



OASIS Content Assembly Mechanism Specification Version 1.1

Committee Specification Draft 02

26 February 2007

Specification URIs:

This Version:

http://docs.oasis-open.org/cam/dev/OASIS-CAM-Specifications-1_1-RC-014-030807.html
http://docs.oasis-open.org/cam/dev/OASIS-CAM-Specifications-1_1-RC-014-030807.doc
http://docs.oasis-open.org/cam/dev/OASIS-CAM-Specifications-1_1-RC-014-030807.pdf

Previous Version:

http://docs.oasis-open.org/cam/dev/OASIS-CAM-Specifications-1_1-RC-012-021507.html
http://docs.oasis-open.org/cam/dev/OASIS-CAM-Specifications-1_1-RC-012-021507.doc
http://docs.oasis-open.org/cam/dev/OASIS-CAM-Specifications-1_1-RC-012-021507.pdf

Latest Version:

http://docs.oasis-open.org/cam/update/OASIS-CAM-specification-v1_1.html
http://docs.oasis-open.org/cam/update/OASIS-CAM-specification-v1_1.doc
http://docs.oasis-open.org/cam/update/OASIS-CAM-specifications-v1_1.pdf

Latest Approved Version:

http://docs.oasis-open.org/cam/OASIS-CAM-specifications-v1_1.html
http://docs.oasis-open.org/cam/OASIS-CAM-specifications-v1_1.doc
http://docs.oasis-open.org/cam/OASIS-CAM-specifications-v1_1.pdf

Technical Committee:

OASIS Content Assembly Mechanism TC

Chair(s):

David RR Webber

Editor(s):

Martin Roberts
David RR Webber

Related work:

This specification replaces or supercedes:

- OASIS CAM v1.0 committee draft

This specification is related to:

- OASIS ebXML specifications (ISO 15000)
- OASIS web services specifications
- W3C XPath, namespaces, XSD and XML specifications

Declared XML Namespace(s):

xmlns:as=<http://docs.oasis-open.org/cam/xmlns>
asm1, asm2, asm3, default namespaces placeholders as needed

Abstract:

The Content Assembly Mechanism (CAM) provides an open XML based system for using business rules to define, validate and compose specific business documents from generalized schema elements and structures.

A CAM rule set and document assembly template defines the specific business context, content requirement, and transactional function of a document. A CAM template must be capable of consistently reproducing documents that can successfully carry out the specific transactional function that they were designed for. CAM also provides the foundation for creating industry libraries and dictionaries of schema elements and business document structures to support business process needs.

The core role of the OASIS CAM specifications is therefore to provide a generic standalone *content assembly mechanism* that extends beyond the basic structural definition features in XML and schema to provide a comprehensive system with which to define dynamic e-business interoperability.

Status:

This document was last revised or approved by the Content Assembly Mechanism TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/cam>

The CAM TC work is operating on an open license approach charter with unencumbered content, see the Technical Committee web page at <http://www.oasis-open.org/committees/cam/charter.php>

For information relating to disclosure of patents pertaining to the CAM TC work, and if any such contributing member statements exist, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/cam/ipr.php>)

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/cam>

Notices

Copyright © OASIS® 1993–2007. All Rights Reserved. OASIS trademark, IPR and other policies apply.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", "Content Assembly Mechanism (CAM)" are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

1	Introduction	6
1.1	Terminology	7
1.2	Normative References	7
1.3	Non-Normative References	7
1.4	Terms and Definitions	7
1.5	Symbols and Abbreviations	8
2	Pre-requisites	10
3	Content Assembly Mechanism Technical Specification	11
3.1	Overview	14
3.2	Header declarations	16
3.2.1	Parameters	16
3.2.2	Pseudo Variables	16
3.2.3	Properties	17
3.2.4	Imports	17
3.3	Assembly Structures	18
3.4	Business Use Context Rules	21
3.4.1	XPath syntax functions	30
3.4.2	Handling CDATA content with XPath	31
3.4.3	CAM content mask syntax	32
3.5	Predicate Format Options	41
3.6	In-line use of predicates and references	44
3.7	Advanced Features	48
3.8	Use of namespace declarations	48
3.9	Extending CAM Processors	50
3.9.1	as:Extension	50
3.9.2	Preprocessor Extensions	51
3.9.3	Postprocessor Extensions	51
3.9.4	as:include	51
3.9.5	Template Location defaulting	51
3.9.6	Selection of Assembly Structure	52
3.10	Future Feature Extensions	53
A	Addendum	55
3.11		55
3.12 A1.1	CAM schema (W3C XSD syntax)	55
3.13 A1.2	CAM Processor Notes (Non-Normative)	55
3.14 A1.3	Processing Modes and Sequencing	56
B	Addendum	57
3.15		57
3.16 B1.1	CAM extension mechanism example	57
	Appendix A. Acknowledgements	58
	Appendix B. Non-Normative Text	59
	Appendix C. Revision History	60

Figures and Tables

- Figure 1 - The implementation model for a CAM processor..... 6
- Figure 2 - Deploying CAM Technology – Context Driven Assembly 11
- Figure 3 - Deploying CAM technology – Context Driven Validation..... 12
- Figure 4 – Deploying CAM technology – Defining Content Rules and Structures..... 13
- Figure 5 - High-level parent elements of CAM (in simple XML syntax)..... 14
- Figure 6 - Structure for entire CAM syntax at a glance 15
- Figure 7 – Example of Structure and format for AssemblyStructure 18
- Figure 8 - Substitution and fixed parameters values, with a well-formed XML structure..... 19
- Figure 9 - The Assertion predicates for BusinessUseContext 21
- Figure 10 – Syntax example for BusinessUseContext..... 23
- Figure 11 - Matrix of predicates for BusinessUseContext declarations..... 25
- Figure 12 - XPath Comparator functions 30
- Figure 13 - Matrix of in-line statement commands and predicate commands 44
- Figure 14 - Use of in-line commands with a well-formed XML structure 48
- Figure 15 - An example of namespace declarations for CAM templates 49

1 Introduction

The core role of CAM remains the same - defining, composing and validating XML content. The version 1.1 of the CAM specification seeks to simplify the original work and more clearly delimit between core normative features and extended non-normative sections and items. Also V1.1 builds from lessons learned over the past two years in developing actual CAM templates. The new approach aligns closely with common industry practice in marshalling and unmarshalling XML content, the XML DOM and allows the use of common XML tools, including rule engines, alongside the CAM toolset. Consequently the CAM toolset now provides a powerful set of typical XML scripted functional components that by default are needed when exchanging XML business transactions.

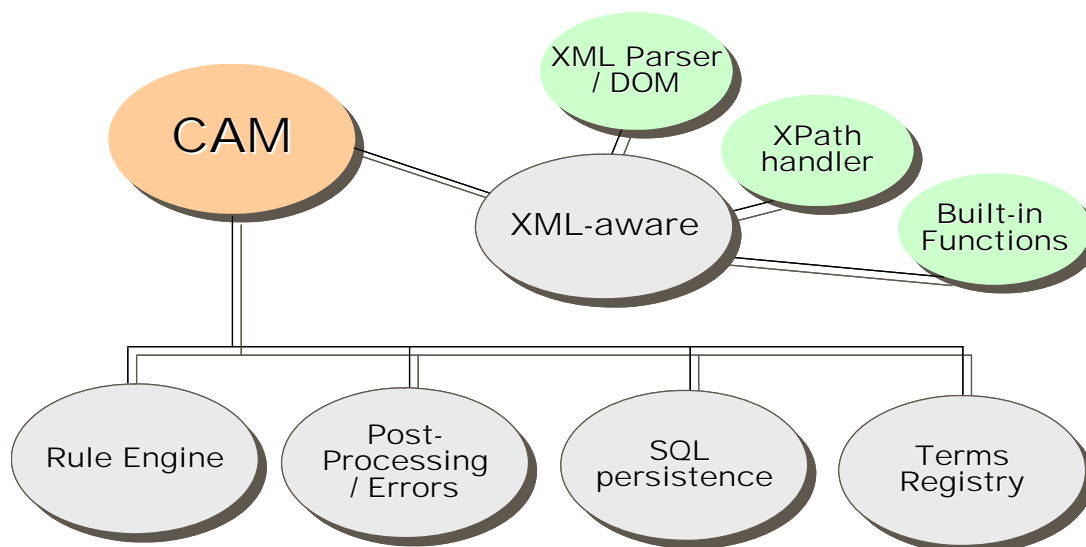
The XML scripting is designed to be obvious, human readable and declarative. This means that the task of providing rule-driven control mechanisms can become open and re-usable across an ebusiness community of practice, not just for localized internal point solutions. This is especially important in today's web service environments to support the concept of loose-coupling of service interfaces and their associated transaction interchanges. We have also taken into account the W3C and OMG work on rules.

The objective in releasing v1.1 is to provide a foundation specification that is simple, clear and easy to implement today. Whereas the new approach now allows integration with specialized tools that link into backend database systems and/or handles specialized structure formats, specialized error handling mechanisms or provide engines for complex rule based logic. In addition support for external context mechanisms are provided to align with business process needs, such as the OASIS ebBP/BPSS.

This approach is designed to separate the common sharable needs from the in-house local specializations in a coherent systematic way. This allows implementers to isolate their own point development and still align with common community practice and core business information handling structures and rules.

Future extensions to the specification may then build out and provide additional normative tools as extended areas are better formalized and common industry practice establishes itself. An example of the need to develop further normalized specification parts include registry interfacing and marshalling and unmarshalling to and from SQL content repositories. Today these are provided by specialized tools and CAM provides a formal extension mechanism and application programming interface (API) for these non-normative needs.

Figure 1 - The implementation model for a CAM processor



Referencing Figure 1 - the top-most XML-aware functions are normative components required of a CAM processor to support the core XML-scripting functionality. The lower components are optional tools supported by the pluggable interface that CAM v1.1 provides. Implementers can use local specialized tools as determined by their specific application environment. It is envisioned this implementation model can be

34 developed using a variety of modern programming languages and the pluggable interface is supported by
35 tools such as the Apache Foundation Maven technology. This flexibility allows for support of W3C Rule
36 Interchange Format (RIF) and OMG Production Rule Representation (PRR) as applicable.
37

38 1.1 Terminology

39 The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”,
40 “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in **Error!**
41 **Reference source not found.**

42 All text is normative unless otherwise labelled.

43 1.2 Normative References

- 44 - XML Path Language (XPath) specifications document, version 1.0, W3C Recommendation 16 November
45 1999, <http://www.w3.org/TR/xpath/>
46
- 47 - Extensible Markup Language (XML) specifications document, version 1.1, W3C Candidate
48 Recommendation, 15 October 2002, <http://www.w3.org/TR/xml11/>
49
- 50 - XML Schema Definitions (XSD) – **[XSD1]** XML Schema Part 1: Structures, W3C Recommendation 2 May
51 2001 <http://www.w3.org/TR/xmlschema-1/>
52 <http://www.oasis-open.org/committees/download.php/6248/xsd1.html>
53 **[XSD2]** XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001
54 <http://www.w3.org/TR/xmlschema-2/>
55 <http://www.oasis-open.org/committees/download.php/6247/xsd2.html>
- 56 - XNL: Specifications & Description Document, OASIS CIQ TC, <http://www.oasis-open.org/committees/ciq>
57
- 58 - XAL: Specifications & Description Document, OASIS CIQ TC, <http://www.oasis-open.org/committees/ciq>
59
- 60 - ISO 16642 – Representing data categories <http://www.loria.fr/projets/TMF/>
61
- 62 - CEFAC – Core components specifications - <http://webster.disa.org/cefact-groups/tmg/>
63

64 1.3 Non-Normative References

- 65
- 66 - Jaxen reference site - <http://jaxen.org/>
67
- 68 - UN – eDocs resource site - <http://www.unece.org/etradet/unedocs/>
69
- 70 - UN – Codelists reference site for eDocs - <http://www.unece.org/etradet/unedocs/codelist.htm>
71

72 1.4 Terms and Definitions

73 **Assembly model**

74 A tree-structured model that can be implemented as a document schema.

75 **Class diagram**

76 A graphical notation used by **[UML]** to describe the static structure of a system, including object
77 classes and their attributes and associations.

- 78 **Component model**
79 A representation of normalized data components describing a potential network of associations and
80 roles between object classes.
- 81 **Context**
82 The circumstance or events that form the environment within which something exists or takes place.
- 83 **Dependency diagram**
84 A refinement of a class diagram that emphasizes the dependent associations between object classes.
- 85 **Document**
86 A set of information components that are interchanged as part of a business transaction; for example,
87 in placing an order.
- 88 **Functional dependency**
89 A means of aggregating components based on whether the values of a set of properties change
90 when another set of properties changes, that is, whether the former is dependent on the latter.
- 91 **Normalization**
92 A formal technique for identifying and defining functional dependencies.
- 93 **Spreadsheet model**
94 A representation of an assembly model in tabular form.
- 95 **XSD schema**
96 An XML document definition conforming to the W3C XML Schema language [\[XSD1\]](#)[\[XSD2\]](#).

97 The terms *Core Component (CC)*, *Basic Core Component (BCC)*, *Aggregate Core Component (ACC)*,
98 *Association Core Component (ASCC)*, *Business Information Entity (BIE)*, *Basic Business Information Entity*
99 *(BBIE)*, and *Aggregate Business Information Entity (ABIE)* if used in this specification refer to the meanings
100 given in [\[CCTS\]](#).

101 The terms *Object Class*, *Property Term*, *Representation Term*, and *Qualifier* are used in this specification with
102 the meanings given in [\[ISO11179\]](#).

103 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
104 RECOMMENDED, MAY and OPTIONAL, when they appear in this document, are to be interpreted as
105 described in [\[RFC2119\]](#).

