Organization for the Advancement of Structured Information Standards

# CONTENT ASSEMBLY MECHANISM (CAM) SPECIFICATIONS AND DESCRIPTION DOCUMENT

VERSION 1.0

## CHANGE HISTORY

| Status | Version | Revision | Date | Editor | Summary of Changes |
|---|---|---|---|---|---|
| Draft | 1.0 | 0.10 | 30 December, 2002 | DRRW | Rough Draft |
| | | 0.11 | 12th February, 2003 | DRRW | Initial Draft |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

**Deleted:** 12/02/03

**Deleted:** 12/02/03

# TABLE OF CONTENTS

**Deleted:** 68

**Deleted:** 12/02/03

## 1.0 Acknowledgements

OASIS wishes to acknowledge the contributions of the members of the CAM Technical Committee to this standards work.

**Deleted:** 12/02/03

## 2.0 Introduction

Content assembly has been solved in a variety of ways in the past. Particularly the traditional electronic data interchange (EDI) approach is to rigorously restrict content variance so as to avoid the need for dynamic definitions in software. This proved to be both the strength and weakness of EDI, and therefore for specific business scenarios EDI itself resorted to the use of written implementation guidelines to formalize the interchange details.

With the advent of XML based transaction content implementers have learned that while constructing schema structure definitions provides a higher degree of flexibility for business scenarios than EDI nevertheless the same limitations on interoperability recur while there is no robust means to specify business scenario details for actual schema use.

OASIS itself has found that its technical teams developing industry vocabularies cannot fully derive the needed depth of detail on the use of such vocabularies while making use of schema alone. Further more the notion of producing business re-usable information components (*BRICs*[1]) both within and across OASIS industry vocabularies has been problematic in XML (especially without robust inclusion and versioning mechanisms).

Clearly the urgent business need is to move beyond this and provide a machine-readable format in XML that can then allow business application software to automatically configure the interchanges according to the business rules.

Additionally to facilitate the broad collaborative adoption of BRICs technology particularly requires a formal way of storing and retrieving vocabulary entries within the Registry technology that OASIS is developing. Again while early work has been attempted in this area using schemas alone, this has not proved suitable for the three key structural needs of atomic element definitions, lists of code values (codelists) and assembly components. Added to this is the need to provide content validation rules and a mechanism to support business context variables. Here again OASIS teams are developing specifications that require these mechanisms to coordinate across the deployment information architecture.

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.

In addition the CAM specifications are providing support and collaboration tools to existing OASIS technical work by linking together key components of the overall e-business systems architecture.

[1] Typical *BRICs* include such concepts as ‘Billing Address’, ‘Person Name’, ‘Shipping Requirements’, and so forth, where the BRICs provide pre-built collections of atomic XML elements that can be simply stacked together into a business transaction. UN/CEFACT specifications refer to these capabilities as a “BIE” – a Business Information Entity, however BIE is a logical concept, whereas BRIC is its physical manifestation in XML syntax.

**Deleted:** 12/02/03

In the context of e-business collaboration the problem fundamentally stems from the need for each partner to be able to both quickly adopt and start using standard industry building blocks and interchanges, while at the same time being able to overlay onto this their own local business context and special needs, (such as product specific information or country or locale specific information).

We now look at how this overall vision translates into specific goals, approach and functional requirements.

### 2.1.1 Goals

- Promote development of interoperable e-business systems and best practice usage for maintenance and ease of adoption of vocabularies.
- Provide the coupling between the conceptual layer and the physical production systems within the overall architecture stack. Particularly that business process definition technology can use CAM to construct transaction content for discreet business steps and provide associated use and context driven mappings.
- Provide a simple migration path for legacy business-to-business (B2B) and EDI systems to adopt XML driven mechanisms.
- Provide a coupling content include mechanism that supports possible object-oriented design methods as part of the include attributes.
- Enable Registry systems to implement library dictionaries of pre-built assembly components (BRICs) and publish these for discovery and re-use.
- Enable development of both simple public domain components and also sophisticated vendor products, and encourage early development by design simplicity.
- Provide ability to develop conformance suites by use of level mechanisms.

### 2.2.1 Approach

- Open mechanism for content assembly using simple XML scripting.
- The use of three levels within the specification to separate out the functionality by complexity. This ensures ease of implementation, future extensibility and conformance.
- Minimalist approach to use of external specifications, therefore the foundation of CAM is XML 1.0 syntax and the XPath specification, augmented with as limited set of functions and extensions as is possible.
- Extensible design coupled with a simple but powerful base foundation. The initial scope will defer complex and extended capabilities to Level 3 components and also later versions beyond the initial V1.0 release.
- Avoid reliance on complex markup devices such as namespaces, XLink and so on.
- Provide a simple coupling content include mechanism that supports possible object-oriented design methods as part of the include attributes.
- Ensure that CAM scripts can be hand-edited and are visually simple to read.

### 2.3.1 Audience

- CAM is intended for use by technical business analysts with IT experience and by implementation programmers constructing e-business systems and particularly business process definitions.

**Deleted:** 12/02/03

- It is also intended to allow solution vendors to integrate CAM functionality underneath their products to provide extended functionality by analysing and purposing CAM content and rules for designer and production components.
- End users should be able to interact with CAM driven components to adapt e-business technology to their discrete business needs.

### 2.4.1 Boundaries

- Re-use XML mechanisms as much as possible, therefore minimizing the need to re-invent software technology.
- Ensure that CAM implementations can use standard libraries, such as XML parsers and web browser environments as much as possible.
- CAM is not intended to be a mapping solution, but instead should support and enable vendors own mapping tools.
- CAM is not intended as a replacement for general-purpose schema systems, instead CAM can be used to dynamically emit schema structures, and particularly for defining e-business transaction structures.
- Collaborate with OASIS Registry and Business Process TC's around use models, functional requirements and context mechanisms.
- Define APIs as needed to exchange semantic content with external systems, such as Registries.
- Provide a neutral structural mechanism that is not specific to any markup technology but instead can handle a range of such definitions.
- Use XML syntax that can be hand edited without need for complex syntax mechanisms and tools.

### 2.5.1 Use Models

- Design process – ability to assemble business transaction models from pre-built components; re-use and discovery are strong needs
- Transaction mapping – physical layer integration between business information transactions, industry standard dictionaries, and backend application systems
- Post-design / pre-production – documentation and verification of business model and rules through generation of test materials, validation scripts and plain text documentation artefacts
- Production – context driven assembly of business transactions and e-forms and their associated validation artefacts such as schema and declarative software scripts
- Standards bodies wanting to document assembly information for industry vocabularies

The work on providing an open means of expressing transaction payloads for eBusiness is in response to a long running and recurring need when building large networks of collaborative partners electronically.

Some real world examples of such uses include the following:

- A large government department looking to simplify and manage over two thousand interfaces between inter-departmental systems including legacy COBOL formats, ERP

**Deleted:** 12/02/03

system formats, EDI transactions, audit reporting, online transactions and new XML documents.

- A government implementing an e-Gov initiative to bring electronic access to government for its citizens via electronic forms integration in two major and several minor alternate languages.
- The automotive industry looking to improve information flow to 20,000 dealerships from anyone of the 15 car manufacturers in North America.
- A supermarket chain looking to provide cheap accounting exchanges with its 2,000 small suppliers in local marketplaces for an array of different products.
- A major PC hardware manufacturer looking to send catalogue information to 27,000 websites worldwide for its reseller network.
- A telecommunications company supporting hundreds of complex technical service engineering requests using variants of EDI and XML based messages in a rapidly changing industry were new categories of products are created every month.

The current work on content assembly is therefore focused on providing a solution that can meet the business transactional needs.

### 2.6.1 Problem and Objectives

Technically within the e-business architecture design stack the CAM component is providing the linkage to the payload formats and transactions from the business process Schema specifications. Within each step of the business process it may have associated with it one or more physical transaction payload(s) that carries the actual information exchanged. The CAM provides the means to capture the structural, contextual and referential information about the payload formatting.

The CAM provides the critical glue between the logical model and the physical implementation, allowing representation of the ABCDE's of the interchange - *A*ssembly Structure(s), *B*usiness Use Context Rules, *C*ontent References (with optional associated data validation), *D*ata Validations (both design time assembly pre-requisites and post-assembly cross-checking.), and *E*xternal Mappings (to backend application data). The CAM defines the structural formatting and the business rules for the transaction content. This then drives the implementation step of linking the derived final contextual details to the actual application information.

### 2.6.1 Operational Requirements

In determining operational needs there are two levels and areas to include. The first level is the overall operational approach to solving large enterprise level interactions, then enterprise to small business interactions, and small business to small business interactions. Several common use cases were presented in the Introduction above for the first two interactions, while the third use is in its infancy today, (such as individuals exchanging address book entries between PDA devices). Therefore we will concentrate on the first two interactions and use areas around the enterprise needs.

The second level is a broad one based around enabling the paradigm of use for the business domain expert generally. The need is to enable the technology to be used by the functional staff across industry, rather than being restricted to specialist IT staff. Notice this requirement is

**Deleted:** 12/02/03

intrinsically linked to the fact that small business to small business interactions today are paper based and not automated electronically. So these domain expert requirements stem from the ability to specify design time details that then leads to operational use. The key new aspect here is a tight coupling between what the business domain experts specify and the actual runtime software physically uses. In today's application systems there is a clear separation of these functions, where programmers take the business domain specifications and convert these into machine instructions, and therefore there is only virtual coupling between the runtime and the design time.

