform as Basic Sender if it satisfies all of the following, non exclusive of other features that do not conflict with this specification:

* It satisfies all the mandatory (MUST, REQUIRED, MUST NOT) statements applicable to Sender in Sections 3 “General Monitoring Constraints” and 4 “The Sending Function”
* It only generates messages that are structured as described in Section 2 “Basic Message Structure” and in a way that is consistent with the semantics of its current configuration, as described in Section 6 “Sender Configuration”.

Conformance Clause 2: Basic Receiver

A Receiver implementation is said to conform as Basic Receiver if it satisfies all of the following:

* It satisfies all themandatory (MUST, REQUIRED, MUST NOT) statements applicable to Receiver in Section 3 “General Monitoring Constraints” and 5 “The Receiving Function”.
* It accepts messages structured as described in Section 2 “Basic Message Structure” and interprets them as described in Section 7 “Message Semantics”.
* It accepts and processes bundled messages.

It may also present features that do not conflict with this specification.

Notes about the above clauses:

The names “Basic Sender” and “Basic Receiver*”* are names of conformance profiles that are not quite the same as the names of our targets (Sender and Receiver). This is to allow future definitions of additional conformance profiles for these targets – e.g. “Advanced Sender”. What matters is that each conformance clause is clearly associated with a conformance target.

The message bundling is an option on the Sender side that may or may not be implemented by the Basic Sender implementation – but here the Basic Receiver clause requires its conformance target to understand and process a message bundle (see (c) ). This is a requirement stronger than the bundling statement for Receivers (SHOULD) in the specification.

A product implementation may act both as Sender and Receiver, in which case it is possible – but not necessary - to define a third conformance clause for that combination –e.g. named “Basic Sender-Receiver”. Such a clause would refer to the two above clauses in order to combine their requirements. Without such a combined clause, a product may still claim conformance as both Basic Sender and Basic Receiver, meaning it satisfies both conformance clauses.

The wording of this clause does not rely on normative keywords such as RFC2119 keywords (MUST, SHOULD…). It could, however, have been written using such keywords, e.g.: “*In order to* *conform as Basic Sender, a Sender implementation MUST satisfy all of the following, non exclusive of other features that do not conflict with this specification: …”* These keywords are most useful in the body of the specification itself.

Now, our specification could have defined more than one clause for a conformance target. For example, the additional clause below can be added in case we want to distinguish two levels of capabilities for a Sender:

Conformance Clause 3: Advanced Sender

A Sender implementation is said to conform as Advanced Sender if it satisfies all of the following:

* It qualifies as a Basic Sender by satisfying the related conformance clause (Clause 1 “Basic Sender”).
* It satisfies in addition all the recommended (SHOULD, SHOULD NOT) statements applicable to Sender in Section 3 “General Monitoring Constraints” and 4 “The Sending Function”.
* When receiving a re-configuration message, the Sender must change its configuration and be effective under the new configuration within 200 milliseconds.

It may also present other features that do not conflict with this specification.

Notes about the above clause:

The Advanced Senderclause refers to the Basic Sender clause. It adds more requirements to Basic Sender. It can be seen as a higher level of conformance for a Sender.

The additional requirement (b) makes mandatory for this level of conformance, normative statements that were otherwise defined as optional (SHOULD) in the specification. Strengthening the prescription level in a conformance clause is allowed. The opposite (relaxing it e.g. transform a MUST in a SHOULD) is not allowed.

A conformance clause may be more specific about practical and operational requirements of an implementation, expressing concrete conditions that would not be appropriate as normative content in a specification narrative – here, time restrictions for an operation execution.

In this example, the implementations under consideration (conformance targets) are message processors – Senders and Receivers. However an implementation is not restricted to executable code. A specification can define conformance targets that are documents, artifacts, services or even a dynamic process, For example, we could write a conformance clause with “Message Log” as conformance target, or even just “Message”, provided our specification has normative statements about these.