106 **1.5 Symbols and Abbreviations**

107 **ABIE**
108 Aggregate Business Information Entity

109 **ACC**
110 Aggregate Core Component

111 **ASBIE**
112 Association Business Information Entity

113 **ASCC**
114 Association Core Component

115 **ASN.1** ITU-T X.680-X.683: Abstract Syntax Notation One; ITU-T X.690-X.693: ASN.1 encoding rules
116 <http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-X.693-0207w.zip>
117 <http://www.oasis-open.org/committees/download.php/6320/X.680-X.693-0207w.zip>

118 **BBIE**
119 Basic Business Information Entity

120 **BCC**
121 Basic Core Component

122 **BIE**
123 Business Information Entity

124 **CC**
125 Core Component

126 **CCTS** UN/CEFACT ebXML Core Components Technical Specification 2.01
127 <http://www.untmg.org/downloads/General/approved/CEFACT-CCTS-Version-2pt01.zip>
128 <http://www.oasis-open.org/committees/download.php/6232/CEFACT-CCTS-Version-2pt01.zip>

129 **EAN**
130 European Article Numbering Association

131 **EDI**
132 Electronic Data Interchange

133 **ISO**
134 International Organization for Standardization

135 **ISO11179** ISO/IEC 11179-1:1999 Information technology — Specification and standardization of data
136 elements — Part 1: Framework for the specification and standardization of data elements
137 [http://www.iso.org/iso/en/ittf/PubliclyAvailableStandards/c002349_ISO_IEC_11179-1_1999\(E\).zip](http://www.iso.org/iso/en/ittf/PubliclyAvailableStandards/c002349_ISO_IEC_11179-1_1999(E).zip)
138 [http://www.oasis-open.org/committees/download.php/6233/c002349_ISO_IEC_11179-](http://www.oasis-open.org/committees/download.php/6233/c002349_ISO_IEC_11179-1_1999%28E%29.pdf)
139 [1_1999%28E%29.pdf](http://www.oasis-open.org/committees/download.php/6233/c002349_ISO_IEC_11179-1_1999%28E%29.pdf)

140 **JSDF**
141 Java Simple Date Format library

142 **NDR**
143 UBL Naming and Design Rules (see Appendix B.4)

144 **RFC2119** Key words for use in **RFCs** to Indicate Requirement Levels
145 <http://www.faqs.org/rfcs/rfc2119.html>
146 <http://www.oasis-open.org/committees/download.php/6244/rfc2119.txt.pdf>
147 S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
148 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.

149 **UML**
150 Unified Modeling Language [UML] Version 1.5 (formal/03-03-01)
151 <http://www.omg.org/docs/formal/03-03-01.pdf>
152 <http://www.oasis-open.org/committees/download.php/6240/03-03-01.zip>

153 **UN/CEFACT**
154 United Nations Centre for Trade Facilitation and Electronic Business

155 **XML**
156 Extensible Markup Language [XML] 1.0 (Second Edition), W3C Recommendation 6 October 2000
157 <http://www.w3.org/TR/2000/REC-xml-20001006>
158 <http://www.oasis-open.org/committees/download.php/6241/REC-xml-20001006.pdf>

159 **XSD**
160 W3C XML Schema Language [\[XSD1\]](#) [\[XSD2\]](#)

161

162 **2 Pre-requisites**

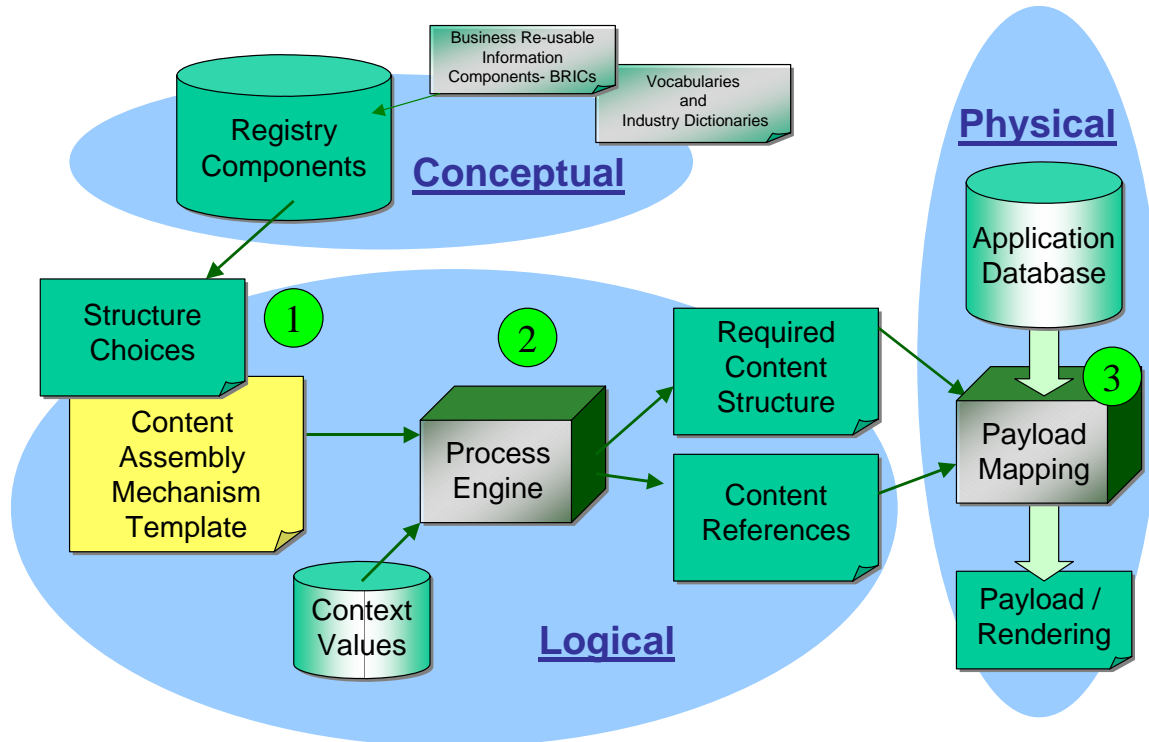
163

164 These specifications make use of W3C technologies, including the XML V1.0, XML namespaces, W3C
165 Schema V1.0 (XSD) with W3C Schema data types V1.0, and XPath 1.0 recommendations. It should be
166 noted that only a subset of the XPath technology, specifically the locator sections of the XPath specification
167 are utilized. Explicit details of XPath syntax are provided in the body of this specification. A schema definition
168 is provided for the assembly mechanism structure. Knowledge of these technologies is required to interpret
169 the XML sections of this document.

170 **3 Content Assembly Mechanism Technical Specification**

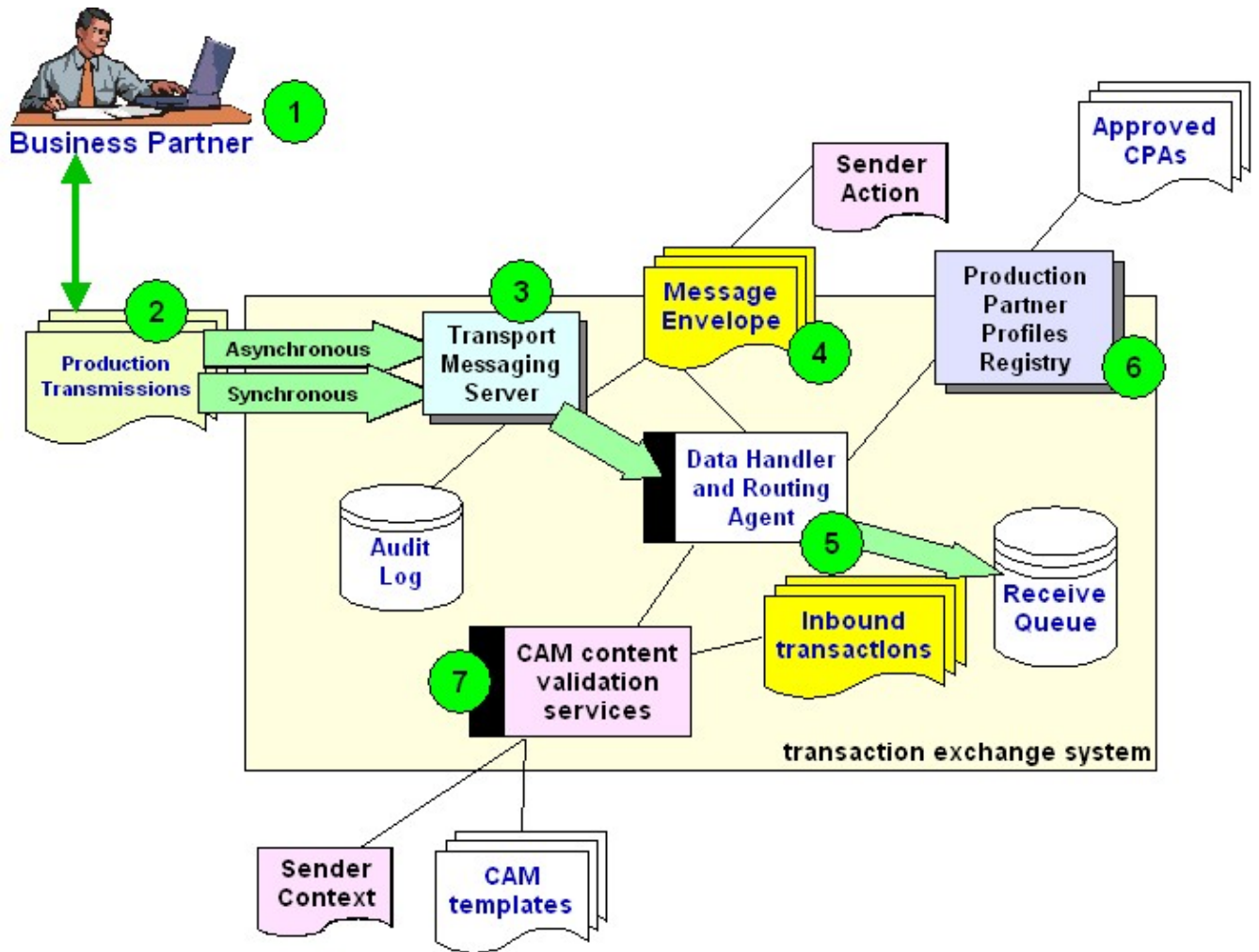
171 This section describes the implementation specifications for CAM. As noted above there are three roles to
172 CAM – defining, composing and validating content. Figure 1 shows how implementers can integrate CAM
173 technology into their existing content generation systems, while Figure 2 shows CAM in a content validation
174 role, and then Figure 3 shows defining content rules.

175 *Figure 2 - Deploying CAM Technology – Context Driven Assembly*



176
177
178 In reference to Figure 2 - Deploying CAM Technology – Context Driven Assembly, item 1 is the subject of this
179 section, describing the syntax and mechanisms. Item 2 is a process engine designed to implement the CAM
180 logic as an executable software component, and similarly item 3 is the application XML marshalling and
181 unmarshalling component that links the e-business software to the physical business application software and
182 produces the resultant transaction payload for the business process needs.
183 Input to the conceptual model section can come from UML and similar modelling tools to define the core
184 components and relevant re-usable business information components themselves, or can come from existing
185 industry domain dictionaries.
186 The specification now continues with the detailing the physical realization in XML of the CAM template
187 mechanism itself using a fully-featured eBusiness deployment environment example.
188 The Figure 2 shows how CAM can be integrated as a content validation service within a transactional
189 exchange system using partner profiles, context and actions to drive transaction validation.

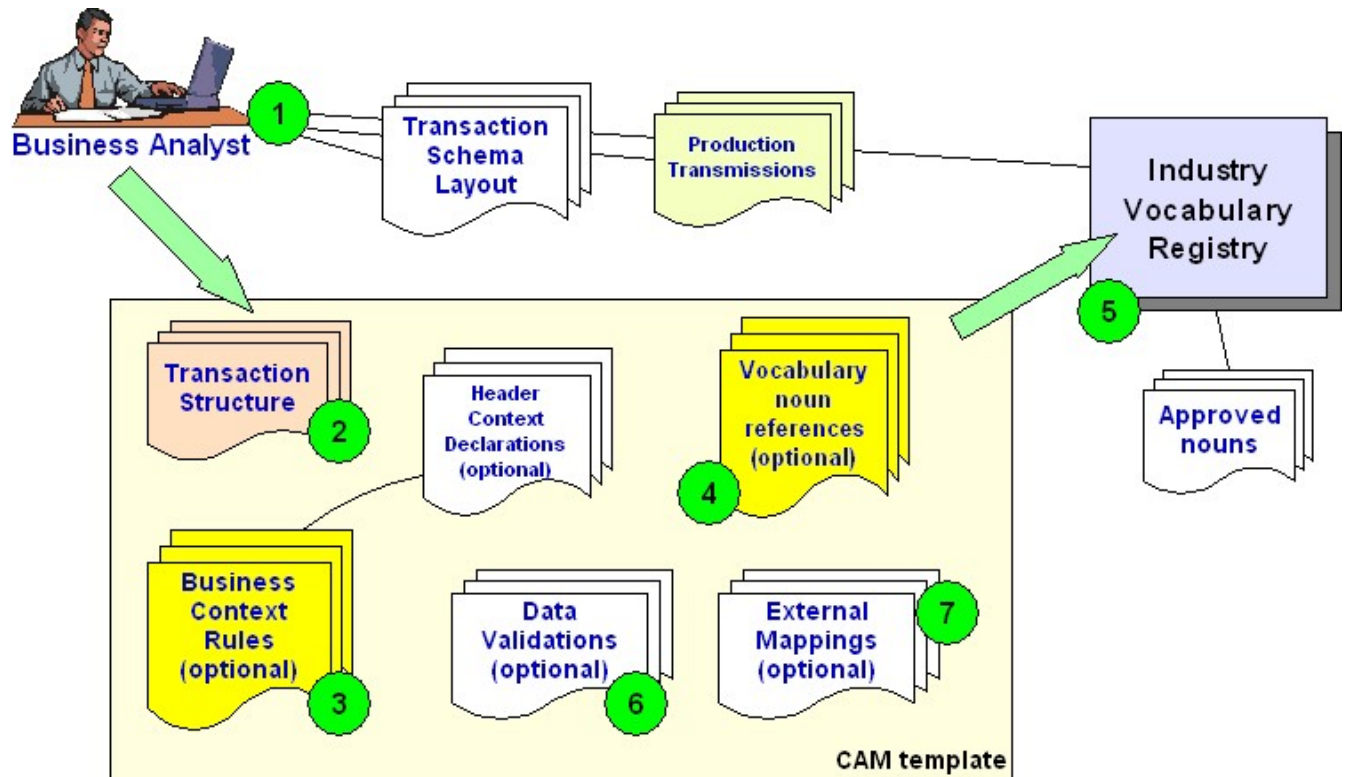
Figure 3 - Deploying CAM technology – Context Driven Validation



191
192

193 Referencing the Figure 3 - Deploying CAM technology – Context Driven Validation, the business partner (#1)
 194 sends business transactions (#2) to the partners messaging server (#3). The messaging envelope (#4)
 195 contains the sender action and the data handler (#5) checks that against the partner profiles on record in the
 196 Registry (#6). The sender action from the envelope also determines via the CPA (Collaboration Partner
 197 Agreement) the CAM template associated with that business process step. The data handler (#5) then
 198 invokes the CAM validation services (#7) and passes the references to: the inbound transaction on the
 199 receive queue, the sender context and the CAM template. The CAM validation services (#7) then verifies the
 200 content and returns either the precise error details found or a valid transaction status back to the data handler
 201 for action. Using this configuration allows CAM to act as a context driven validation service that is
 202 configurable via the partner CPA, the Sender Action from the message envelope received, and the CAM
 203 templates defined for the business process.