The operational requirements therefore fit under two broad categories, one relates to enabling the coupling of runtime software to design artefacts and managing those consistently across a complex organization, the second relates to empowering business domain experts to be able to take on the task of constructing business processes and the associated information exchanges. Business domain experts need the ability to manage and specify information content structures and assemble and discover and reuse existing definitions of common components, such as address or invoice.

The exchange of business information as transactions is how the physical business process is facilitated. Reducing the cost and effort of managing and maintaining these business transaction interactions is therefore pivotal in defining the operational requirements. For a large enterprise this translate into reducing the headcount of staff needed, reducing the effort to migrate between implementation versions, reducing the necessary specialty skills and instead enabling general business staff. For small business it means being able to support multiple large partners diverse requests for information interchanges from a single technology base.

Summarizing these operational requirements for enabling the business information layer approach produces the following items:

- Ability to provide the enterprise with a single consistent method that provides the linkage between the business domain and the physical information exchanges
- Ability to allow multiple information domains to coexist naturally with verifiable integration across the information services layer between enterprises and domains
- Ability to drive runtime interactions from design-time component definitions
- Ability to support use by technical functional staff, not just specialist IT staff
- Ability to build consistent simple transaction definitions that can be selectively adapted for a broad range of localized uses
- Ability to create a discrete content set for exchanging where adherence to the business content rules is known and verifiable prior to transmission
- Ability to extend a base definition to include domain use details in a controlled way, including versioning
- Ability to support use of a dictionary and registry to retrieve extended central definitions and business metadata from, not just simple field level typing information
- Reusable primitive content components that business users can purpose as needed into bigger transactions and content
- Ability to apply a use context to a primitive component (i.e. address = billing.address)
- Ability to apply use context to a structure of content to select required and optional components (i.e. if (product_type=perishable then refrigeration_details = required))

**Deleted:** 12/02/03

- Ability to completely substitute content structure depending on business use localization (i.e. if (delivery=USA then ZIPcode.address else International.address))
- Ability to support multiple different legacy content structure types, not just XML

Next we consider the implementation design constraints that apply when considering the operational constraints and the implementation technology details.

### 2.6.2 Design Constraints

- Suitable for use by technical business users, not just programmers
- Declarative approach, not procedural
- Neutral approach - can support a variety of structural languages, XML, DTD, XSD, EDI, HTML, XForm, and more.
- Re-use of the XML family of specifications toolset to provide specific functionalities as needed
- Provide Registry facilitation
- Provide ability to re-use structure components
- Provide support for migration of legacy transaction formats
- Support Business Process Modelling needs for substitution transaction formats
- Support use of UN/CEFACT Core Components and Business Information Entities (BIEs)

These list technical behaviours and capabilities. From the end user functional perspective the lesson learned is that provide a single set of business transactions, while useful to provide a base point, does not accommodate the actual fielded instances that implemented business applications need. Therefore CAM must provide users with that ability to quickly assimilate standard transaction components, while being able to easily tailor them to their own environment and requirements. In combination with an OASIS Registry of domain applicable content, this provides business users "help from above", where they can reference assembly components to align with pre-developed and consistent usage, while ensuring that the logical model and the their physical implementation are tightly coupled. This avoids the lesson learned, that developing 'standard' examples leads to a gap between what people actually use. From the viewpoint of the Model Driven Architecture[2] approach this bridges the gap between the model and the physical world. Therefore CAM provides this crucial piece in the e-business architecture stack, thereby ensuring that implementers are getting uniformity and interoperability that are at the heart of providing cost-effective and maintainable electronic business interchanges.

### 2.6.3 Related Specifications

The OASIS CAM TC is doing preliminary work with other OASIS teams including the CIQ and Registry teams on utilizing CAM technology, and also the OASIS ebMS IIS team for context parameter mechanisms. Other potential collaborations include the UBL team and the CEFACT Core Components and ATG teams, and the OAG BOD development team. Contact has also been made with ISO on their work with the LISA.org projects and the potential for aligning OASIS CAM with the ISO 16642 and more related ISO specifications. See the CAM website for more details.

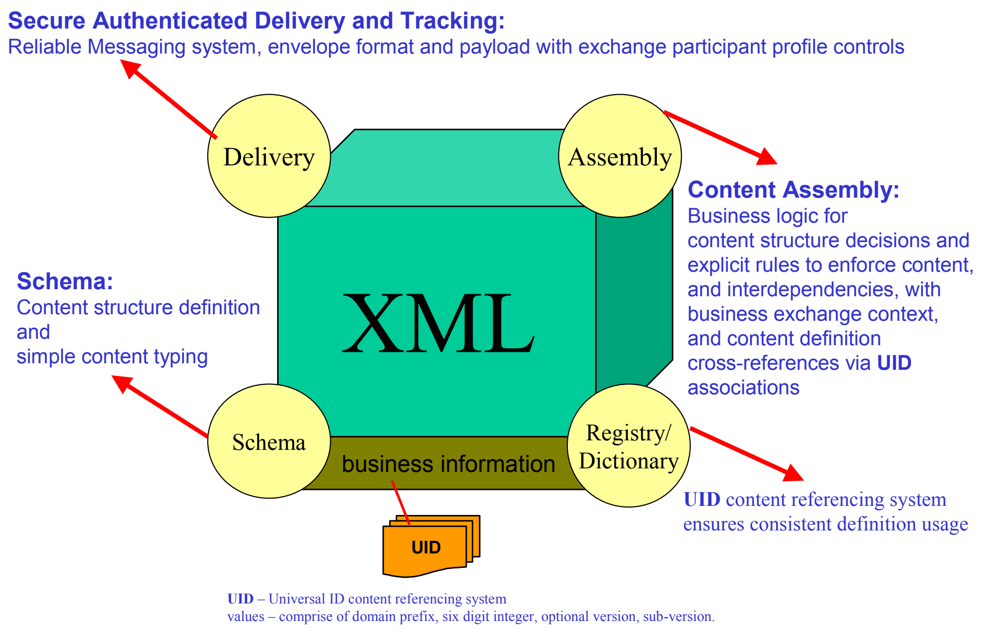
[2] For more details on Model Driven Architecture see US Government OMG approach http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

**Deleted:** 12/02/03

### 2.7.1 Implementation Aspects of Content Assembly technology

This section continues by looking at how the CAM mechanism is implemented and fits within the overall e-business architecture and XML technologies. The diagram shown in figure 2.7.1 shows how CAM integrates within the overall combination of available components designed to ensure accurate, consistent and secure information interchanges.

**Figure 2.7.1: Ensuring Information Exchange Accuracy and long-term consistency**


Secure Authenticated Delivery and Tracking:
Reliable Messaging system, envelope format and payload with exchange participant profile controls
Delivery
Assembly
Content Assembly:
Business logic for content structure decisions and explicit rules to enforce content, and interdependencies, with business exchange context, and content definition cross-references via UID associations
Schema:
Content structure definition and simple content typing
XML
Schema
business information
Registry/ Dictionary
UID
UID content referencing system ensures consistent definition usage
UID – Universal ID content referencing system
values – comprise of domain prefix, six digit integer, optional version, sub-version.

### 2.7.1 Technical Factors

With reference to figure 2.7.1, the four crucial components of consistent interchanges are laid out around the central need to control the payload of information itself.

Delivery is the first part of the equation shown here, ensuring that reliable information is received, in the expected format, from the parties that are expected to deliver the content. This part is clearly the mission of the e-business messaging components, and the associated business process and business scenario details.

Next up below the delivery are the schema definitions. Traditionally people approaching XML and e-business anticipate that all that is required is a schema of some form, and that the complete interchange is thereby described and their business integration problems are solved. Unfortunately schema is not engineered and designed to solve this complete problem, instead it

**Deleted:** 12/02/03

only describes in a limited fashion the information model for document content. So which ever kind is selected, W3C Schema, XML DTD, RELAX, or even EDI structures, these only provide the model of the complete 'set' of all possible uses of the structure. This kind of validation is suitable for detecting basic structural errors within content where performance requirements preclude more extensive and rigorous error checking. An extreme example is found in something like an 850 EDI Purchase Order, and its XML schema equivalents, where there are literally thousands of possible permutations permissible from a single schema instance. Consequently a transaction can be valid for the schema checks, but fail more extensive checks of the information integrity.

The result is that there is no way to provide consistent replicatable interchange specification using Schema or DTD. Instead we find that publishers of industry schema dictionaries are augmenting these base schema definitions with extended use case documentation using word processor documents and spreadsheets to capture the "MIG" Message Implementation Guidelines and show "samples". To solve this quandary the next item shown in figure 2.7.1 is the CAM mechanism that brings together the business implementation detail and relates it directly to the structural permutations.

These problems are obviously not new, and we know that a declarative approach leads to a finite set of implementation points, without requiring to necessarily state all possible combinations procedurally. Therefore our CAM approach adopts a declarative approach, and as a bonus this also happens to be how people logically address business definitions themselves. You state what rules apply and then you expect the machine to take those and apply them logically. So a declarative approach is much more intuitive and naturally familiar to crosscheck and verify for business users.

The CAM mechanism specifically allows the use of declarative predicates to be applied to structural components described by the transaction structure. Notice that the transaction structure can be provided by Schema, DTD, simple XML, or even EDI and legacy structures; and the CAM approach works by combining these predicates with stating input / output path locations using XPath syntax. This ability of CAM to work independently of any flavour of structure language - DTD, Schema, EDI, and so on, is a key feature. Notice that XPath itself, by providing structure reference pairs, is in fact independent of XML, since the structure node names referenced in the XPath syntax need not be of an XML structure, but will work against any formal structural system that uses explicit nodes.

Again this is not a new concept, but a refinement of prior work. For instance when using well-formed XML as the reference structure, the predicates can be provided with in-line directives to describe the content model and this also supports legacy formats, where the legacy format itself is being modelled by the XML reference structure.

Using XPath and declarative predicates in this way allow CAM to state the MIG (message implementation guidelines) in a machine accessible format. Notice that such predicates require parameter values at design and runtime to provide explicit context. For example, if my CAM template provides for both USA and UK mailing address formats, a business process will need to know explicitly which to use. Therefore CAM provides the means to couple to the business context of the Business Process Mechanism (BPM) of the implementation stack, and this will include a set of partner parameter profile context values and link from the BPM itself (the details of this mechanism are provided later in this document in the addendum).

**Deleted:** ~~12/02/03~~

Using this mechanism allows for localization and substitution structures to match the business transaction needs at run-time.

The final item in figure 2.7.1 shows the Registry and Dictionary referencing. Again while using a simple schema in isolation does provide the simple ability to datatype structural content, and also limit content to lists of tokens, it does not provide the means to express business rules about content. Therefore this mechanism is no substitute for a registry system that is able to provide extended metadata semantics about each and every component of a business transaction, along with versioning and access control.

To achieve the coupling between the structural definition and the business semantics, CAM again uses the XPath referencing system, and points each such reference to a UID reference value, that then locates the explicit metadata for that item within the Registry and / or business dictionary required. When referencing a simple existing business dictionary this can also offer just a simple inline locally defined reference (thus providing a lightweight implementation model that is not totally dependent on the presence of a Registry system always being present).

### 2.7.2 Summary

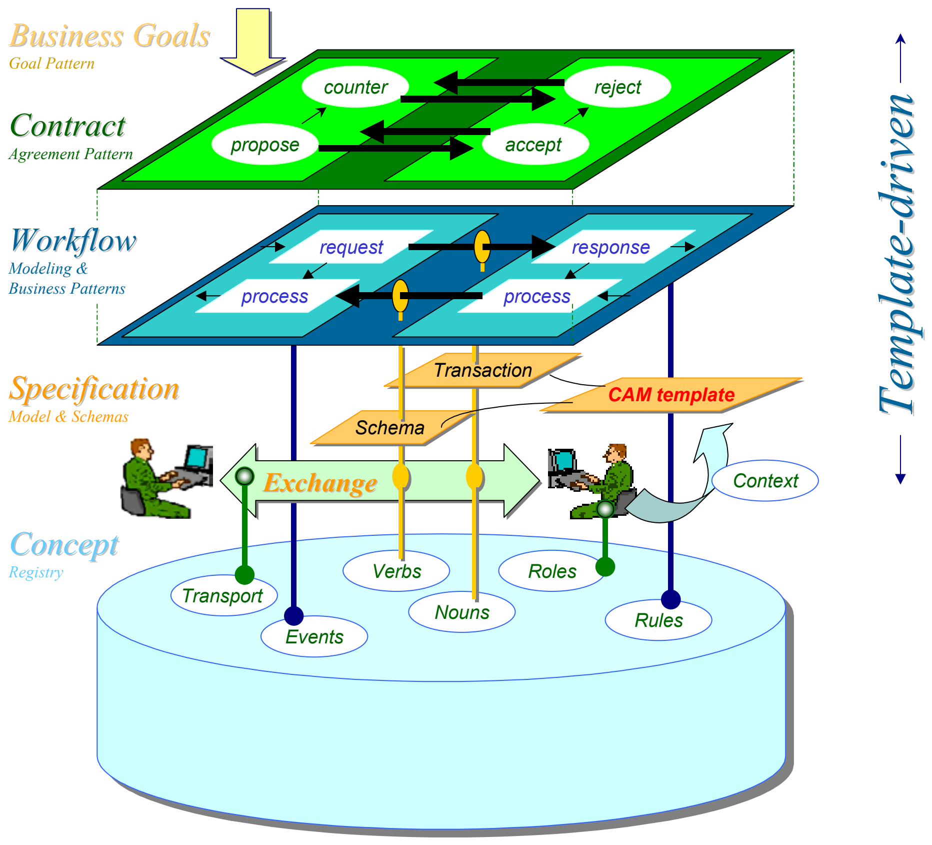
In summary what figure 2.7.1 demonstrates is the means to make consistent assembly possible, and drives adoption of simple transaction structures. Again one of the lessons learned is the "kitchen-sink" effect with schema used by itself in isolation. Such implementers strive to make a schema represent all possible combinations of an interchange, and rapidly the schema itself becomes voluminous and unwieldy, exactly the opposite of what they originally intended to create with a simple interchange definition. Whereas CAM allows implementers to create simple standalone pieces for just a limited set of use cases. Then when more use cases are required the context rules and additional structural items are compartmentalized within the CAM and immediately identifiable, instead of being lost inside complex combinations of schema structure components. Encouraging good design habits and implementation methods is definitely a major intention of the overall e-business architecture utilizing the CAM approach.

In line with this architecture, the CAM system breaks away from vendor specific mapping technologies and provides an open infrastructure that any vendor can implement, and more importantly end users can build to and then exchange easily without requiring special tools or editors.

Figure 2.7.2 below shows how CAM technology itself fits into the overall e-business components architecture.

**Deleted:** 12/02/03

**Figure 2.7.2: CAM technology and e-business architecture stack.**



The next section will provide the actual implementation details of CAM and how to construct and utilize them.

## 3.0 Pre-requisites

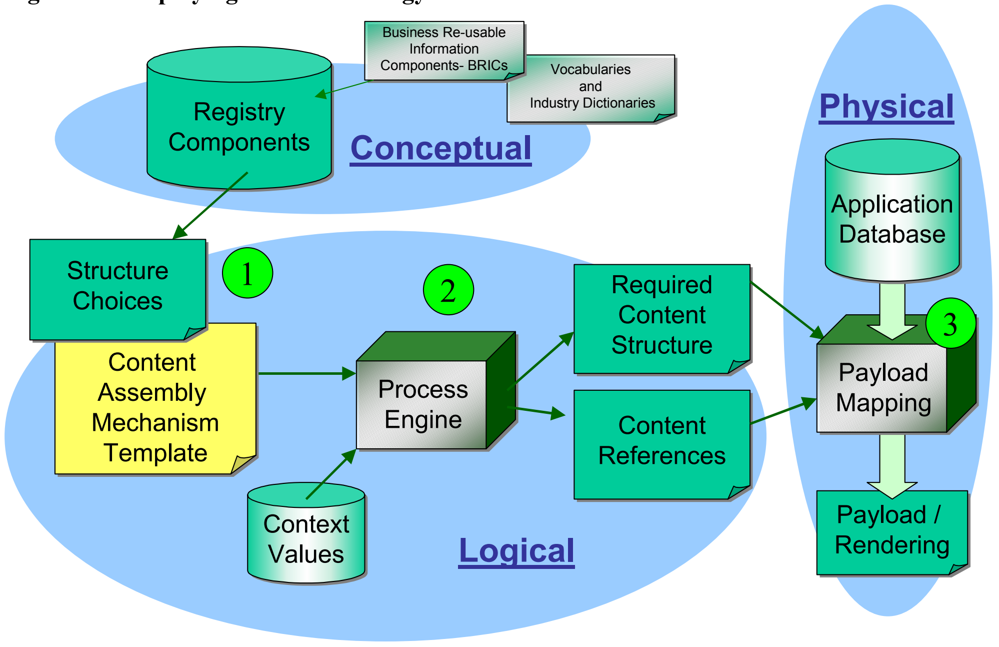
These specifications make use of W3C technologies, including XML V1.1 and XPath. It should be noted that only a subset of the XPath technology, specifically the locator sections of the XPath specification are utilized. Explicit details of XPath syntax are provided in the body of this specification. A schema definition is provided for the assembly mechanism structure. Knowledge of these technologies is required to interpret the XML sections of this document.

**Deleted:** 12/02/03

## 4.0 Content Assembly Mechanism Technical Specification

This section describes the implementation specifications for CAM. Figure 4.1 shows how implementers can integrate CAM technology into their existing systems, and then extend this out to include all aspects of the e-business information content management technologies.

**Figure 4.1: Deploying CAM technology**



In reference to figure 4.1, item 1 is the subject of this section, describing the syntax and mechanisms. Item 2 is a process engine designed to implement the CAM logic as an executable software component, and similarly item 3 is an application software component that links the e-business software to the physical business application software and produces the resultant transaction payload for the business process itself (these aspects are covered in this document in the addendum on implementation details).