204 Then Figure 4 below provides a lower level of detail into the XML script mechanisms required and the
 205 business analysis steps that lead to the definition of these contents.



207
208
209

210 Referencing Figure 4 above the business analyst examines the business transaction schema layouts (#1), the
 211 sample production transmissions, and references the industry vocabulary dictionary. Using the CAM template
 212 the actual transaction structure required (#2) is defined. This may optionally contain additional context rules
 213 (#3) that direct CAM processing based on variables and values (the header section can contain global context
 214 declarations). Then noun references may also be created (#4) that cross-reference between the structure
 215 elements (#2) and the registry dictionary (#5) and the approved industry noun definitions. Optionally local
 216 application validation rules (#6) may also be added that test specific local requirements and also optional (#7)
 217 is the application mappings (such as database table columns). Used in this role the CAM template captures
 218 the information exchange details in an XML template that can then be shared and referenced between
 219 partners and agreed to as the business information requirements.

220 The tools from both Figure 3 and Figure 4 can also be deployed interactively via a web browser interface to
 221 allow partners to pre-test, and / or, self-certify prior to production message exchanges being sent. This can
 222 provide online interactive tools where sample XML transactions can be tested by upload to a CAM validation
 223 tool that applies the selected template and reports online any errors detected.

224

225

226 3.1 Overview

227 The CAM itself consists of four logical sections and the CAM template is expressed in XML syntax. This is
228 shown in figure 5 as high-level XML structure parent elements¹.

229 *Figure 5 - High-level parent elements of CAM (in simple XML syntax)*

230

```
231 <CAM CAMlevel="1" version="1.1">  
232     <Header>  
233     <AssemblyStructure/>  
234     <BusinessUseContext/>  
235     <Extension/> <!--Optional, repeatable -->  
236 </CAM>
```

237

238 The structure sections provide the core of the publically agreed interchange definition between exchange
239 partners - Assembly Structure(s), and Business Use Context Rules. Then the internal pre- or post processing
240 can be referenced as local include extensions as needed for specializations.

241 The optional extensions and includes are envisioned to support specialized non-normative handling that in the
242 prior CAM specification functionality included items such as Content References (with optional associated
243 data validation), extended Data Validations including rule agents and marshalling/unmarshalling content via
244 External Mappings. These process needs are now retained as future potential normative items that are still
245 evolving and described in a non-normative companion document to the main V1.1 specification as Appendix
246 B.

247 Figure 6 - Structure for entire CAM syntax at a glance² next shows the complete v1.1 specification hierarchy
248 for CAM at a glance.

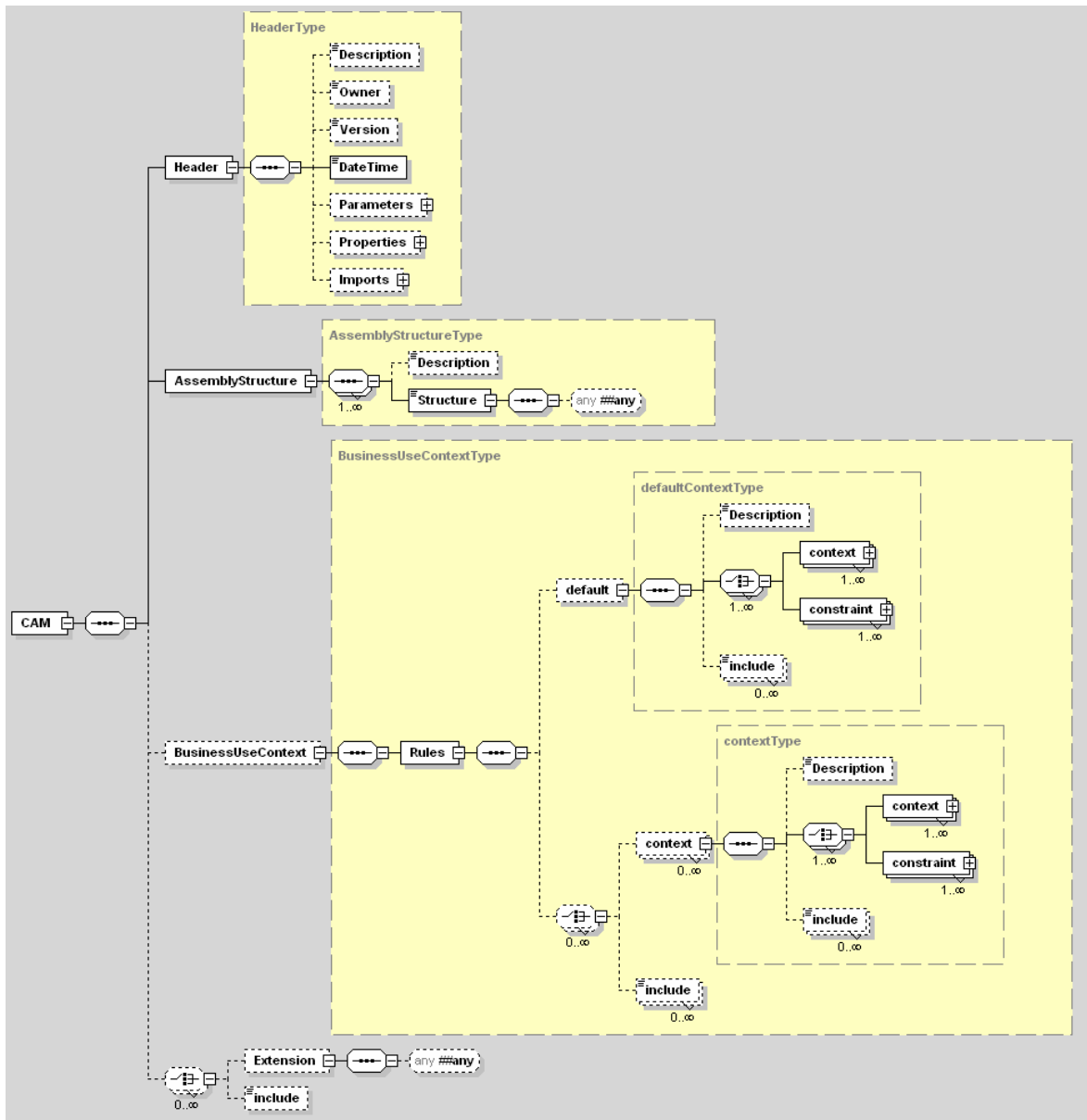
249 The CAM header it should be noted has built-in support for compatibility levels within the specification to both
250 aid in implementation of the CAM tools, and also to ensure interoperability across versions.

251 This is controlled via the CAMlevel attribute of the CAM root element. More details on the CAM
252 implementation levels and features are provided in advanced options section later.

¹ Note: elements have been labelled using UN spellings, not North American spellings

² This diagrammatic syntax uses modelling notations to show parent, repeated, choice and optional model element linkages. Elements outlined with dashed lines are optional.

Figure 6 - Structure for entire CAM syntax at a glance



254

255

256

257

258

259

260

261

262

263

264

Each of the parent items is now described in detail in the following sub-sections, while the formal schema definition for CAM is provided at the OASIS web site in machine readable Schema format XSD syntax. While the documented schema provides a useful structural overview, implementers should always check for the very latest version on-line at the docs.oasis-open.org/cam area to ensure conformance and compliance to the latest explicit programmatic details.

The next sections describe each parent element in the CAM in sequence, their role and their implementation details.

265 **3.2 Header declarations**

266 The purpose of the Header section is to declare properties and parameters for the CAM process to reference.
267 There are three sub-sections: parameters, properties and imports. Within the main header there are
268 elements that allow documenting of the template description, owner, assigning of a version number and
269 providing a date/time stamp. These are used for informational purposes only and maybe used by external
270 processes to verify and identify that a particular CAM template instance is the one required to be used.

271 **3.2.1 Parameters**

272 This section allows parameters to be declared that can then be used in context specific conditions and tests
273 within the CAM template itself. These can either be substitution values, or can be referencing external
274 parameter values that are required to be passed into this particular CAM template by an external process.
275 CAM uses the \$name syntax to denote external parameter references where required in the CAM template
276 statements. External parameters can be passed using the CAM context mechanism (see later section on
277 Advanced Features support).

278 **3.2.2 Pseudo Variables**

279 This item is non-normative, level 2.

280 When processing documents it is often expedient to have access to the system time. This would allow
281 checks against that time to be made and therefore validation to check for example that delivery dates are in
282 the future. To do this CAM defines the following pseudo variables.

- 283 • \$date – this gives today's date in the format YYYY-MM-DD
- 284 • \$time – this gives the time at the start of processing the incoming file in the format HH:MI:SS
- 285 • \$dateTime – this is a combination of the previous variables in the format YYYY-MM-DDTHH:MI:SS

286 These variables should be set by the processor at the start of processing for each incoming document.

287 In addition there is a need for date math functions to be provided to allow checks against the current time and
288 date and also between date fields. The following is considered a minimal set that may be provided.

289 These functions compare a field with the date or time of the validation:

- 290 • dateAfterNow(xpath,dateMask)
- 291 • timeAfterNow(xpath,timeMask)
- 292 • dateBeforeNow(xpath,dateMask)
- 293 • timeBeforeNow(xpath,timeMask)

294 The following functions allow either a positive or negative integer, which represents either days or hours to be
295 added to Now:

- 296 • dateAfterDays(xpath,dateMask,numofdays)
- 297 • timeAfterHours(xpath,dateMask,numofhours)
- 298 • dateBeforeDays(xpath,dateMask,numofdays)
- 299 • timeBeforeHours(xpath,dateMask,numofhours)

300

301 The following functions allow comparison between two fields:

- 302 • after(xpath,mask,xpath,mask)
- 303 • before(xpath,mask,xpath,mask)

304

305 3.2.3 Properties

306 This item is non-normative, level 2.

307 These allow creation of shorthand macros that can be referenced from anywhere in the remainder of the CAM
308 template using the `#{macroname}` reference method. This is designed to provide an easy way to maintain
309 references to external static URL values particularly. It can also be used to define shorthand for commonly
310 repeated blocks of syntax mark-up within the CAM template itself, such as a name and address layout, or a
311 particular XPath expression.

312

313 3.2.4 Imports

314 This item is non-normative, level 2.

315 The import reference allows the CAM processor to pre-load any reference links to external files containing
316 syntax to be included into the CAM template. It also allows the external path of that include file to be
317 maintained in just one place in the template; making easier maintenance if this is re-located. In addition this
318 then allows an `<include>` statement within the CAM template to reference the import declaration and select a
319 particular sub-tree of content syntax to insert at that given point (using an XPath statement to point to the
320 fragment within the overall import file). This also allows the included content to be done by using just one
321 large file, instead of multiple small files.

322 The include statements would have the format:

```
323 <as:include>${importname}/xpath</as:include>
```

324 An example with an import declared as 'common_rules' would be as follows:

```
325 <as:include>${common_rules}//as:BusinessUseContext/as:Rules/as:default</as:include>
```

326 This example will load any default rules from the 'common_rules' CAM Template into the current template.

327 The next section begins describing the main processing associated with the CAM template.

329 3.3 Assembly Structures

330 The purpose of the AssemblyStructure section is to capture the required content structure or structures that
 331 are needed for the particular business process step (i.e. one business process step may have more or more
 332 structures it may contextually need to create). This section is designed to be extremely flexible in allowing the
 333 definition of such structures. The current V1.x series of the specification uses simple well-formed XML
 334 throughout to illustrate the usage. Later releases of the CAM specification consideration will be made to allow
 335 any fixed structured markup as potentially being utilized as an assembly structure, such as DTD, Schema,
 336 EDI³, or other (typically they will be used as substitution structures for each other). It is the responsibility of
 337 the implementer to ensure that all parties to an e-business transaction interchange can process such content
 338 formats where they are applicable to them (of course such parties can simply ignore content structures that
 339 they will never be called upon to process).