Input to the conceptual model section can come from UML and similar modelling tools to define the core components and relevant re-usable business information components themselves, or can come from existing industry domain dictionaries (see the OASIS/CEFACT work on core components specifications for details).

The specification now continues with the detailing the physical realization in XML of the CAM template mechanism itself.

**Deleted:** 12/02/03

### 4.1.1 Overview

The CAM itself consists of four logical sections, and the CAM is expressed in XML syntax. These are shown in figure 4.1.1 as high-level XML structure parent elements[3].

**Figure 4.1.1: High-level parent elements of CAM (in simple XML syntax)**

```
<CAM>
        <AssemblyStructure/>
        <BusinessUseContext/>
        <ContentReference/>
        <DataValidations/>
        <ExternalMapping/>
</CAM>
```

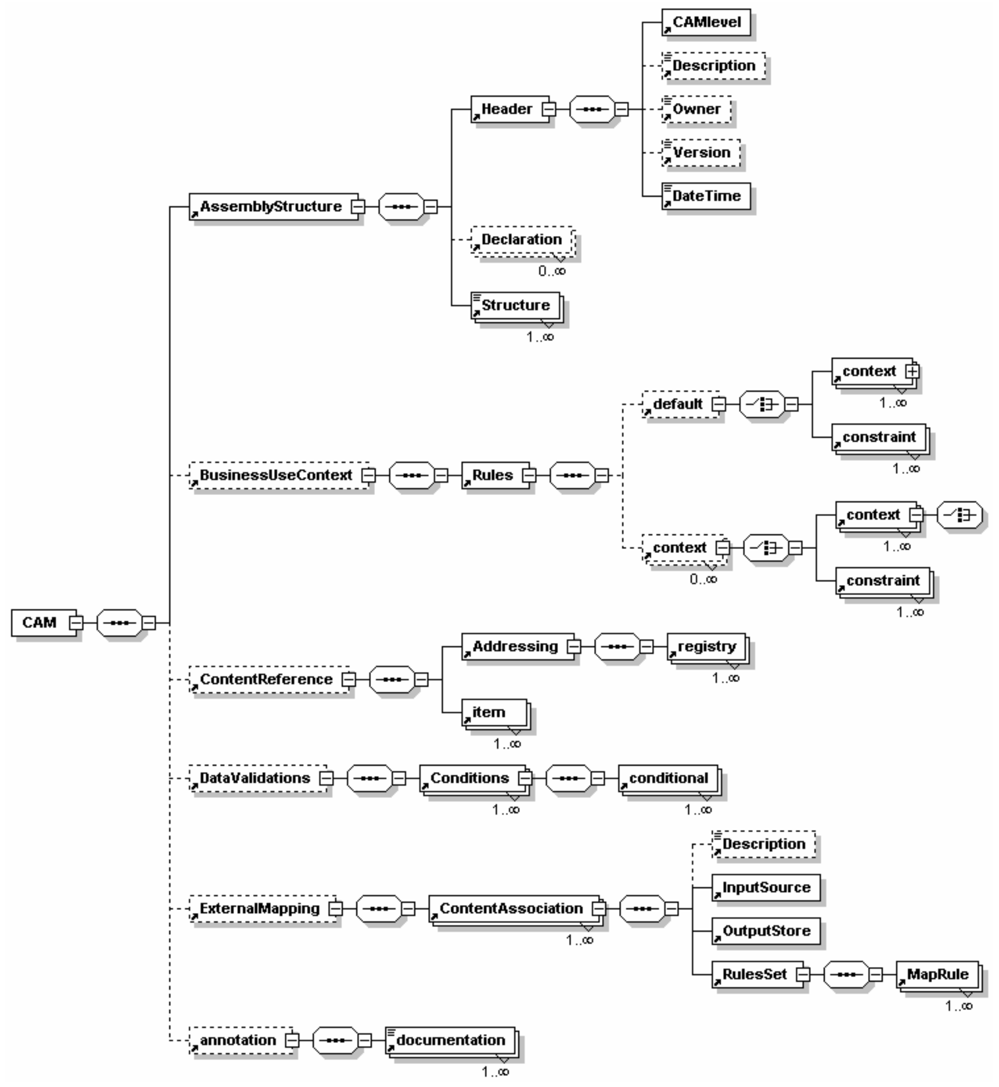
The five sections provide the ABCDE's of the interchange definition - *A*ssembly Structure(s), *B*usiness Use Context Rules, *C*ontent References (with optional associated data validation), *D*ata Validations and *E*xternal Mappings. The figure 4.1.2[4] here shows the complete hierarchy for CAM at a glance.

[3] Note: elements have been labelled using UN spellings, not North American spellings

[4] This diagrammatic syntax is preferred for all structure definitions in this document, as being neutral markup and easy to visualize for human interpretation. The notation follows basic XML V1.0 logic where "+" equals "One or more", "*" equals "Zero or more", and so on.

**Deleted:** 12/02/03

**Figure 4.1.2: Structure for entire CAM syntax at a glance**



Each of these parent items is now described in detail in the following sub-sections, while figure 4.1.3 here shows the formal schema definition for CAM (see the Appendix for machine readable Schema formats in XSD syntax). While the documented schema provides a useful structural overview, implementers should always check for the very latest version on-line to ensure conformance and compliance to the latest explicit programmatic details.

**Deleted:** 12/02/03

**Figure 4.1.3: CAM structure definition in DTD syntax**

```
<!-- CAM structure for OASIS CAM. February 10th, 2003
Modification history:
1.00 Initial
Copyright (c) 2003 OASIS. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that such redistributions retain this
copyright notice.
This CAM structure is provided "as is" and there are no expressed or implied
warranties. In no event shall OASIS be liable for any damages arising out of
the use of this structure.  -->

<!ELEMENT CAM (AssemblyStructure, BusinessUseContext?, ContentReference?,
DataValidations?, ExternalMapping?, annotation?) >

<!ELEMENT AssemblyStructure (Header, Declaration*,Structure+)>
<!ELEMENT Header (CAMlevel, Description?, Owner?, Version?, DateTime)>
<!ELEMENT CAMlevel EMPTY>
<!ATTLIST CAMlevel
          value (1 | 2 | 3) #REQUIRED >
<!ELEMENT Description (#PCDATA) >
<!ELEMENT Owner       (#PCDATA) >
<!ELEMENT Version (#PCDATA) >
<!ELEMENT DateTime (#PCDATA) >

<!ELEMENT Declaration EMPTY >
<!ATTLIST Declaration
          parameter CDATA #REQUIRED
          values    CDATA #IMPLIED
          default   CDATA #IMPLIED
          datatype  CDATA #IMPLIED >

<!ELEMENT Structure ANY >
<!ATTLIST Structure
          ID         CDATA    #IMPLIED
          reference  CDATA    #IMPLIED
          taxonomy (UID | XSD | DTD | RNG | XML)   #REQUIRED >

<!ELEMENT BusinessUseContext (Rules)>
<!ELEMENT Rules (default?, context*)>
```

**Deleted:** 12/02/03

Formatted: French (France)

```
<!ELEMENT default (context+ | constraint+ )>
<!ELEMENT context (context+ | constraint+ )>
<!ATTLIST context
        condition CDATA #REQUIRED >

<!ELEMENT ContentReference (Addressing,item+)>
<!ELEMENT Addressing (registry+)>

<!ELEMENT constraint EMPTY >
<!ATTLIST constraint
        condition CDATA #IMPLIED
        action CDATA #REQUIRED >
<!-- predicates (  excludeAttribute | excludeElement | excludeTree |
                  makeOptional | makeMandatory | makeRepeatable |
                  setChoice | setId | setLength | setLimit | setMask |
                  setValue | restrictValues | restrictValuesByUID |
                  useAttribute | useChoice | useElement | useTree |
                  useAttributeByID | useChoiceByID | useElementByID |
                  useTreeByID ) -->

<!ELEMENT DataValidations ( Conditions+ )>
<!ELEMENT Conditions ( conditional+ )>
<!ATTLIST Conditions
          condition CDATA #IMPLIED >
<!ELEMENT conditional EMPTY >
<!ATTLIST conditional
          expression CDATA #REQUIRED
          syntax (XPath | JavaScript | VB | Perl | Other) #IMPLIED
          outcome ( fail | ignore | report ) #REQUIRED
          message CDATA #IMPLIED
          test    ( always | postcheck | precheck )  #REQUIRED >

<!ELEMENT registry EMPTY>
<!ATTLIST registry
          name CDATA #REQUIRED
          access CDATA #REQUIRED
          method (URL | http | SOAP | ebXML | UDDI | Other) #REQUIRED
          description CDATA #IMPLIED
>
```

Deleted: 12/02/03

```
<!ELEMENT item EMPTY>
<!ATTLIST item
        type (noun | corecomponent | BIE | BRIC | defaultAssembly | identifier
| verb | schema | documentation) #REQUIRED
        name CDATA #IMPLIED
        UIDReference CDATA #REQUIRED
        taxonomy CDATA #REQUIRED
        registry CDATA #IMPLIED
         datatype CDATA #IMPLIED
         setlength CDATA #IMPLIED
         setmask CDATA #IMPLIED
>
<!ELEMENT annotation  (documentation+) >

<!ELEMENT documentation  (#PCDATA) >

<!ATTLIST documentation
               type (description | note | license | usage | other)   #REQUIRED
>
<!ELEMENT ExternalMapping (ContentAssociation+) >
<!ELEMENT ContentAssociation (Description?,InputSource,OutputStore,RulesSet) >
<!ELEMENT InputSource EMPTY >
<!ATTLIST InputSource
          type ( SQL | XML | EDI | TXT | ODBC | OTHER ) #IMPLIED
          location CDATA #IMPLIED >
<!ELEMENT OutputStore EMPTY >
<!ATTLIST OutputStore
          type ( SQL | XML | EDI | TXT | ODBC | OTHER ) #IMPLIED
          location CDATA #IMPLIED >

<!ELEMENT RulesSet (MapRule+) >
<!ELEMENT MapRule EMPTY >
<!ATTLIST MapRule
          output CDATA #REQUIRED
          input CDATA #REQUIRED >
```

**Formatted:** French (France)

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

**Deleted:** 12/02/03

### 4.2.1 Assembly Structures

The purpose of the AssemblyStructure section is to capture the required content structure or structures that are needed for the particular business process. This section is designed to be extremely flexible in allowing the definition of such structures. While in this document simple well-formed XML is used throughout to illustrate the usage, any fixed structured markup can potentially be utilized, such as DTD, Schema, EDI, or other (typically they will be used as substitution structures for each other). It is the responsibility of the implementer to ensure that all parties to an e-business transaction interchange can process such content formats where they are applicable to them (of course such parties can simply ignore content structures that they will never be called upon to process).

The formal structure rules for AssemblyStructure are expressed by the syntax in figure 4.2.2 below. The figure 4.2.1 here shows a simple example for an AssemblyStructure using two different structures for content.

**Figure 4.2.1: Example of Structure and format for AssemblyStructure**

```
<AssemblyStructure>
 <Header>
     <CAMlevel value="1"/>
     <Description>Example 4.2.1 using structures</Description>
 </Header>
 <Structure taxonomy="…">
       <!-- the physical structure of the required content goes here, and can be
       a schema instance, or simply well-formed XML detail, see example below in
       figure 4.2.2 -->
 </Structure >
</AssemblyStructure>
```

In the basic usage, there will be just a single structure defined in the AssemblyStructure / Structure section. However, in the more advanced use, then multiple substitution structures may be provided, and these can also be included from external sources, and nesting of assemblies; see the section below on Advanced Features for details.

**Deleted:** 12/02/03

To provide the direct means to express content values within the structure syntax the following two methods apply. A substitution value is indicated by two percentage signs together “%%”, while any other value is assumed to be a fixed content value. Figure 4.2.2 shows examples of this technique.

**Figure 4.2.2: Substitution and fixed parameter values, with a well-formed XML structure**

```
<AssemblyStructure>
 <Header>
     <CAMlevel value="1"/>
     <Description>Example 4.2.2 Well-formed XML structure</Description>
 </Header>
 <Structure taxonomy="XML">
   <Items CatalogueRef="2002">
     <SoccerGear>
     <Item>
            <RefCode>%%</RefCode>
            <Description>%%</Description>
            <Style>WorldCupSoccer</Style>
            <UnitPrice>%%</UnitPrice>
     </Item>
     <QuantityOrdered>%%</QuantityOrdered>
     <SupplierID>%%</SupplierID>
     <DistributorID>%%</DistributorID>
     <OrderDelivery>Normal</OrderDelivery>
     <DeliveryAddress/>
     </SoccerGear>
   </Items>
 </Structure>
</AssemblyStructure>
```

**Formatted:** French (France)

Referring to figure 4.2.2, the “2002”, “WorldCupSoccer” and “Normal” are fixed values that will always appear in the payload transaction at the end of the CAM process.

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

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

**Deleted:** 12/02/03

### 4.3.1 Business Use Context Rules

Once the assembly structure(s) have been defined, then the next step is to define the context rules that apply to that content. The technique used is to identify a part of the structure by pointing to it using an XPath locator reference, and then also applying an assertion using one of the structure predicates provided for that purpose (an optional comparison evaluation expression can also be used with the XPath locator reference where applicable). Figure 4.3.1 shows these structure assertion predicates.

**Figure 4.3.1: The assertion predicates for BusinessUseContext**

```
excludeAttribute()
excludeElement()
excludeTree()
makeOptional()
makeMandatory()
makeRepeatable()
setChoice()
setId()
setLength()
setLimit()
setMask()
setValue()
restrictValues()
restrictValuesByUID()
useAttribute()
useChoice()
useElement()
useTree()
useAttributeByID()
useChoiceByID()
useElementByID()
useTreeByID()
startBlock()
endBlock()
```

Each predicate provides the ability to control the cardinality of elements within the structure, or whole pieces of the structure hierarchy (children within parent). An example of such context rules use is provided below, and also each predicate and its' behaviour is described in the matrix in figure 4.3.3 below. Also predicates can be used in combination to provide a resultant behaviour together, and example is using makeRepeatable() and makeOptional() together on a

**Deleted:** 12/02/03

structure member.

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

Referring to the structure in the example shown in figure 4.2.2 then figure 4.3.2 here provides examples of context based structural predicate assertions. Notice that such context rules can be default ones that apply to all context uses of the structure, while other context rules can be grouped and constrained by a XPath locator rule expression. There are three styles of such XPath expressions:

- XPath expression refers to structure members directly and controls their use
- XPath expression refers to structure member and contains condition of its value
- XPath expression refers to token that is not member of structure, but is a known external control value from the profile of the business process itself.

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

**Figure 4.3.2: Syntax example for BusinessUseContext**

```
<BusinessUseContext>
<Rules>
   <default>
     <context> <!-- default structure constraints -->
      <constraint action="makeRepeatable(//SoccerGear)" />
      <constraint action="makeMandatory(//SoccerGear/Items/*)" />
       <constraint action="makeOptional(//Description)" />
       <constraint action="makeMandatory(//Items@CatalogueRef)" />
       <constraint action="makeOptional(//DistributorID)" />
       <constraint action="makeOptional(//SoccerGear/DeliveryAddress)" />
     </context>
   </default>
   <context condition="token='//SoccerGear/SupplierID' and
                                       contains(value,'SuperMaxSoccer')">
     <constraint action="makeMandatory(//SoccerGear/DeliveryAddress)"/>
   </context>
   <context condition="token='%DeliveryCountry%' and contains(value,'USA'">
     <constraint action="useTree(//SoccerGear/DeliveryAddress/USA)"/>
   </context>
</Rules>
```

**Deleted:** 12/02/03

```
</BusinessUseContext>
```

Referring to the XPath expressions in figure 4.3.2, examples of all three types of expression are given to show how the XPath expressions are determined and used. For external control values the special name "token=" is used and the XPath does not contain the leading '//' delimiters to denote such items, instead a pair of % indicators denote a substitution value.

Referring to figure 4.3.3 below, the following applies:

| | |
|---|---|
| //elementpath | XPath expression resolving to an element(s) in the structure |
| //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 |
| 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 |

**Figure 4.3.3: Matrix of predicates for BusinessUseContext declarations.**

| Predicate | Parameter(s) | Description |
|---|---|---|
| `excludeAttribute()` | `//elementpath@attributename` | Conditionally exclude attribute from structure |
| `excludeElement()` | `//elementpath` | Conditionally exclude element from structure |
| `excludeTree()` | `treepath` | Conditionally exclude a whole tree from structure |
| `makeOptional()` | `//elementpath` | Conditionally allow part of structure to be optional |
| `makeMandatory()` | `//elementpath` | Conditionally make part of structure required |
| `makeRepeatable()` | `//elementpath` | Conditionally make part of structure occur one or more times in the content |

**Deleted:** 12/02/03

| Predicate | Parameter(s) | Description |
|---|---|---|
| `setChoice()` | `//elementpath` | Indicate that the first level child elements below the named elementpath are actually choices that are conditionally decided with a useChoice() predicate action |
| `setId()` | `//elementpath,IDvalue` | Associate an ID value with a part of the structure so that it can be referred to directly by ID |
| `setLength()` | `//memberpath` | Control the length of content in a structure member |
| `setLimit()` | `//elementpath, count` | For members that are repeatable, set a count limit to the number of times they are repeatable |
| `setMask()` | `//memberpath, mask, syntax` | Assign a regular expression or picture mask to describe the content. |
| `setValue()` | `//memberpath, value` | Place a value into the content of a structure |
| `restrictValues()` | `//memberpath, [valuelist]` | Provide a list of allowed values for a member item |
| `restrictValuesByUID()` | `//memberpath, UIDreference` | Provide a list of allowed values for a member item from a registry reference |
| `useAttribute()` | `//elementpath@attributename` | Require use of an attribute for a structure element and exclude other attributes |
| `useChoice()` | `//elementpath` | Indicate child element to select from choices indicated using a setChoice() predicate. |
| `useElement()` | `//elementpath` | Where a structure definition includes choices indicate which choice to use. |

**Deleted:** 12/02/03

| Predicate | Parameter(s) | Description |
|---|---|---|
| `useTree()` | `//treepath` | Where a structure member tree is optional indicate that it is to be used |
| `useAttributeByID()` | `StructureID` | As per useAttribute but referenced by structure ID defined by SetId or in-line ID assignment |
| `useChoiceByID()` | `StructureID` | As per useChoice but referenced by structure ID defined by SetId or in-line ID assignment |
| `useTreeByID()` | `StructureID` | As per useTree but referenced by structure ID defined by SetId or in-line ID assignment |
| `useElementByID()` | `StructureID` | As per useElement but referenced by structure ID defined by SetId or in-line ID assignment |
| `StructureType()` | `StructureID, type` | Denote the type of content format within an assembly structure, such as X12EDI, EDIFACT, XML, CDF, HL7, FIX and so on. |
| `startBlock()` | `StartBlock, [StructureID]` | Denote the beginning of a logical block of structure content. The StructureID 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. |

**Deleted:** 12/02/03

| Predicate | Parameter(s) | Description |
|---|---|---|
| `endBlock()` | `endBlock, [StructureID]` | 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. |

The predicates shown in figure 4.3.3 can also be used as in-line statements within an assembly structure, refer to the section on advanced usage to see examples of such use.