340 Notice also that typically a single business process with multiple steps would be expected to have multiple
 341 CAM templates, one for each business process step. While it is also possible to provide a single CAM
 342 template with multiple structures for a business process with multiple steps, this will likely not work unless the
 343 business transaction for each step is essentially the same (since the content reference section and context
 344 rules section would have to reference potentially extremely different structures).

345 Using single CAM templates per step and transaction structure also greatly enhances re-use of CAM
 346 templates across business processes that use the same structure content, but different context.

347 The formal structure rules for AssemblyStructure are expressed by the syntax in 0 below. The Figure 7 –
 348 Example of Structure and format for AssemblyStructure here shows a simple example for an
 349 AssemblyStructure using a single structure for content.

350

351 *Figure 7 – Example of Structure and format for AssemblyStructure*

```

352 <Header>
353     <Description>Example 4.2.1 using structures</Description>
354     <Version>0.05</Version>
355 </Header>
356 <AssemblyStructure>
357     <Structure taxonomy="XML"> //XML is the only allowed value for Version 1.1
358         <!-- the physical structure of the required content goes here, and can be a schema
359             instance, or simply well-formed XML detail, see example below in Figure 8 -->
360     </Structure >
361 </AssemblyStructure>
  
```

³ EDI is used in the generic sense through out this document to refer to the family of pre-XML text markup systems, such as EDI-X12, UN/EDIFACT, HL7, FIX, SWIFT and more. See <http://www.disa.org> for more details on EDI technologies. Each flavour of EDI can be accommodated within the AssemblyStructure section of the CAM template as needed.

362

363 In the basic usage, there will be just a single structure defined in the AssemblyStructure / Structure section.
364 However, in the more advanced use, multiple substitution structures may be provided and use of include
365 directives. These can also be included from external sources, with nesting of assemblies; see the section
366 below on Advanced Features for details. Also a mechanism is provided to select a structure based on an
367 XPath reference to content within an XML instance.

368 To provide the direct means to express content values within the structure syntax the following two methods
369 apply. A variable substitution value for an element or attribute is indicated by text that must start and end
370 with a '%' sign, for example '%Description%'; or simply %% where no indicative content is preferred. Any other
371 value is assumed to be a fixed content value. Figure 8 - Substitution and fixed parameters values, with a
372 well-formed XML structure shows examples of this technique.

373 *Figure 8 - Substitution and fixed parameters values, with a well-formed XML structure*

```
374 <Header>
375     <Description>Example 4.2.2 Well-formed XML structure</Description>
376     <Version>1.0</Version>
377     <as:Parameters>
378         <as:Parameter name="DeliveryCountry"
379             values="USA|Mexico|Canada|Europe "
380             use="Global"
381             default="USA"/>
382     </as:Parameters>
383
384 </Header>
385 <AssemblyStructure>
386     <Structure taxonomy="XML" ID="SoccerGear">
387         <Items CatalogueRef="2006"> //Fixed Value
388             <SoccerGear>
389                 <Item>
390                     <RefCode>%000_00_0000%</RefCode> // Value subject to rules
391                     <Description>%any text line%</Description>
392                     <Style>WorldCupSoccer</Style>
393                     <UnitPrice>%amount%</UnitPrice>
394                 </Item>
395                 <QuantityOrdered>%integer%</QuantityOrdered>
396                 <SupplierID>%</SupplierID>
397                 <DistributorID>%</DistributorID>
398                 <OrderDelivery>Normal</OrderDelivery>
399                 <DeliveryAddress>
400                     <USA> // details of address here
401                     </USA>
402                     <Mexico> // details of address here
403                     </Mexico>
404                     <Canada> // details of address here
405                     </Canada>
```

```
406     <Europe>           // details of address here
407     </Europe>
408     </DeliveryAddress>
409     </SoccerGear>
410     </Items>
411     </Structure>
412 </AssemblyStructure>
```

413
414 Referring to Figure 8 - Substitution and fixed parameters values, with a well-formed XML structure, the
415 “2006”, “WorldCupSoccer” and “Normal” are fixed values that will always appear in the payload transaction at
416 the completion of the CAM processing of the content.

417 In addition to the XML markup, within the AssemblyStructure itself may optionally be included in-line syntax
418 statements. The CAM system provides the BusinessUseContext section primarily to input context rules (see
419 section below), however, these rules may be optionally included as in-line syntax in the AssemblyStructure.
420 However, all rules where present in the BusinessUseContext section take precedence over such in-line
421 syntax rules.

422 The next section details examples of in-line context rules.

423 3.4 Business Use Context Rules

424 Once the assembly structure(s) have been defined, then the next step is to define the context rules that apply
425 to that content. The technique used is to identify a part of the structure by pointing to it using an XPath
426 locator reference, and then also applying an assertion using one of the structure predicates provided for that
427 purpose (an optional comparison evaluation expression can also be used with the XPath locator reference
428 where applicable).

429 **Note:** By default CAM assumes that any XML structure item, element or attribute, is mandatory unless a rule
430 is added in the BusinessUseContext section or an inline rule is placed in the structure.

431 **Note:** By default CAM will not enforce order of elements within an XML structure unless a rule is added in the
432 BusinessUseContext section or an inline rule is placed in the structure (same behaviour as with XML 1.0
433 attributes ordering being undetermined). This feature makes CAM templates more flexible, particularly for
434 complex structures, and prevents erroneous error flagging.

435 There are two sections to these business context rules, default rules normally apply, and conditional rules that
436 only apply if a particular rule block evaluates to true. The business rules then take the form of structure
437 assertion predicates that define the cardinality (aka occurrence usage rules) of the structure members and
438 content definitions. Figure 9 - The Assertion predicates for BusinessUseContext shows the structure
439 assertion predicates.

440 *Figure 9 - The Assertion predicates for BusinessUseContext*

<code>excludeAttribute()</code>	<code>useAttribute()</code>
<code>excludeElement()</code>	<code>useChoice()</code>
<code>excludeTree()</code>	<code>useElement()</code>
<code>makeOptional()</code>	<code>useTree()</code>
<code>makeMandatory()</code>	<code>useAttributeByID()</code>
<code>makeRepeatable()</code>	<code>useChoiceByID()</code>
<code>setChoice()</code>	<code>useElementByID()</code>
<code>setId()</code>	<code>useTreeByID()</code>
<code>setLength()</code>	<code>startBlock()</code>
<code>setLimit()</code>	<code>endBlock()</code>
<code>setValue()</code>	<code>checkCondition()</code>
<code>setDateMask()</code>	<code>makeRecursive()</code>
<code>setStringMask()</code>	<code>setUID()</code>
<code>setNumberMask()</code>	<code>restrictValues()</code>
<code>datatype() or setDataType()</code>	<code>restrictValuesByUID()</code>
<code>setRequired()</code>	<code>orderChildren()</code>
<code>allowNulls()</code>	<code>setDefault()</code>
	<code>setNumberRange()</code>

441 Each predicate provides the ability to control the cardinality of elements⁴ within the structure, or whole pieces
442 of the structure hierarchy (children within parent).

443 An example of such context rules use is provided below, and also each predicate and its' behaviour is
444 described in the matrix in figure 4.3.3 below. Also predicates can be used in combination to provide a
445 resultant behaviour together, an example is using makeRepeatable() and makeOptional() together on a
446 structure member.

447
448 Note that the BusinessUseContext section controls use of the structure, while if it is required to enforce
449 explicit validation of content, then there is also the non-normative DataValidations section that provides the
450 means to check explicitly an element to enforce content rules as required. See below for details on this
451 section. This validation section is also further described in the advanced use section since it can contain
452 extended features.

453 Predicates that affect the definition are applied using the following precedence rules. The lower numbered
454 rules are applied first and can be overridden by the high numbered rules.

- 455 1. AssemblyStructure Inline predicates.
- 456 2. BusinessUseContext default rules and predicates.
- 457 3. BusinessUseContext conditional rules and predicates.

458
459 Referring to the structure in the example shown in Figure 8 - Substitution and fixed parameters values, with a
460 well-formed XML structure, Figure 10 – Syntax example for BusinessUseContext provides examples of
461 context based structural predicate assertions. Notice that such context rules can be default ones that apply to
462 all context uses of the structure, while other context rules can be grouped and constrained by a XPath locator
463 rule expression. There are three styles of such XPath expressions:

- 464 1. XPath expression refers to structure members directly and controls their use
- 465 2. XPath expression refers to structure member and contains condition of its value
- 466 3. XPath expression refers to a variable that has been created from the Parameter or the Properties
467 section in the Header.

468
469 Such XPath expressions will match all the structural elements that they can refer to, so if a unique element is
470 always required, implementers must ensure to provide the full XPath identity so that only a single unique
471 match occurs. An example is a reference to “//ZIPCode” which will match any occurrence, whereas
472 “/BillingAddress/ZIPCode” will only match that item.

⁴ Predicates can also be used on attributes as well as elements in the XML structure.

473 Figure 10 – Syntax example for BusinessUseContext

```
474
475 <BusinessUseContext>
476 <Rules>
477   <default>
478     <context> <!-- default structure constraints -->
479       <constraint action="makeRepeatable(//SoccerGear)" />
480       <!-- type 1 Xpath-->
481       <constraint action="makeMandatory(//SoccerGear/Items/*)" />
482       <constraint action="makeOptional(//Description)" />
483       <constraint action="makeMandatory(//Items@CatalogueRef)" />
484       <constraint action="makeOptional(//DistributorID)" />
485       <constraint action="makeOptional(//SoccerGear/DeliveryAddress)" />
486     </context>
487   </default>
488   <context condition="//SoccerGear/SupplierID = 'SuperMaxSoccer'">
489     <!-- type 2 Xpath-->
490     <constraint action="makeMandatory(//SoccerGear/DeliveryAddress)" />
491   </context>
492   <context condition="$DeliveryCountry = 'USA'">
493     <!-- type 3 Xpath using parameter DeliveryCountry-->
494     <constraint action="useTree(//SoccerGear/DeliveryAddress/USA)" />
495   </context>
496 </Rules>
497 </BusinessUseContext>
```

498
499 Referring to the XPath expressions in Figure 10 – Syntax example for BusinessUseContext, examples of all
500 three types of expression are given to show how the XPath expressions are determined and used. For
501 external control values the special leading \$ indicator followed by the variable name denotes a substitution
502 value from a context reference variable that is declared in the CAM template header.

503
504 Referring to Figure 11 - Matrix of predicates for BusinessUseContext declarations) below, the following
505 applies:

506

//elementpath	XPath expression resolving to an element(s) in the structure. This parameter is not required when predicate is used in-line, since then it is implicit.
//memberpath	XPath expression resolving to either an element(s) or an attribute(s) in the structure
//treepath	XPath expression resolving to parent element with children in the structure
//StructureID	reference to an in-line ID assignment within the structure, or ID value assigned using setID() predicate.

//elementpath@ attributename	XPath expression resolving to an attribute or attributes in the structure
//attributepath	This can be used interchangeably with //elementpath when //memberpath is an allowed parameter of a predicate. Either a single XPath expression resolving to an attribute in the structure, or a collection of XPath expressions referencing more than one attribute for the given element of the form //elementpath@[attributename1, attributename2, attributename3,...], or //elementpath@[*] to reference all attributes for that element.
IDvalue	String name used to identify structure member
UIDreference	Valid UID and optional associated registry and taxonomy that points to an entry in a Registry that provides contextual metadata content such as a [valuelist] or other information
value, valuelist, count, mask	String representing parameter. When lists are required then group with paired brackets [a, b, c, ...], and when group of groups use nested brackets [[a, b, d, f],[d, e, g, m]] Note: groups are required for collections of attributes in in-line predicate assertions.

507

Predicate	Parameter(s)	Description
<code>excludeAttribute()</code>	<code>//elementpath@attributename</code>	Conditionally exclude attribute from structure
<code>excludeElement()</code>	<code>//elementpath</code>	Conditionally exclude element from structure
<code>excludeTree()</code>	<code>treepath</code>	Conditionally exclude a whole tree from structure
<code>makeOptional()</code>	<code>//elementpath</code>	Conditionally allow part of structure to be optional
<code>makeMandatory()</code>	<code>//elementpath</code>	Conditionally make part of structure required
<code>makeRepeatable()</code>	<code>//elementpath</code>	Conditionally make part of structure occur one or more times in the content
<code>setChoice()</code>	<code>//elementpath</code>	Indicate that the first level child elements below the named elementpath are actually choices that are conditionally decided with a <code>useChoice()</code> predicate action
<code>setId()</code>	<code>//elementpath, IDvalue</code>	Associate an ID value with a part of the structure so that it can be referred to directly by ID
<code>setLength()</code>	<code>//memberpath, value</code>	Control the length of content in a structure member
<code>setLength()</code>	<code>//memberpath, [minvalue-maxvalue]</code>	Control the length of content in a structure member, allows two factors for range of lengths.
<code>setLimit()</code>	<code>//elementpath, count</code>	For members that are repeatable, set a count limit to the number of times they are repeatable
<code>setDateMask()</code> <code>setStringMask()</code> <code>setNumberMask()</code>	<code>//memberpath, [mask masklist]</code> <code>or</code> <code>//memberpath, [mask masklist]</code>	Assign a CAM picture mask to describe the content. The mask can also set explicit datatype of an item as well using the first parameter of the mask accordingly (default is string if datatype parameter omitted). Masklist allows an optional list of masks to be provided as well as one single mask.
<code>datatype()</code> <code>or</code> <code>setDatatype()</code>	<code>//memberpath, value</code>	associate datatype with item, valid datatypes are same as W3C datatypes. If a <code>setMask()</code> statement is present for the item, this statement will be ignored.