#### 4.3.1.1 XPath syntax functions

The W3C XPath specification provides for extended functions. The CAM XPath usage exploits this to add the following set of related conditional evaluations. The base XPath provides the "contains" function for examining content, the functions shown in figure 4.3.4 extend this to provide the complete set of familiar logical comparisons.

**Figure 4.3.4 XPath Comparator functions.**

| Comparator | Format | Description |
|---|---|---|
| `equal()` | `equal (value, 'value')` | Conditionally check for a matching value |
| `NOTequal()` | `NOTequal (value, 'value')` | Conditionally check for a non-matching value |
| `greater ()` | `greater (value, 'value')` | Conditionally check for a greater value |
| `lessthan ()` | `lessthan (value, 'value')` | Conditionally check for a lesser value |
| `greaterEQ ()` | `greaterEQ (value, 'value')` | Conditionally check for a greater than or equal to value |
| `lessthanEQ ()` | `lessthanEQ (value, 'value')` | Conditionally check for a lesser or equal value |

**Deleted:** 12/02/03

| Comparator | Format | Description |
|---|---|---|
| `begins ()` | `begins (value, 'value')` | Conditionally check for a string matching the front part of value, equal or longer strings match. |
| `ends ()` | `ends (value, 'value')` | Conditionally check for a string matching the end part of value, equal or longer strings match. |
| `lookup ()` | `lookup (value, 'call address')` | Conditionally check for a string being located in a list referenced by a call address. *Note: call address is defined in ContentReference section.* |
| `member ()` | `member (valuelist, '[value,value,value,…]')` | Conditionally check for a string matching the value of a member of a list of unique values. |

Only the extensions to XPath syntax shown in figure 4.3.4 are supported. Additionally XPath syntax support is limited to locator expressions only, and excludes the XPath features for manipulation of content directly as a result set.

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

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

**Deleted:** 12/02/03

### 4.4.1 Content Referencing

The purpose of content referencing is to provide additional information about the metadata of each item of the structure and the content model and associated data typing when applicable. It also provides crosswalk information to a dictionary of noun definitions, and thus potentially from your physical implementation to the logical BRIC components themselves. This ability to provide crosswalk implementation details is vital to maximizing interoperability and re-use within the optimal e-business architecture and also allowing the use of modelling tools and object-oriented technologies.

The example in figure 4.4.1 shows the content referencing for the structure in figure 4.2.2, and shows how multiple dictionary domains (namespaces) can be accommodated in blending a composite structure together, while also allowing extensions using locally defined content items that are not part of any dictionary. The use cases for content referencing can be summarized as:

- No registry dictionary is available so all content referencing is locally defined
- A default content model can be defined using the predicates, (these however will not take precedence over explicit rules in the BusinessContext section), but will override any inline predicates within AssemblyStructure
- A single registry and industry domain is referenced only
- Multiple registry domains are referenced
- Combinations of all of the above

Further notes on aspects of the particular syntax instructions for content referencing are given below.

**Figure 4.4.1: Example of Content Referencing for AssemblyStructure**

```
<ContentReference>
   <Addressing>
      <registry name="SGIR" access="registry.sgir.org:1023" method="URL"
                     description="Sporting Goods Industry Registry"/>
      <registry name="SGIRWSDL" access="registry.sgir.org:1025" method="WSDL"
                     description="Sporting Goods Industry Registry"/>
      <registry name="UN" access="registry.un.org:9090" method="ebXML"
 description="United Nations EDIFACT Registry"/>
      <registry name="UPS" access="registry.ups.com:7001" method="URL"
                     description="United Parcels Service Registry"/>
      <registry name="USPS" access="registry.usps.gov:8080" method="URL"
                     description="United States Postal Service Registry"/>
      <registry name="LocalSQL" access="rdbms.mybusiness.com:4040" method="SQL"
                     description="Local Product Database stored procedures"/>

   </Addressing>
```

Deleted: 12/02/03

```
   <item type="noun" name="RefCode"
            UIDReference="SGIR010027" taxonomy="UID" registry="SGIR"/>
   <item type="noun" name="Description"
            UIDReference="SGIR010050" taxonomy="UID" registry="SGIR"/>
   <item type="noun" name="Style"
            UIDReference="SGIR010028" taxonomy="UID" registry="SGIR"/>
   <item type="noun" name="SupplierID"
            UIDReference="SGIR010029" taxonomy="UID" registry="SGIR"/>
   <item type="noun" name="CatalogueRef" UIDReference="none" taxonomy="none"
            datatype="text" setlength="4" setMask="'\d\d\d\d',PERL" />
   <item type="noun" name="DistributorID" UIDReference="none" taxonomy="none"
            datatype="text" setlength="30" />
   <item type="noun" name="UnitPrice"
            UIDReference="070010" taxonomy="EDIFACT" registry="UN"/>
   <item type="noun" name="QuantityOrdered"
            UIDReference="070011" taxonomy="EDIFACT" registry="UN"/>
   <item type="noun" name="OrderDelivery"
            UIDReference="UPS050050" taxonomy="UID" registry="UPS"/>
   <item type="defaultAssembly" name="DeliveryAddress"
            UIDReference="USPS090081:01:05" taxonomy="UID" registry="USPS"/>
 </ContentReference>
```

Each of the modes of determining a content reference is shown in figure 4.4.1, along with the use of the Registry addressing section to link between the logical and physical addresses of Registry content. Notice that with locally defined items (UIDReference="none" taxonomy="none"), then one of the optional predicate[5] parameters is used to further define the content model (e.g.: setlength="4").

Typically references are to nouns within the assembly structure, but can also be to a composite item as a defaultAssembly, as is the case with the DeliveryAddress example (such defaultAssembly items can equate to BRICs, and have an <as:include> for their structure content, see details in the advanced techniques section below).

Similarly the taxonomy preferred is that of the UID system, however where legacy schemes exist such as EDI element dictionary numbering, then the UIDReference can accommodate such values accordingly. The UID values themselves are composed of an alpha prefix representing an acronym for the domain organization, followed by a simple 6-digit numeric. Optionally a UID can also have a suffix of colon, major version, and colon, minor version, to provide version control. When the version information is omitted then the UID reference points to the latest current information from the registry by default.

[5] Implementation note: the XPath parameter for the predicate defaults to the name value to identify the item within the assembly structure

**Deleted:** 12/02/03

If an item refers to a registry acronym that is not defined in the //Addressing/registry statement, then a warning should be issued, but process can continue. Similarly, warnings should be generated for assembly structure members that do not have ContentReference entries, but all such items will have a default content model of type="text" as a simple string type. Notice that type="[datatype]" supports the W3C Schema data types by default.

The content referencing is intended to provide assembly metadata for the information content model during assembly. The next section can handle post-assembly processing and validation requirements on receipt of content, as well as on creation of content.

### 4.5.1 Data Validations

This section provides the means to verify information content of transaction instances built from CAM structure and context rules. This verification can occur at design/runtime during creation of a content instance, and also some verification can occur post-content creation, typically upon content receipt by some other party. The DataValidation section is thus more likely to be tied to a particular production implementation and environment, particularly for post-content creation checks. However, users can choose to provide generic CAM formulas that apply to all implementations within a domain using XPath expressions as allowed within CAM, and then allow implementers to extend these for particular local instances.

Since validation rules are highly syntax dependent, parties using them must agree on their support for the expression syntax(es) selected that are not CAM compatible XPath expressions or calls via the Registry call mechanism defined within the Content Reference section . (Within the e-business architecture this would occur at the business agreement level where parties specify support for particular processing environment features – see figure 2.7.2).