Predicate	Parameter(s)	Description
<code>setRequired()</code>	<code>//elementpath,value</code>	For members that are repeatable, set a required occurrence for the number of members that must at least be present (nnnn must be greater than 1) ⁵ .
<code>setValue()</code>	<code>//memberpath, value</code>	Place a value into the content of a structure
<code>setValue()</code>	<code>//memberpath, [valuelist]</code>	Place a set of values into the content of a structure (allows selection of multiple values of member items).
<code>as:datetime()</code> Non-Normative,level 2	<code>date-picture-mask</code> <code>date-picture-mask + P7D</code> <code>date-picture-mask - P30D</code>	Allows variables to contain computed date values for use in rule comparisons or setting event timings (value is returned from system clock of server)
<code>setUID()</code> Non-Normative,level 2	<code>//memberpath, alias, value</code>	Assign a UID value to a structure element. Alias must be declared in registry addressing section of ContentReferences).
<code>restrictValues()</code>	<code>//memberpath,</code> <code>[valuelist],[defaultValue]</code>	Provide a list of allowed values for a member item
<code>restrictValuesByUID()</code>	<code>//memberpath, UIDreference,</code> <code>[defaultValue]</code>	Provide a list of allowed values for a member item from a registry reference
<code>useAttribute()</code>	<code>//elementpath@attributename, or</code> <code>//attributepath</code>	Require use of an attribute for a structure element and exclude other attributes
<code>useChoice()</code>	<code>//elementpath</code>	Indicate child element to select from choices indicated using a <code>setChoice()</code> predicate.
<code>useElement()</code>	<code>//elementpath</code>	Where a structure definition includes choices indicate which choice to use (this function is specific to an element path, and does not require a prior <code>setChoice()</code> predicate to be specified).

⁵ Design note: `makeRepeatable()`, `makeMandatory()` is the preferred syntax over the alternate: `makeRepeatable() as:setRequired="1"`.

Predicate	Parameter(s)	Description
<code>useTree()</code>	<code>//treepath</code>	Where a structure member tree is optional indicate that it is to be used. Note: the <code>//treepath</code> points directly to the parent node of the branch and implicitly the child nodes below that, that are then selected.
<code>useAttributeByID()</code> Non-Normative	<code>StructureID</code>	As per <code>useAttribute</code> but referenced by structure ID defined by <code>SetId</code> or in-line ID assignment
<code>useChoiceByID()</code> Non-Normative	<code>StructureID</code>	As per <code>useChoice</code> but referenced by structure ID defined by <code>SetId</code> or in-line ID assignment
<code>useTreeByID()</code> Non-Normative	<code>StructureID</code>	As per <code>useTree</code> but referenced by structure ID defined by <code>SetId</code> or in-line ID assignment
<code>useElementByID()</code> Non-Normative	<code>StructureID</code>	As per <code>useElement</code> but referenced by structure ID defined by <code>SetId</code> or in-line ID assignment
<code>checkCondition()</code> Non-Normative, level 2	<code>conditionID</code>	<code>conditionID</code> is required and references the ID of the conditional block in the data validation section (defined in attribute – conditioned). The validation block will be performed at that point in the structure processing flow.
<code>makeRecursive()</code>	<code>StructureID</code>	Denote that the specified parent element can occur recursively as a child of this parent. Note that if the <code>orderChildren()</code> is set the recursive element must occur after all the other children.
<code>startBlock()</code> Non-Normative, level 2	<code>StartBlock, [StructureID]</code>	Denote the beginning of a logical block of structure content. The <code>StructureID</code> is an optional reference. This function is provided for completeness. It should not be required for XML structures, but may be required for non-XML content; basic CAM conformance at Level 1 does not require this function.

Predicate	Parameter(s)	Description
<code>endBlock()</code> Non-Normative, level 2	<code>endBlock, [StructureID]</code>	Denote the end of a logical block of structure content. The StructureID is an optional reference, but if provided must match a previous startBlock() reference. This function is provided for completeness. It should not be required for XML structures, but may be required for non-XML content; basic CAM conformance at Level 1 does not require this function.
<code>orderChildren()</code>	<code>//elementpath</code>	This means that the children must occur within the element in the order that they occur in the Structure provided. This overrides the default CAM behaviour which is to allow child elements to occur in any order.
<code>allowNulls()</code>	<code>//memberpath</code>	<p>When used for elements either the XML empty syntax <code><empty/></code> format or the <code><empty></empty></code> format would be accepted as valid mandatory content.</p> <p>For attributes they are permitted to be empty i.e. no white space or any characters between value delimiters (" or ").</p> <p><i>Note: This is to enable a similar functionality to the "nillable" function in xsd, however the user would not have to supply the XML instance <code>xsi:nil="true"</code> attribute.</i></p>
<code>setDefault()</code>	<code>//memberpath</code>	<p>Sets the default value for a node to the value given (applies to element or attribute) when the item is empty or missing (if optional).</p> <p>This will allow defaults to be applied either directly or in conjunction with the <code>restrictValues()</code> function.</p> <p><i>Note: This can also apply with the <code>lookup()</code> extension function (non-normative).</i></p>

Predicate	Parameter(s)	Description
<code>setNumberRange ()</code>	<code>//memberpath</code>	<p>For use with nodes of content type number.</p> <p>This would allow the specification of a number being between two values inclusively (e.g. 0-10 would include 0 and 10).</p> <p><i>Note: This supplements the restrictValues() function for nodes of type number.</i></p>

510

511 The predicates shown in Figure 11 - Matrix of predicates for BusinessUseContext declarations) can also be
512 used as in-line statements within an assembly structure, refer to the section on advanced usage to see
513 examples of such use.

514

515

516 3.4.1 XPath syntax functions

517 The W3C XPath specification provides for extended functions. The CAM XPath usage exploits this by
518 following the same conditional evaluations as used in the open source project for the jaxen parser (this is
519 used as the reference XPath implementation). The base XPath provides the “contains” function for
520 examining content, the jaxen functions shown in Figure 12 - XPath Comparator functions below extend this to
521 provide the complete set of familiar logical comparisons.

522 *Figure 12 - XPath Comparator functions*

523

Comparator	Syntax	Description
Equal to	<code>\$variable = 'testValue'</code>	Conditionally check for a matching value
Not equal to	<code>not(value1,'value')</code>	Conditionally check for a non-matching value
Greater than	<code>value > value</code> or <code>value &gt; value</code>	Conditionally check for a greater value
Less than	<code>value < value</code> or <code>value &lt; value</code>	Conditionally check for a lesser value
Greater than or equal	<code>value >= value</code> or <code>value &gt;= value</code>	Conditionally check for a greater than or equal to value
Less than or equal	<code>Value <=value</code> or <code>value &lt;= value</code>	Conditionally check for a lesser or equal value
begins	<code>starts-with(value,value)</code>	Conditionally check for a string matching the front part of value, equal or longer strings match.
ends	<code>ends-with(value,value)</code>	Conditionally check for a string matching the end part of value, equal or longer strings match.
String length	<code>string-length()</code>	Conditional check for the length of a string.
Count	<code>count()</code>	Conditionally check for the occurrence of an element
Contains	<code>contains (value,'value')</code>	Conditional check for an occurrence of one string within another.

<code>concat</code>	<code>concat(//elementpath, //elementpath, 'stringvalue')</code>	This operator concatenates the values from locators together as a string, or constant string values. This allows evaluations where the content source may separate related fields; e.g. Month, Day, Year.
<code>after</code>	<code>after(xpath, DateMaskPicture, \$pseudovvariable)</code>	Non-normative extra function for comparison of dates and times
<code>before</code>	<code>before(xpath, DateMaskPicture, \$pseudovvariable)</code>	Non-normative extra function for comparison of dates and times

524

525 Using these capabilities provides sufficient expressive capability to denote structural combinations for context
526 driven assembly and also for basic data validation (see following applicable sections).

527 The next section shows how to associate a reference to a dictionary of content model metadata, or to provide
528 the content model directly for members of the structure content.

529

530 **3.4.2 Handling CDATA content with XPath**

531 An XML element parent may enclose a CDATA section of embedded information. When outputting such
532 information there are two choices, the CDATA markup may be stripped off and the data processed, or the
533 CDATA section, including the markup, is passed through “as-is” into the output. The XPath expression can
534 only reference the parent element and not any markup within the CDATA itself. This specification does not
535 stipulate how to treat CDATA sections.

537 **3.4.3 CAM content mask syntax**

538 In order to provide a base-line character mask set, and also to provide a character mask set that is accessible
 539 to business technical users as well as programming staff, CAM provides a default character mask system.
 540 This mask system is based on that used by typical program generator tools available today and is designed to
 541 provide a neutral method that can be mapped to specific program language syntax as needed. The mask
 542 system syntax is provided below and usage details can be found by studying the examples provided in the
 543 example tables.

544 The ability to support alternate date mask syntax for dates, such as with the Java Simple Date and Numeric
 545 Format (JSDF / JSNF) syntax⁶ and class methods, is now also permitted and a mechanism described.

546 The JSDF / JSNF functionality is very similar to the original CAM mask system but provides some extra
 547 capabilities and formats.

548 (Note: this technique can allow use of alternate mask systems syntaxes such as SQL, Perl, and so on as may be required
 549 for specific industry / partner use).

550 **Description**

551 Picture masks are categorized by the basic data-typing element that they can be used in combination with.
 552 CAM processors must check the content of the element or attribute against the masks and report any errors.

553 Note for items of arbitrary length and no mask – use the datatype() function instead of mask functions.

554 **String Pictures**

555 The positional directives and mask characters for string pictures are as follows:

556 X - any character mandatory

557 Aa - A for alphanumeric mandatory and a for alphanumeric optional may include spaces

558 ? – any character optional, * - more than one character, arbitrary occurrence of – (equivalent to CDATA).

559 U - a character to be converted to upper case

560 ^ - uppercase optional

561 L - a character to be converted to lower case

562 _ - Lowercase optional

563 0 - a digit (0-9 only)

564 # - a digit (0-9 only), trailing and leading zeros shown as absent

565 ‘ ‘ – single quotes, escape character block to denote actual mandatory character

566 Examples of string pictures are shown in the following table:

⁶ See details of SDF at - <http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html>

567

String value	Picture mask (shorthand)	Full expanded mask	Validation match
portability	X6	XXXXXX	portab
portability	UX3	UXXX	Port
portability	XXXXing	XXXXing	porting
realtime	XXXX-XXXX	XXXX-XXXX	real-time
BOLD!	L5	LLLLL	bold!
asX	XX'X'	XX'X'	Matches asX but not asd

568

569 **Numeric Pictures**

570 The positional directives and mask characters for numeric pictures are as follows:

571 0 - a digit (0-9 only)

572 # - a digit (0-9 only), trailing and leading zeros shown as absent

573 . - indicates the location of the decimal point. For example, '0000.000' defines a numeric variable of four whole digits and three decimal digits

575 J - Uppercase, first character of - invoke alternate optional Java character format library methods to handle mask processing - character J is ignored in actual mask (see alternate masks item below)

577 Examples of numeric pictures are shown in the following table (the ^ symbol represents one space character):

Numeric value	Picture
-1234.56	#####.##
-1234.56	000000.##
-1234.56	-#####.##
0	-#####.##Z* where Z indicates zero suppress - e.g. 000000.01 becomes 0.01

578 **Basic Date Pictures**

579 The typical date formats are DD/MM/YYYY (European), MM/DD/YYYY (American), or YYYY/MM/DD
 580 (Scandinavian). When you define the attribute Date for a variable, you must also select the format for the date
 581 item (see below). You can change this default picture and place in it any positional directives and mask
 582 characters you need.

583 DD—A place holder for the number of the day in a month

584 DDD—The number of the day in a year

585 DDDD—The relative day number in a month

586 MM—A place holder for the number of the month in a year

587 MMM...—Month displayed in full name form (up to 10 'M's in a sequence). e.g. January, February. If the
 588 month name is shorter than the number 'M's in the string, the rest of the 'M' positions are filled with blanks.

589 YY—A place holder of the number of the year

590 YYYY—A place holder for the number of the year, represented in full format (e.g. 1993)

591 W—Day number in a week

592 WWW...—Name of day in a week. The string can be from 3 to 10 'W's. If the name of the day is shorter than
 593 the number of 'W's in the string, the rest is filled with blanks.

594 /—Date separator position.

595 —Date separator position (alternate).

596 J - Uppercase, first character of – invoke alternate optional Java character format library methods to handle
 597 mask processing – character J is ignored in actual mask (see alternate masks item below)

598 Examples of date pictures are shown in the following table, using the date of 21 March 1992 (the ^ symbol
 599 represents one space character – used to show spaces for this document only):

600

Picture	Validation Matches
MM/DD/YYYY	03/21/1992
MMMMMMMMMM^DDDD, ^YYYY	March^21st,^1992
MMMMMMMMMM^DDDD, ^YYYYT	March^21st,^1992 with trimming directive (see below)
WWWWWWWWWW^~W	Saturday^~^7
WWWWWWWWWW^~WT	Saturday^~^7 with trimming directive (see below)

601 "Trimming directive" is invoked by adding the directive T to the variable picture. This directive instructs XML
 602 parser to remove any blanks created by the positional directives 'WWW...' (weekday name), 'MMM...' (month
 603 name), or 'DDDD' (ordinal day, e.g. 4th, 23rd). Since these positional directives must be specified in the
 604 picture string using the maximum length possible, unwanted blanks may be inadvertently created for names
 605 shorter than the specified length. The Trim Text directive will remove all such blanks. If a space is required
 606 nevertheless, it must be explicitly inserted in the picture string as a mask character, (the ^ symbol is used to
 607 indicate a blank character), e.g., 'TWWWWWWWWWW^DDDD MMMMMMMMMM,^YYYY'

608 "Zero fill" is invoked by adding the functional directive Z to the variable picture. This directive instructs XML
 609 parser to fill the entire displayed variable, if its value is zero, with the "Character" value. If you don't specify a
 610 Character the variable is filled with blanks.

611 **Time Pictures**

612 The XML parser defines the default picture mask HH/MM/SS for an element of datatype Time. Examples of
 613 time pictures are shown in the following table:

614

Picture	Result	Comments
HH:MM:SS	08:20:00	Time displayed on 24-hour clock.
HH:MM:SS	16:40:00	Time displayed on 24-hour clock.
HH:MM PM	8:20 am	Time displayed on 12-hour clock.
HH:MM PM	4:40 pm	Time displayed on 12-hour clock.
HH-MM-SS	16-40-00	Using Time Separator of '-'

615

616 3.4.3.1 Alternate Simple Date Format - Date and Time Patterns

617 The simple date and time formats are specified by *date and time pattern* strings⁷. Within date and time pattern
618 strings, unquoted letters from 'A' to 'Z' and from 'a' to 'z' are interpreted as pattern letters representing
619 the components of a date or time string. Text can be quoted using single quotes (') to avoid interpretation,
620 where "' ' " represents a single quote. All other characters are not interpreted; they're simply copied into the
621 output string during formatting or matched against the input string during parsing.

622 The following tables provide details of the patterns and their usage.

623 A compliant implementation should first check the initial character of the picture mask. If it is uppercase J
624 character – then the mask is assumed to be of Java simple format. Then the processor should pass the mask
625 to the equivalent alternate mask processor – such as the Java Simple Date Format method - for either date or
626 time handling, and if that then fails – then an error should be returned.

⁷ Source: Sun Java documentation - <http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html>

627

628 The following pattern letters are defined (all other characters from 'A' to 'Z' and from 'a' to 'z' are reserved):
629

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
Y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

630 Pattern letters are usually repeated, as their number determines the exact presentation:

- 631 • **Text:** For formatting, if the number of pattern letters is 4 or more, the full form is used; otherwise a
632 short or abbreviated form is used if available. For parsing, both forms are accepted, independent of
633 the number of pattern letters.
- 634 • **Number:** For formatting, the number of pattern letters is the minimum number of digits, and shorter
635 numbers are zero-padded to this amount. For parsing, the number of pattern letters is ignored unless
636 it's needed to separate two adjacent fields.
- 637 • **Year:** For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it
638 is interpreted as a [number](#).

639 For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless
640 of the number of digits. So using the pattern "MM/dd/yyyy", "01/11/12" parses to Jan 11, 12 A.D.

641 For parsing with the abbreviated year pattern ("y" or "yy"), `SimpleDateFormat` must interpret the
642 abbreviated year relative to some century. It does this by adjusting dates to be within 80 years before
643 and 20 years after the time the `SimpleDateFormat` instance is created. For example, using a
644 pattern of "MM/dd/yy" and a `SimpleDateFormat` instance created on Jan 1, 1997, the string
645 "01/11/12" would be interpreted as Jan 11, 2012 while the string "05/04/64" would be interpreted as
646 May 4, 1964. During parsing, only strings consisting of exactly two digits, as defined by
647 [Character.isDigit\(char\)](#), will be parsed into the default century. Any other numeric string, such
648 as a one digit string, a three or more digit string, or a two digit string that isn't all digits (for example, "-
649 1"), is interpreted literally. So "01/02/3" or "01/02/003" are parsed, using the same pattern, as Jan 2, 3
650 AD. Likewise, "01/02/-3" is parsed as Jan 2, 4 BC.

- 651 • **Month:** If the number of pattern letters is 3 or more, the month is interpreted as [text](#); otherwise, it is
652 interpreted as a [number](#).
- 653 • **General time zone:** Time zones are interpreted as [text](#) if they have names. For time zones
654 representing a GMT offset value, the following syntax is used:
 - 655 • *GMTOffsetTimeZone:*
 - 656 • *GMT Sign Hours : Minutes*
 - 657 • *Sign:* one of
 - 658 • *+ -*
 - 659 • *Hours:*
 - 660 • *Digit*
 - 661 • *Digit Digit*
 - 662 • *Minutes:*
 - 663 • *Digit Digit*
 - 664 • *Digit:* one of
 - 665 • *0 1 2 3 4 5 6 7 8 9*

666 *Hours* must be between 0 and 23, and *Minutes* must be between 00 and 59. The format is locale
667 independent and digits must be taken from the Basic Latin block of the Unicode standard.

668 For parsing, [RFC 822 time zones](#) are also accepted.

- 669 • **RFC 822 time zone:** For formatting, the RFC 822 4-digit time zone format is used:
 - 670 • *RFC822TimeZone:*
 - 671 • *Sign TwoDigitHours Minutes*
 - 672 • *TwoDigitHours:*
 - 673 • *Digit Digit*

674 *TwoDigitHours* must be between 00 and 23. Other definitions are as for [general time zones](#).

675 For parsing, [general time zones](#) are also accepted.

676 `SimpleDateFormat` also supports *localized date and time pattern* strings. In these strings, the pattern
677 letters described above may be replaced with other, locale dependent, pattern letters. `SimpleDateFormat`
678 does not deal with the localization of text other than the pattern letters; that's up to the client of the class.

679 3.4.3.2 Examples

680 The following examples show how date and time patterns are interpreted in the U.S. locale. The given date
681 and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

Date and Time Pattern	Examples
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, ''yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700

682

683

684 3.4.3.3 Alternate Simple Decimal Format - Number Patterns

685 The Java simple decimal formats are specified by patterns that represent the number formatting required⁸.
686 These patterns are selected using an uppercase J character to indicate the pattern syntax.

687 3.4.3.4 Patterns

688 DecimalFormat patterns have the following syntax:

689 *Pattern:*

690 *PositivePattern*

691 *PositivePattern ; NegativePattern*

692 *PositivePattern:*

693 *Prefix_{opt} Number Suffix_{opt}*

694 *NegativePattern:*

695 *Prefix_{opt} Number Suffix_{opt}*

696 *Prefix:*

697 any Unicode characters except \uFFFE, \uFFFF, and special characters

698 *Suffix:*

699 any Unicode characters except \uFFFE, \uFFFF, and special characters

700 *Number:*

701 *Integer Exponent_{opt}*

702 *Integer . Fraction Exponent_{opt}*

703 *Integer:*

704 *MinimumInteger*

⁸ Java simple decimal format - <http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html>

705 #
 706 # *Integer*
 707 # , *Integer*
 708 *MinimumInteger*:
 709 0
 710 0 *MinimumInteger*
 711 0 , *MinimumInteger*
 712 *Fraction*:
 713 *MinimumFraction*_{opt} *OptionalFraction*_{opt}
 714 *MinimumFraction*:
 715 0 *MinimumFraction*_{opt}
 716 *OptionalFraction*:
 717 # *OptionalFraction*_{opt}
 718 *Exponent*:
 719 E *MinimumExponent*
 720 *MinimumExponent*:
 721 0 *MinimumExponent*_{opt}
 722

723 A `DecimalFormat` pattern contains a positive and negative subpattern, for example,
 724 "`#,##0.00;(#,##0.00)`". Each subpattern has a prefix, numeric part, and suffix. The negative subpattern
 725 is optional; if absent, then the positive subpattern prefixed with the localized minus sign (code>'-' in most
 726 locales) is used as the negative subpattern. That is, "`0.00`" alone is equivalent to "`0.00;-0.00`". If there is
 727 an explicit negative subpattern, it serves only to specify the negative prefix and suffix; the number of digits,
 728 minimal digits, and other characteristics are all the same as the positive pattern. That means that
 729 "`#,##0.0#;(#)`" produces precisely the same behavior as "`#,##0.0#;(#,##0.0#)`".

730 The prefixes, suffixes, and various symbols used for infinity, digits, thousands separators, decimal separators,
 731 etc. may be set to arbitrary values, and they will appear properly during formatting. However, care must be
 732 taken that the symbols and strings do not conflict, or parsing will be unreliable. For example, either the
 733 positive and negative prefixes or the suffixes must be distinct for `DecimalFormat.parse()` to be able to
 734 distinguish positive from negative values. (If they are identical, then `DecimalFormat` will behave as if no
 735 negative subpattern was specified.) Another example is that the decimal separator and thousands separator
 736 should be distinct characters, or parsing will be impossible.

737 The grouping separator is commonly used for thousands, but in some countries it separates ten-thousands.
 738 The grouping size is a constant number of digits between the grouping characters, such as 3 for 100,000,000
 739 or 4 for 1,0000,0000. If you supply a pattern with multiple grouping characters, the interval between the last
 740 one and the end of the integer is the one that is used. So "`#,##,###,####`" == "`#####,####`" ==
 741 "`##,####,####`".

742

743 3.4.3.5 Special Pattern Characters

744 Many characters in a pattern are taken literally; they are matched during parsing and output unchanged
745 during formatting. Special characters, on the other hand, stand for other characters, strings, or classes of
746 characters. They must be quoted, unless noted otherwise, if they are to appear in the prefix or suffix as
747 literals.

748 The characters listed here are used in non-localized patterns. Localized patterns use the corresponding
749 characters taken from this formatter's `DecimalFormatSymbols` object instead, and these characters lose
750 their special status. Two exceptions are the currency sign and quote, which are not localized.

Symbol	Location	Localized?	Meaning
0	Number	Yes	Digit
#	Number	Yes	Digit, zero shows as absent
.	Number	Yes	Decimal separator or monetary decimal separator
-	Number	Yes	Minus sign
,	Number	Yes	Grouping separator
E	Number	Yes	Separates mantissa and exponent in scientific notation. <i>Need not be quoted in prefix or suffix.</i>
;	Subpattern boundary	Yes	Separates positive and negative subpatterns
%	Prefix or suffix	Yes	Multiply by 100 and show as percentage
\u2030	Prefix or suffix	Yes	Multiply by 1000 and show as per mille
¤ (\u00A4)	Prefix or suffix	No	Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator.
'	Prefix or suffix	No	Used to quote special characters in a prefix or suffix, for example, "'#'#" formats 123 to "#123". To create a single quote itself, use two in a row: "# o' 'clock".

751

752 For more information, examples and pattern manipulation see the documentation for the Java `DecimalFormat`
753 method and links to examples there. The library also supports use of scientific notation numbers.

754
755
756
757
758
759
760
761
762
763

3.5 Predicate Format Options

There are several ways in which predicates can be referenced with a CAM template. The tables below show the different forms to be used and when. The first table shows the BusinessUseContext Rules format when a constraint is applying one and only one action to an element or attribute. The second table is for when a constraint is applying several actions to one item (specified by a path). The third table shows the inline functions when applied to elements. The fourth shows a proposed extension for the inline definitions to be used with attributes.

TABLE 1: Functions used for constraint action attribute: <as:constraint action="functiondefn" />
<code>excludeAttribute(xpath)</code>
<code>excludeElement(xpath)</code>
<code>excludetree(xpath)</code>
<code>makeMandatory(xpath)</code>
<code>makeOptional(xpath)</code>
<code>makeRepeatable(xpath)</code>
<code>restrictValues(xpath, valuesList)</code>
<code>setChoice(xpath)</code>
<code>setDateMask(xpath, dateMask)</code>
<code>setID(xpath, idValue)</code>
<code>setLength(xpath, lengthDescription)</code>
<code>setLimit(xpath, limitValue)</code>
<code>setMask(xpath, datatype, Mask)</code>
<code>setValue(xpath, value)</code>
<code>useAttribute(xpath)</code>
<code>useChoice(xpath)</code>
<code>useElement(xpath)</code>
<code>useTree(xpath)</code>
<code>orderChildren(xpath)</code>

764
765

TABLE 2: Function used for constraint action element:

```
<as:constraint item="xpath">  
  <as:action>functiondefn</as:action>  
</as:constraint>
```

`excludeAttribute()`

`excludeElement()`

`excludetree()`

`makeMandatory()`

`makeOptional()`

`makeRepeatable()`

`restrictValues(valuesList)`

`setChoice()`

`setDateMask(dateMask)`

`setID(idValue)`

`setLength(lengthDescription)`

`setLimit(limitValue)`

`setMask(datatype,Mask)`

`setValue(value)`

`useAttribute()`

`useChoice()`

`useElement()`

`useTree()`

`orderChildren()`

766

767

TABLE 3: Inline Element functions – used alongside structure example - all are attributes
<code>as:makeMandatory="true"</code>
<code>as:makeOptional="true"</code>
<code>as:makeRepeatable="true"</code>
<code>as:restrictValues="valuesList"</code> <code>valuesList ::= value value ... value ::= string with or without single quotes</code>
<code>as:setChoice="idValue"</code> all elements in choice have same idValue
<code>as:setDateMask="dateMask"</code>
<code>as:setID="idValue"</code>
<code>as:setLength="lengthDescription" : lengthDescription = min-max or max</code>
<code>as:setLimit="limitValue"</code>
<code>as:setMask="Mask" - must be used with a as:datatype attribute for non string masks</code>
<code>as:setValue="value"</code>
<code>as:orderChildren="true"</code>

TABLE 4: Inline attribute functions – used alongside structure example all are attributes. Assumed to be for an attribute called 'example' - <code><element example="value"/></code>
<code>as:makeMandatory-example="true"</code>
<code>as:makeOptional-example="true"</code>
<code>as:restrictValues-example="valuesList"</code> <code>valuesList ::= value value ... value ::= string with or without quotes</code>
<code>as:setMask-example="Mask" - must be used with a as:datatype attribute for non string masks</code>
<code>as:setID-example="idValue"</code>
<code>as:setLength-example="lengthDescription" : lengthDescription = min-max or max</code>
<code>as:setNumberMask-example="numberMask"</code>
<code>as:setValue-example="value"</code>

771 **3.6 In-line use of predicates and references**

772 Figure 8 in Section 3.3 above shows an example for an AssemblyStructure with different structure
 773 components for address (e.g. US, Europe, Canada). Using different structures for content can be controlled
 774 with in-line statements indicating by context those optional and required content selections. The in-line
 775 commands are inserted using the "as:" namespace prefix, to allow insertion of the command statements
 776 wherever they are required. These in-line commands compliment the predicates used within the
 777 <BusinessUseContext> section of the assembly for setChoice() and useChoice(). The table in Figure 13
 778 below gives the list of these in-line statements and the equivalent predicate form where applicable.