**Figure 4.5.1: Example of Data Validations for AssemblyStructure**

```
<DataValidations>
   <Conditions
       condition="token='%DeliveryCountry%' and contains(value,'USA'">
       <conditional
              expression="'//UnitPrice' and greater(value,'0.00')"
              syntax="XPath" outcome="fail"
              message="Item price not valid / missing" test="always"/>
       <conditional
              expression="'//RefCode + //UnitPrice' and
              lookup(value,'SGIRWSDL:unitprice_check')" outcome="report"
              message="Unit price value does not match catalog" test="always"/>
       <conditional
              expression="'//SupplierID' and
              lookup(value,'SGIRWSDL:supplierID_check')" outcome="fail"
              message="Unknown Supplier ID" test="always"/>
```

**Deleted:** 12/02/03

```
      <conditional
            expression="'//DistributorID' and
            lookup(value,'SGIRWSDL:distributor_check')" outcome="fail"
            message="Unknown distributor ID" test="postcheck"/>
      <conditional itemRef="//QuantityOrdered" conditioncheck="value"
            expression="'//QuantityOrdered' and
            lookup(value,'LocalSQL:quantityOnHand()')" outcome="report"
            message="Item not available / backordered" test="postcheck"/>
   </Conditions>
</DataValidations>
```

The conditional section shown in figure 4.5.1 shows a variety of methods, from in-line XPath expressions, through to remotely executed 'verbs' from a registry as a web service, through to SQL stored procedures. Notice that WSDL is used as the interface example to web services, and the WSDL description may involve passing of parameters (such as the //RefCode to verify the //UnitPrice). These details can be determined through the programmatic interface to the particular lookup reference service[6].

Again, support for these methods are dependent on the business agreements between parties and the capabilities and requirements of parties. Some parties may simply opt to not support DataValidation conditions, or only those using XPath, and so on. Because of this, it is anticipated that the DataValidation section will provide useful hints to parties on requirements for a complete and accurate business exchange. How far they will be able to support these, and how many local extensions are built using the base mechanisms provided in the syntax methods of DataValidation will depend on the maturity of the information systems of the implementers. Since these mechanisms and section are least accessible to business users, and most accessible to programmers the initial intent here is to provide basic functionality that is useful to a broad range of business use. It is not intended to replace extensive, proprietary and complex application logic in backend systems.

For a simple implementation it is suggested that basic information checks are instituted using the provided XPath syntax and comparator functions. Then later more extended checks can be supported via external calls. Similarly if the outcome is marked as 'ignore' or 'report', this means that early implementers can treat these checks simply as documentation notes as to the checking that backend complex application logic will perform, until they are more fully able to support the recovery and post-processing required via their business processing service components.

### 4.6.1 Discrete Value List Support ("Codelists")

[6] Note: OASIS Registry support for CAM services through this specification is covered separately in the addendum of this specification document.

**Deleted:** 12/02/03

This note discusses support for code list functionality. Over 50% of traditional EDI transaction content is comprised of code values that are referenced and shared between trading partners. CAM provides two XPath functions to directly implement these capabilities. Firstly is the `member()` function that allows specific code values to be specified in the CAM template itself. Second is the `lookup()` function that supports the use of code values external to the template itself, where one or more parameters are passed into it. The configuration of the lookup function external access is configured through the Content Reference section Registry definition statements. See the examples provided in that section, and in the validation examples in figure 4.5.1 above. Nested code list lookups can be configured using nested `<conditions>` expressions.

The next section details further advanced features that can be used to augment the basic CAM functionality.

**Deleted:** 12/02/03

### 4.7.1 Advanced Features

This section details extended uses of the basic features. For this first release this is focused on three aspects, in-line use of predicates within structures, non-XML structure content referencing, and including external content into a CAM.

## *4.6.1 In-line use of predicates and references*

The figure 4.6.1.1 here shows an extended example for an AssemblyStructure using two different structures for content and the in-line statements indicating those content selections. The in-line commands are inserted using the "as:" namespace prefix, to allow insertion of the command statements wherever they are required. These in-line commands compliment the predicates used within the <BusinessUseContext> section of the assembly. The table in figure 4.6.1.2 gives the list of these in-line statements and the equivalent predicate form where applicable.

**Figure 4.6.1.1: Example of Multiple substitution structures for AssemblyStructure**

```
<CAM xmlns:as="http://www.oasis-open.org/committees/cam">
 <AssemblyStructure>

   <Structure as:choiceID="FirstOne" taxonomy='XML'>
         <!-- the physical structure of the required content goes here -->
   </Structure >

   <Structure as:choiceID="SecondOne" taxonomy='XML'>

        <createTroubleTicketByValueResponse as:choiceID="OptionA">
              <!-- the physical structure of the required content goes here -->
        </createTroubleTicketByValueResponse>

        <createTroubleTicketByValueResponse as:choiceID="OptionB">
              <!-- the physical structure of the required content goes here -->
        </createTroubleTicketByValueResponse>
   </Structure >

 </AssemblyStructure>
</AssemblyScript>
```

Reviewing figure 4.6.1.1 there are two main substitution structures, and within the second there are also two sub-structure choices. The actual behaviour and which structure content is included in the physical content is controlled by predicate statements within the <BusinessUseContext> section of the assembly.

**Deleted:** 12/02/03

The in-line statements available are detailed in the table shown in figure 4.6.1.2, and where the in-line command entry is marked "not applicable" then these items can only be used within the <BusinessUseContext> section. Also where there is both a predicate statement and an in-line command, then the predicate statement overrides and takes precedent.

**Figure 4.6.1.2: Matrix of in-line statement commands and predicate commands.**

| Predicate | In-line Command | Notes |
|---|---|---|
| `excludeAttribute()` | `Not applicable` | |
| `excludeElement()` | `Not applicable` | |
| `excludeTree()` | `Not applicable` | |
| `makeOptional()` | `as:makeOptional="true"` | Make part of structure optional, or make a repeatable part of the structure optional (i.e. occurs=zero) |
| `makeMandatory()` | `as:makeMandatory="true"` | Make part of structure required |
| `makeRepeatable()` | `as:makeRepeatable="true"` `as:setLimit="5"` | Make part of structure occur one or more times in the content; the optional as:limit="nnnn" statement controls how many times maximum that the repeat can occur. |
| `setChoice()` | `Not applicable` | |
| `setId()` | `as:choiceID="label"` | Associate an ID value with a part of the structure so that it can be referred to directly by ID |
| `setLength()` | `as:setLength="nnnn"` | Control the length of content in a structure member |
| `setLimit()` | `as:setLimit="nnnn"` | For members that are repeatable, set a count limit to the number of times they are repeatable |
| `setMask()` | `as:setMask=` `"'Mask','syntax'"` | Assign a regular expression or picture mask to describe the content. |
| `setValue()` | `as:setValue="string"` | Place a value into the content of a structure |
| `restrictValues()` | `as:restrictValues=` `"[valuelist]"` | Provide a list of allowed values for a member item |
| `restrictValuesByUID()` | `as:restrictValuesByUID=` `"UID"` | Provide a list of allowed values for a member item from an registry reference |
| `useAttribute()` | `Not applicable` | |
| `useChoice()` | `Not applicable` | |

Deleted: 12/02/03

| Predicate | In-line Command | Notes |
|---|---|---|
| `useElement()` | `as:useElement="true"` | Where a structure definition includes choices indicate which choice to use. |
| `useTree()` | `as:useTree="true"` | Where a structure member tree is optional indicate that it is to be used. |
| `useAttributeByID()` | `Not applicable` | |
| `useChoiceByID()` | `Not applicable` | |
| `useTreeByID()` | `Not applicable` | |
| `useElementByID()` | `Not applicable` | |
| `StructureType()` | `as:structureType="string"` | Denote the type of content format within an assembly structure, such as X12EDI, EDIFACT, XML, CDF, HL7, FIX and so on. |
| `Not applicable` | `<include>URL` `</as:include>` | 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 next figure 4.6.1.3 shows some examples of using these in-line commands within a structure.

**Figure 4.6.1.3: Use of in-line commands with a well-formed XML structure**

```
<AssemblyStructure xmlns:as="http://www.oasis-open.org/committees/cam">
 <Structure taxonomy='XML'>
   <Items CatalogueRef="2002">
     <SoccerGear>
       <Item as:makeRepeatable="true">
              <RefCode as:makeMandatory="true" as:setLength="10">%%</RefCode>
              <Description>%%</Description>
              <Style>WorldCupSoccer</Style>
              <UnitPrice as:setMask="'999.99','SQL'">%%</UnitPrice>
       </Item>
       <QuantityOrdered as:setMask="'999','SQL'">%%</QuantityOrdered>
       <SupplierID as:makeMandatory="true">%%</SupplierID>
       <DistributorID>%%</DistributorID>
       <OrderDelivery>Normal</OrderDelivery>
```

**Formatted:** French (France)

**Deleted:** 12/02/03

```
      <DeliveryAddress/>
     </SoccerGear>
   </Items>
 </Structure>
</AssemblyStructure>
```

The next section shows the use of non-XML structure, and it should be noted that in-line commands cannot be used with non-XML structures, all such structures require the use of predicates within the <BusinessUseContext> section of the assembly.

## 4.6.2 Non-XML structure referencing

This section shows how the CAM system supports referencing to non-XML content as shown in figure 4.6.2.1 for a legacy EDI structure definition. The XPath system can reference to nodes within such structures using an appropriate node-referencing scheme that is pre-determined, (for instance in an EDI transaction this would be segment identifier and field number within the segment as the node name).