779 In-line command entries marked as "not applicable" can only be used within the <BusinessUseContext>
 780 section. Also where there is both a predicate statement and an in-line command, then the predicate
 781 statement overrides and takes precedent. For attributes inline functions can be included by using the format
 782 `'as:attributename-functionname="value"'. .`

783 The in-line statements available are detailed in the table shown in Figure 13. In-line command entries marked
 784 as "not applicable" can only be used within the <BusinessUseContext> section. Also where there is both a
 785 predicate statement and an in-line command, then the predicate statement overrides and takes precedent.
 786 See Figure 14 below for examples of using in-line predicates.

787 *Figure 13 - Matrix of in-line statement commands and predicate commands*

Predicate	In-line Command	Notes
<code>excludeAttribute()</code>	Not applicable	
<code>excludeElement()</code>	Not applicable	
<code>excludeTree()</code>	Not applicable	
<code>makeOptional()</code>	<code>as:makeOptional="true"</code>	Make part of structure optional, or make a repeatable part of the structure optional (e.g. occurs=zero)
<code>makeMandatory()</code>	<code>as:makeMandatory="true"</code>	Make part of the structure required; leaf element may not be nillable
<code>allowNull()</code>	<code>as:allowNull="true"</code>	Allow null content model for leaf element

Predicate	In-line Command	Notes
<code>makeRepeatable()</code>	<pre>as:makeRepeatable="true" as:setLimit="5n" as:setRequired="3n"</pre>	Make part of the structure occur one or more times in the content; the optional <code>as:setLimit="nnnn"</code> statement controls the maximum number of times that the repeat can occur ⁹ . The optional <code>as:setRequired="nnnn"</code> statement controls the required occurrences that must at least be present.
<code>setChoice()</code>	Not applicable	
<code>setId()</code>	<code>as:choiceID="label"</code>	Associate an ID value with a part of the structure so that it can be referred to directly by ID
<code>setLength()</code>	<code>as:setLength="nnnn-mmmmm"</code>	Control the length of content in a structure member
<code>setLimit()</code>	<code>as:setLimit="nnnn"</code>	For members that are repeatable, set a count limit to the number of times they are repeatable
<code>setRequired()</code>	<code>as:setRequired="nnnn"</code>	For members that are repeatable, set a required occurrence for the number of members that must at least be present (nnnn must be greater than 1) ¹⁰ .
<pre>setDateMask() setNumberMask() setStringMask()</pre>	<pre>as:setDateMask="DD-MM-YY" as:setNumberMask="####.##" as:setStringMask="U8" "x'Mask' "</pre>	Assign a regular expression or picture mask to describe the content. First character of the mask indicates the type of mask.

⁹ Design note: the `setLimit` / `setRequired` are deliberately optional. It is intended they only be used sparingly, when exceptional constraints are really needed. In W3C schema max/min are used as required factors. This impairs the ability to know when an exceptional constraint is present and therefore is an inhibitor to engineering robust interoperable systems.

¹⁰ Design note: `makeRepeatable()`, `makeMandatory()` is the preferred syntax over the alternate: `makeRepeatable() as:setRequired="1"`.

Predicate	In-line Command	Notes
<code>setValue()</code>	<code>as:setValue="string"</code>	Place a value into the content of a structure
<code>restrictValues()</code>	<code>as:restrictValues="'value' 'value'"</code> <code>"[valuelist]"</code>	Provide a list of allowed values for a member item
<code>restrictValuesByUID()</code>	<code>as:restrictValuesByUID="</code> <code>UID"</code>	Provide a list of allowed values for a member item from an registry reference
<code>useAttribute()</code>	Not applicable	
<code>useChoice()</code>	Not applicable	
<code>useElement()</code>	<code>as:useElement="true"</code>	Where a structure definition includes choices indicate which choice to use.
<code>useTree()</code>	<code>as:useTree="true"</code>	Where a structure member tree is optional indicate that it is to be used.
<code>useAttributeByID()</code>	Not applicable	
<code>useChoiceByID()</code>	Not applicable	
<code>useTreeByID()</code>	Not applicable	
<code>useElementByID()</code>	Not applicable	
Not applicable	<code><as:include>URL</code> <code></as:include></code> <code><as:include ignoreRoot="yes"></code>	Allows inclusion of an external source of assembly instructions or structure. The URL is any single valid W3C defined URL expression that resolves to physical content that can be retrieved. Note: can only be used in the <Structure> section of assembly. The optional ignoreRoot attribute permits inclusion of fragments of XML that are not well-formed by ignoring the root element from the XML source content.
<code>checkCondition()</code>	<code>as:checkCondition="</code> <code>conditionID"</code>	Points to the condition to be tested in the data validation section.
<code>makeRecursive()</code>	<code>as:makeRecursive="true"</code>	Denotes element as a recursive structure member, so can appear as child of this parent.

Predicate	In-line Command	Notes
<code>orderChildren()</code>	<code>as:orderChildren="true"</code>	Denotes that the children of the element must occur in the order they occur in the reference structure template.

788

789 The next Figure 14 shows some examples of using these in-line commands within a structure.

790 *Figure 14 - Use of in-line commands with a well-formed XML structure*

```
791
792 <AssemblyStructure xmlns:as="http://www.oasis-open.org/committees/cam">
793   <Structure taxonomy='XML'>
794     <Items CatalogueRef="2002">
795       <SoccerGear>
796         <Item as:makeRepeatable="true">
797           <RefCode as:makeMandatory="true" as:setLength="10">%%</RefCode>
798           <Description>%%</Description>
799           <Style>WorldCupSoccer</Style>
800           <UnitPrice as:setNumberMask="q999.9###.##">%%</UnitPrice>
801         </Item>
802         <QuantityOrdered as:setNumberMask="q999####">%%</QuantityOrdered>
803         <SupplierID as:makeMandatory="true">%%</SupplierID>
804         <DistributorID>%%</DistributorID>
805         <OrderDelivery>Normal</OrderDelivery>
806         <DeliveryAddress/>
807       </SoccerGear>
808     </Items>
809   </Structure>
810 </AssemblyStructure>
```

811 It should be noted that in-line commands cannot be used with non-XML structures; all such structures require
812 the use of predicates within the <BusinessUseContext> section of the assembly instead.

813

814 **3.7 Advanced Features**

815 The following sections contain advanced feature options and use details.

816 **3.8 Use of namespace declarations**

817 The default CAM template assumes that all that is required is one namespace declaration for use with in-line
818 CAM predicates within a template (e.g. <myTagName as:setValue="xxx">).

819 However many business vocabularies have adopted wholesale use of namespace prefixes for the elements
820 and attributes in their schemas regardless of whether this is necessary or not. While this is not an issue for
821 the design of CAM it is an issue for several of the XML parser implementations and the way they have been
822 coded, including their DOM representations. Essentially when multi-namespace declarations exist in an XML
823 instance they can no longer support the default namespace having no prefix.

824 Unfortunately this is a common behaviour that has been widely copied due to sharing of the underlying Java
825 libraries involved. Another issue is the placing of namespace declarations. Again the XML specifications
826 permit these to occur anywhere in the XML instance. However the Java library implementation will often fail if
827 all namespace declarations are not placed at the top of the XML instance.

828 To resolve this CAM templates permit the use of a global namespace at the root CAM template level and
829 placing all namespace declarations in the root element declaration. You should only need to resort to this
830 when handling structures that involve multiple inline namespace declarations within the XMI content.

831 Processors can provide a function to extract namespace definitions from an XML example and correctly
832 define a CAM template skeleton with namespaces moved to the root node and any anonymous namespaces

833 provided with a prefix (the jCAM editor implementation provides an example of this, along with the
834 autogenerate template feature in jCAM itself). The figure 15 here illustrates an example.

835

836 *Figure 15 - An example of namespace declarations for CAM templates*

```
837 <?xml version="1.0" encoding="utf-8"?>
838 <!-- Sample CAM Template showing use of namespaces extensions -->
839 <as:CAM CAMlevel="1" version="0.13"
840     xmlns:as="http://www.oasis-open.org/committees/cam"
841     xmlns:tic="http://era.nih.gov/Projectmgmt/SBIR/CGAP/ticket.namespace"
842     xmlns:cb="http://era.nih.gov/Messaging/SBIR/CGAP/ticket.namespace"
843     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
844     xsi:schemaLocation="http://www.oasis-open.org/committees/cam
845     file:///D:/eclipse/workspace/camprocessor/schema/CAMv0151.xsd">
846     <!-- note: namespace declarations should all be here, not in body of CAM template -->
847     <as:Header>
848     <as:Description>Validates an Incoming transaction</as:Description>
849     <as:Owner>CAM smaple templates</as:Owner>
850     <as:Version>0.1</as:Version>
851     <as:DateTime>2004-09-09T17:00:00</as:DateTime>
852     <as:Parameters>
853     <!-- example parameter declaration -->
854     <as:Parameter name="applicationType" values="competing_continuation|80|70" use="global"
855     default="competing_continuation"/>
856     </as:Parameters>
857     </as:Header>
858     <as:AssemblyStructure>
859     <as:Structure ID="default" taxonomy="XML">
860     <cb:MessageType>
861     <tic:ticket>
862     <tic:institutionID>%%</tic:institutionID>
863     <tic:correctionID>%%</tic:correctionID>
864     <tic:timestamp>%%</tic:timestamp>
865     <tic:application>
866     <tic:projectTitle>%text%</tic:projectTitle>
867     <tic:applicationType>%%</tic:applicationType>
868     <tic:revisionNumber>%%</tic:revisionNumber>
869     </tic:application>
870     </tic:ticket>
871     </cb:MessageType>
872     </as:Structure>
873     </as:AssemblyStructure>
874     <as:BusinessUseContext>
```

```

875 <as:Rules>
876 <as:default>
877 <as:context>
878 <as:constraint action="setNumberMask(//tic:institutionID,#9)"/>
879 <as:constraint action="restrictValues(//tic:correctionID,'N'|'Y')"/>
880 <as:constraint action="setDateMask(//tic:timestamp,YYYY-MM-DDTHH:MI:SS)"/>
881 <as:constraint
882 action="restrictValues(//tic:applicationType,'competing_continuation'|'other')"/>
883 <as:constraint action="setNumberMask(//tic:revisionNumber,##)"/>
884 </as:context>
885 </as:default>
886 <!-- example additional rules -->
887 <as:context>
888 </as:context>
889 </as:Rules>
890 </as:BusinessUseContext>
891 </as:CAM>
892

```

893 **3.9 Extending CAM Processors**

894 Originally CAM v1.0 was designed to have 5 distinct areas within the template. These were to cover off
895 expected forms of content handling and advanced functionality. In the 1.1 specification these have been
896 replaced in favour of a more extensible framework. This framework is based on the idea of a CAM processor
897 being able to provide a core set of XML handling functions, while allowing extensions via the optional include
898 or ANY functionality.

899 An extension entry is designed to allow CAM processors to invoke functionality that is too specialized to allow
900 strict normative definition by the CAM specification and implementation by the CAM processor developers
901 (such as for local integration specialization needs, error handling and reporting, XML marshalling or un-
902 marshalling, or mutually agreed to vertical industry extensions).

903 There are two types of extension allowed, `preprocessor` and `postprocessor`. If more than one included
904 extension is defined of a given type they will be handled in the order that the extensions appear within the
905 CAM template.

906 Further ideas for implementing extensions and example syntax are provided in the Addendum B of this
907 document.

908 **3.9.1 as:Extension**

909 This is a hook to enable any extension to be included in the CAM Template. It may contain any valid XML
910 from any defined source. Any number of extensions may be defined. Any process dependencies must be
911 defined by the CAM processor supporting the Extension.

912 For Java implementations of CAM the Apache Maven linkage approach provides a default configuration
913 method for associating external process handlers with the default CAM processor.

914 An example of an extension is provision of a lookup() function. This can be tailored to suit the particular
915 domain and/or local application needs.

916

917 **3.9.2 Preprocessor Extensions**

918 Preprocessor extensions are run **after** the CAM template has been read in to the processor and after any
919 pseudo-variables have been defined for the run. Any includes of any type are also completed before the
920 extensions run. They are run **before** any BusinessUseContext rules are applied to the Structure in question.

921 In order to run the processor must supply an API to allow the preprocessor extension access to the complete
922 CAM template and also to any input file that has been supplied to the processor. The preprocessor may then
923 update either of these items before completion. A method to pass back any errors to the processor for
924 onward communication must be provided.

925 **3.9.3 Postprocessor Extensions**

926 Postprocessor extensions are to be run after all the BusinessRulesContext rules have been completed.
927 Processors are at liberty to provide an option as to whether extensions are run in the case of errors occurring
928 during the core processing.

929 As with the preprocessors there are requirements to be able to access both the CAM template after any
930 processing and the input file that has been processed. Each extension may change these and return them
931 via the API for either the processor to complete work or to pass onto further extensions. A method to pass
932 back any errors to the processor for onward communication must be provided.

933 **3.9.4 as:include**

934 The include provided outside the AssemblyStructure and BusinessUseContext elements is purely to allow
935 as:Extension elements to be included.

936 In addition note that the as:include may optionally specify the ignoreRoot="yes" attribute. This permits
937 inclusion of XML fragments that are not well-formed, by allowing a dummy root element to be used to ensure
938 the fragment is well-formed – but then the dummy root element is ignored.

939 e.g. :
940 `<tempRoot>`
941 `<not_well_formed_by_itself/>`
942 `<tag1_include/>`
943 `<tag2_include/>`
944 `<well_formed>`
945 `<tag3_include/>`
946 `<well_formed>`
947 `</tempRoot>`

948
949 So tempRoot will be ignored

950 **3.9.5 Template Location defaulting**

951 This provides the ability to associate from an XML instance to the CAM template that validates it. A URL is
952 provided for the CAM template location. A CAM processor therefore can locate and validate XML directly.

953 The syntax for this is:

954 `asi:templateLocation="[URL]">`

955 and the namespace declaration is:

956 `xmlns:asi="http://www.oasis-open.org/committees/cam/instance"`

957 and these should occur on the root element of the XML instance.

958

959 3.9.6 Selection of Assembly Structure

960 When a template contains more than one structure instance (such as different versions of the same structure)
961 it is necessary to provide the ability to dynamically select which structure to apply to an XML instance for
962 validation. One option is to pass in a CAM parameter. However this advanced feature permits the use of an
963 xpath attribute onto the Structure element that then uniquely identifies the ID value of the relevant structure
964 that should be used to validate the message (this can optionally be overridden by the structure ID name being
965 passed in from outside the template). This first matching XPath expression that returns true is then selected
966 for use.

967
968 The XML below provides an example. The xpath expression effectively equates to true if the XML instance
969 contains the matching relevant structure item, and / or associated value.

```
970  
971 <as:AssemblyStructure>  
972 <as:Structure ID="ex_1" taxonomy="XML"  
973 xpath="/ex:example"> <!-- Xpath check here -->  
974 <ex:example>  
975 <ex:test name="Fred">  
976 </ex:example>  
977 </as:Structure>  
978  
979 <as:Structure ID="new_1" taxonomy="XML"  
980 xpath="/new:example"> <!-- Xpath check here -->  
981 <new:example>  
982 <new:test name="%Fred%">  
983 <new:inside>%value%</new:inside>  
984 </new:example>  
985 </as:Structure>  
986 </as:AssemblyStructure>  
987
```

988

989 **3.10 Future Feature Extensions**

990

991 This section is provided as a holding area for potential extensions to the base CAM specifications.

992

993 **W3C RIF and OMG PRR Rule Support**

994 The ability to add extensions to the base CAM templates means that common rule syntax approaches can be
995 exploited easily to augment the base XML content validations that CAM provides. W3C Rule Interchange
996 Format (RIF) and OMG Production Rule Representation (PRR) are both examples of such extended rules
997 syntax that can be used to augment the basic built-in XPath support and CAM functions to add more complex
998 logic handling. Examples of these techniques will be developed for future use.

999

1000 **RDF / OWL support**

1001 The ability to use RDF / OWL syntax to provide metadata and semantics in the ContentReference section for
1002 elements.

1003

1004 **Registry based noun semantics**

1005 This is currently under development with the Registry SCM group and will be referenced here when complete.

1006

1007 **WSDL support for CAM processor**

1008 A draft WSDL interface has been posted to the OASIS CAM TC site for discussion and is available.
1009 Implementers may use this as a basis for deploying a CAM processor as a web service.

1010

1011 **Accessing content in ebXML Registry**

1012 The ebXML Registry Services Specification (RSS) describes this capability.
1013 Typical functions include the QueryManager's getRegistryObject, and getRepositoryItem operations. Also
1014 there is the HTTP interface and also the SQL or Filter query interface as described by AdhocQueryRequest.

1015 This also includes the possibility of running external library functions offered by a registry.

1016

1017 The registry specifications may be found at:

1018

1019 [3] ebXML Registry specifications

1020 <http://www.oasis-open.org/committees/regrep/documents/2.5/specs/>

1021 **Import Feature**

1022 Some basic IMPORT functionality is available in this V1.0 of CAM, however this is not intended to be
1023 comprehensive or complete. Subsequent versions of CAM will enhance the basic functions available in V1.0
1024 and allow more sophisticated sub-assembly techniques.

1025

1026 **XACML support**

1027 In many ways the CAM context mechanisms mirror the ability to include or exclude content as a filtering style
1028 operation between the input and output. An extension to support XACML (eXtensible Access Control Markup
1029 Language) syntax is there a natural addition to CAM processing. CAM functions can simplify the creation and
1030 coding of XACML while being able to call an XACML extension.

1031

1032

1033

1034

1035

1036

1037 **A Addendum**

1038 **3.11**

1039 **3.12 A1.1 CAM schema (W3C XSD syntax)**

1040 This item is provided as a reference to the formal specification of the XML structure definition for CAM itself.
1041 However specific implementation details not captured by the XSD syntax should be referenced by studying
1042 the specification details provided in this document and clarification of particular items can be obtained by
1043 participating in the appropriate on-line e-business developer community discussion areas and from further
1044 technical bulletins supplementing the base specifications. For specific details of the latest XSD
1045 documentation please reference the OASIS CAM TC documents area where the latest approved XSD version
1046 is available. This is also mirrored to the open source jCAM site as well (<http://www.jcam.org.uk>). See
1047 document download area from OASIS website: <http://www.oasis-open.org/committees/cam>

1048 In addition OASIS may provide a static location to the reference CAM XSD schema under [http://docs.oasis-](http://docs.oasis-open.org/cam)
1049 [open.org/cam](http://docs.oasis-open.org/cam) once an approved specification is available.

1050

1051 **3.13 A1.2 CAM Processor Notes (Non-Normative)**

1052 CAM processor notes assist implementers developing assembly software, these are non-normative. Within
1053 an assembly implementation the processor examines the assembly document, interprets the instructions, and
1054 provides the completed content structure details given a particular set of business context parameters as
1055 input. This content structure could be stored as an XML DOM structure for XML based content, or can be
1056 stored in some other in-memory structure format for non-XML content. Additionally the memory structure
1057 could be temporarily stored and then passed to a business application step for final processing of the
1058 business content within the transaction.

1059 Since typical development environments already contain linkage between the XML parser, the DOM, an
1060 XPath processor, a scripting language such as JavaScript, the data binding toolset such as XSLT or a
1061 comparable mapping tool. The assembly approach based on an XML script fits naturally into this
1062 environment.

1063 Some suggested uses and behaviours for CAM processors include:

1064

1065 · Design time gathering of document parts to build a context sensitive assembly service that can be
1066 called via an API or webservice interface.

1067

1068 · Design time generation of validation scripts and schemas for the run time environment that is not
1069 CAM savvy or that does not provide any context flexibility. Think of this as a CAM compiler. This
1070 would mean that context parameters would be passed in as input to this.

1071

1072 · Runtime validation engine based on context parameters and controlled via a Business Process
1073 engine with BPM script definitions running within the gateways of trading partners.

1074 **3.14 A1.3 Processing Modes and Sequencing**

1075 **Non-normative**

1076 Context elements can have conditions. These conditions can either be evaluated against variables
1077 (parameters) or XPath statements. These conditions can be evaluated in two modes:

1078

- 1079 1) A standalone CAM template - i.e. on the basis of external parameters values passed to the CAM
1080 processor to evaluate the conditionals.
- 1081 2) CAM template and XML instance - check the XML instance to evaluate the condition and then
1082 proceed (this is the normal mode for a CAM processor).

1083

1084 The first mode is typically used when you are trying to produce documentation about what is allowed for a
1085 transaction and it is useful to pre-process (precompile) the structure rules without the existence of an XML
1086 instance file. This means that any condition that falls into the second category can not be evaluated (these
1087 conditions then behave equivalent of having Schematron asserts, and are documented but not actioned).

1088

1089 B Addendum

1090 3.15

1091 3.16 B1.1 CAM extension mechanism example

1092 This item illustrates the approach using Apache Maven linker technology to implement the component and
1093 Extension mechanism in CAM as implemented in the jCAM open source tool. It also shows how alternative
1094 strict and lax XML conformance can be optionally configured via this mechanisms.

1095

1096 Figure B1.1.1

1097 <container>

1098 <component-implementation class="uk.org.jcam.processor.dataObjects.Template" />

1099 <component-implementation class="uk.org.jcam.processor.dataObjects.DataFile" />

1100 <!-- <component-implementation
1101 class='uk.org.jcam.processor.validator.DefaultValidator' />

1102 -->

1103 <!-- <component-implementation
1104 class='uk.org.jcam.processor.validator.UnOrderedValidatorLax' />

1105 -->

1106 <component-implementation
1107 class="uk.org.jcam.processor.validator.UnOrderedValidatorStrict" />

1108 <component-implementation class="uk.org.jcam.processor.trimmer.DefaultTrimmer" />

1109 <component-implementation class="uk.org.jcam.processor.adorner.DefaultAdorner" />

1110 <component-implementation class="uk.org.jcam.processor.transformer.XSLTransformer" />

1111 <component-implementation class="uk.org.jcam.drools.DroolsDataValidator" />

1112 <component-implementation class="uk.org.jcam.groovy.GroovyDataValidator" />

1113 <component-implementation class="uk.org.jcam.beanshell.BeanShellDataValidator" />

1114 </container>

1115 **Appendix A. Acknowledgements**

1116 The views and specification expressed in this document are those of the authors and are not necessarily
1117 those of their employers. The authors and their employers specifically disclaim responsibility for any
1118 problems arising from correct or incorrect implementation or use of this design.

1119 The following individuals have participated in the creation of this specification and are gratefully
1120 acknowledged.

1121 **Participants:**

1122

1123

Fred Carter	AmberPoint	CAM TC member
Chris Hipson	BTplc	CAM TC member
Martin Roberts	BTplc	CAM TC member
Hans Aanesen	Individual	CAM TC member
Ram Kumar	Individual	CAM TC member
Joe Lubenow	Individual	CAM TC member
Colin Wallis	New Zealand Government	Member, CIQ TC
David Webber	Individual	Chair CAM TC
Tom Rhodes	NIST	CAM TC member
Bernd Eckenfels	Seeburger, AG	CAM TC member
Paul Boos	US Dept of the Navy	CAM TC member

1124

Appendix B. Non-Normative Text

1125 Non-normative items are noted as such in the body of the specification as applicable. Possible Future
1126 Extensions are noted in that section above. Also a separate document is maintained by the CAM TC of
1127 experimental and extension items that are under consideration for inclusion in future versions of the
1128 specification. The latest public version of that draft non-normative items document is available from the
1129 committee area web site.

1130

Appendix C. Revision History

1131 [optional; should not be included in OASIS Standards]

1132

1133

1134

Change History:

Status	Version	Revision	Date	Editor	Summary of Changes
Draft	1.0	0.10	30 December, 2002	DRRW	Rough Draft
		0.11	12 th February, 2003	DRRW	Initial Draft
		0.12	23 rd February, 2003	DRRW	Revision for comments to 28/02/2003
		0.13	17 th May, 2003	DRRW	Revision for comments to 08/05/2003
		0.14	13 th August, 2003	DRRW	Revision for comments to 15/08/2003
		0.15	3 rd February, 2004	DRRW	Final edits prior to first public release
		0.16	15 th February, 2004	DRRW	Release Candidate for Committee Draft CAM
		0.17	19 th February 2004	MMER	Edited detailed comments into draft.
Committee Draft		0.17C	12 th March 2004	DRRW	Cosmetic changes to look of document to match new OASIS template and notices statement.
Revised Committee Draft		0.18	10 th December 2004	DRRW	Revisions from comment period, corrections, and bug fixes to examples. Added Table of Figures index.
		0.19	4 th January, 2005	DRRW	Layout changes to align with new OASIS document template formatting and logo. Update figure 4.1.2 to reflect latest schema, and also 4.5 for noun content referencing. Add addendum glossary of terms and abbreviations.
Revised Committee Draft	1.1	0.01	25 th May, 2006	DRRW	New revised specification to reflect extensible model and architecture.
	1.1	0.02	27 th June 2006	MR	Explicit corrections to line up with implementable features and also explicit definition of normative and non-normative sections.
	1.1	0.03	28 th June 2006	MR	Included section on extensions (plug-ins).
	1.1	0.04	4 th July 4, 2006	DRRW	Refined text, general edits.
	1.1	0.05	27 th July, 2006	DRRW	Revise examples + figure captioning
	1.1	0.06	5 th Sept 2006	MR	Issues around Order tackled

Status	Version	Revision	Date	Editor	Summary of Changes
	1.1	0.06	12 th September 2006	DRRW	Changes consolidation and clean-up edits
	1.1	0.07	12 th Sept 2006	MR	Schema Diagram Updated, Appendix re-factored, extensions approach re-worked
	1.1	0.08	15 th Sept 2006	DRRW / MR	Edits and changes for accuracy. Import Function refined, Date comparison functions amended. W3C RIF and OMG PRR notes
	1.1	0.09	21 st October 2006	MR / DRRW	Very Simple Extensions added. Align date masks with Java SDF. Correct examples XML
	1.1	0.10	24 th October 2006	DRRW	Refine masks mechanism details, including both date and numeric masks.
	1.1	0.11	2 nd November 2006	DRRW/ MR	Minor editing corrections and fixes to mask details, handling of quote characters and non-normative clarification of psuedo-variables.
	1.1	0.12	15 th February 2007	DRRW	Revised to include comments from OASIS member 60 day review period (changes noted in comment review log document).
	1.1	0.13	5 th March 2007	DRRW	Revised to use new OASIS document template and include TC member comments prior to formal Committee Specification ballot (changes noted in comment review log document).
	1.1	0.14	8 th March 2007	DRRW	Cosmetic edits/fixes to URLs and layout to meet OASIS document specification, template and site requirements.

1135

1136