**Figure 4.6.2.1: An EDI example of referencing non-XML content structures**

```
<AssemblyStructure xmlns:as="http://www.oasis-open.org/committees/cam">
 <Structure as:choiceID="EDI850" as:structureType="X12EDI" taxonomy='XML'>

  <EDI Type="ASCII" Version="4040" Standard="X12">
   <TransactionSet ID="850" Name="Purchase Order" Note="">
    <Segment ID="ST" Name="Transaction Set Header" Req="M" MaxUse="1">
      <Element ID="01" Name="Transaction Set Identifier Code" Req="M"
             Type="ID" MinLength="3" MaxLength="3"
             Note="The transaction set identifier 'ST01' is used by the
             translation routines of the interchange partners to select the
             appropriate transaction set definition 'e.g., 810 select the
             Invoice Transaction Set'."/>
      <Element ID="02" Name="Transaction Set Control Number" Req="M"
             Type="AN" MinLength="4" MaxLength="9"/>
      <Element ID="03" Name="Implementation Convention Reference" Req="O"
             Type="AN" MinLength="1" MaxLength="35"
             Note="The implementation convention reference 'ST03' is used by
             the translation routines of the interchange partners to select
             the appropriate implementation convention to match the
             transaction set definition."/>
    </Segment>
    <!-- then more segments follow here... -->
```

**Formatted:** French (France)

**Deleted:** 12/02/03

```
   </TransactionSet>
  </EDI>
 </Structure >
</AssemblyStructure>
```

The EDI structure definition in figure 4.6.2.1 is one system for describing an EDI structure; another example would be the IGML system (http://www.igml.org) or similar systems, or a very simple system using substitution tokens as shown in figure 4.6.2.2, and then the UN/EDIFACT transaction IMPDEF system.   Another alternate to using well-formed XML as the structure example is to use a DTD or Schema instance itself.

All these may be considered for use with assembly as the business needs require.

**Figure 4.6.2.2:  Tokens EDI example of referencing non-XML content structures**

```
<AssemblyStructure>
 <Structure as:choiceID="Healthcare Transaction" as:structureType="Tokens"
taxonomony='EDI'>
ISA*00*%%*00*%%*01*%%*01*Interchange Rec*010404*1031*U*00200*000025331*0*I*:~
GS*AA*%%*%%*20010404*1031*000000000*T*004010X097~
ST*276*0001~
BHT*0010*13**%%~
HL*1**%%*1~
NM1*PR*2*%%*****PI*%%~
HL*2*1*%%*1~
NM1*41*2*%%*****46*X67E~
HL*3*2*%%*1~
NM1*1P*2*%%*****SV*987666~
HL*4*3*22*0~
DMG*D8*%%*M~
NM1*QC*1*%%*%%****MI*%%~
TRN*1*%%~
REF*BLT*%%~
AMT*T3*%%~
REF*1K*%%~
REF*BLT*%%~
AMT*T3*%%~
SE*%%*0001~
GE*1*000000000~
IEA*1*%%~
 </Structure >
```

**Deleted:** 12/02/03

```
</AssemblyStructure>
```

The tokens method using "%%" for the replacement items as shown in figure 4.6.2.2 is easily adapted to suit a wide variety of non-XML content structures.

An example of an XPath predicate reference would `makeRepeatable(\\HL::NM1)` for a block of lines, and `makeOptional(\\REF)` to indicate a segment line or `makeMandatory(\\AMT\01)` to indicate a field within a segment.

In each case partners using these systems must agree on the processing rules for the non-XML content they are intending to process. Industry standards bodies can also define such rules as extensions to the base CAM system for legacy payloads within their own domain. Implementers may provide a generic tokens method as a default for non-XML content since it can handle a broad range of such content.

## 4.6.1 Including External Structures into CAM

In the first release of CAM, this facility is restricted to including external structure definitions within the <structure> section of the document only. This ensures a reasonable level of complexity for implementations, while allowing use of existing structure definitions such as DTD or Schema specifications easily and simply. The external structure can also be a CAM BRIC structure emitted from a modelling tool, or similar means of allowing combinations of structure components together to make a complete whole. Such tools can easily use in-line commands within the structure to align the assembly process with the model definitions.

The example in figure 4.6.3.1 shows syntax for including an external structure or composite fragments of structure together for use within assembly. The business rules within the <BusinessUseContext> section can then reference these structure items to complete the functionality required.

**Figure 4.6.3.1: Use of <as:include> commands within an assembly XML structure**

```
<CAM xmlns:as="http://www.oasis-open.org/committees/cam">
       <AssemblyStructure>
        <Structure taxonomy='XML'>
          <BusinessInvoice>
              <as:include>
                     http://www.uncefact.org/strct/invoice.xml
              </as:include>
              <billingAddress>
              <as:include>
                     http://www.uncefact.org/strct/address.xml
              </as:include>
              </billingAddress>
          </BusinessInvoice>
        </Structure>
```

Deleted: 12/02/03

```
       </AssemblyStructure>
       <BusinessUseContext/>
       <ContentReference/>
       <DataValidations/>
</CAM>
```

Include statements are assumed to retrieve consistent pieces of content, and not fragments that do not parse as a contiguous whole.

The document referenced by an <as:include> statement may contain one or more further <as:include> statements, however, if this contains a circulatory reference, then processing of the include statements should fail and stop with an appropriate error message. Nested including provides direct support for core component mechanisms and BRIC components that can be assembled together.

## 4.6.2 Object Oriented Includes Support

In order to augment the ability of modelling tools to generate CAM structure objects, the include statement has optional parameters attached to it of extends=" " and implements=" ".

**Figure 4.6.2.1 Example of CAM include with OO extensions**

```
<CAM xmlns:as="http://www.oasis-open.org/committees/cam">
       <AssemblyStructure>
        <Structure taxonomy='XML'>
          <BusinessInvoice>
             <as:include extends="SGIR:UN034500" implements="SGIR:UN034750">
                    http://www.uncefact.org/strct/invoice.xml
             </as:include>
             <billingAddress>
             <as:include extends="SGIR:CIQ010100" implements="SGIR:CIQ010350">
                    http://www.uncefact.org/strct/address.xml
             </as:include>
             </billingAddress>
          </BusinessInvoice>
        </Structure>
       </AssemblyStructure>
       <BusinessUseContext/>
       <ContentReference/>
       <DataValidations/>
</CAM>
```

**Formatted:** French (France)

**Deleted:** 12/02/03

The `extends` and `implements` parameters are optional, and the CAM processor does not parse the information contained in them. Essentially they are external notes for use in modelling tools.

Typical values however may consist of a registry alias prefix, with UID reference value that denote semantic content.

The next section reviews the requirements of the last step of the assembly process, which bridges to the physical business application and data content. It provides the means to formalize that step beyond the assembly and the linkage to the physical systems.

### 4.8.1 Implementation Notes

These notes are provided to assist implementers developing assembly software. The specifications provided here are not intended to describe a complete system for implementers to build. They provide the underlying specifications that implementers can use to construct products that support the assembly process. Implementers will need to build their own user interface components to manage and control the assembly itself. However the design of assembly has anticipated that implementers can take advantage of an iterative set of steps from the providing of the content structure through to the definition of the context rules and validation rules that can assist the users and reduce the need for repetitive entry through the use of prompts and intelligent software cues that derive their information from the previous step in the definition of the complete assembly.

**Deleted:** 12/02/03

### 4.9.1 CAM Processor Notes

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

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

**Deleted:** 12/02/03

### 4.10.1 External Business Content Mapping

The business content mapping is an optional component to the base assembly functionality, and is primarily intended to bridge between the neutral assembly approach and specific domain implementations. The business content mapping script instructions are designed to provide non-procedural hints to implementation systems. Implementers can choose to use these to drive specific back-end application systems, or simply as documentation to constructing such application system linkages within their own systems. This can then provide useful hints to the assembly process itself or to implementations integrating between multiple application systems and requiring extended crosswalk information.

This initial release is a simple non-procedural system that allows specification of statements that can bridge between the assembly transaction and the business application. It is not intended to provide a complete full-function computation engine, but does provide the ability to simply equate between application content and structure content members with some ability to manipulate the content (it should be noted to that XPath statements contain some limited content manipulation functionality as well).

**Figure 4.10.1: Example of business content mapping script**

```
<ExternalMapping>
 <ContentAssociation>
  <Description>Product List</Description>
  <InputSource/>
  <OutputStore type=”SQL” location=”product_table”/>
  <RulesSet>
   <MapRule output="Products_List" input="@STARTGRP()"/>
   <MapRule output="type" input="Sales/Company/Year/Qtr/Product@type"/>
   <MapRule output="name"
             input="@trim(Sales/Company/Year/Qtr/Product/Item@name)"/>
   <MapRule output="manufacturer"
             input="Sales/Company/Year/Qtr/Product/Item@manufacturer"/>
   <MapRule output="value"
             input="Sales/Company/Year/Qtr/Product/Item@value"/>
   <MapRule output="sold"
             input="Sales/Company/Year/Qtr/Product/Item@sold"/>
   <MapRule output="Products_List" input="@ENDGRP()"/>
  </RulesSet>
 </ContentAssociation>
</ExternalMapping>
```

**Deleted:** 12/02/03

The syntax for this section is summarized in the following table shown in figure 4.10.2 and these predicates are designed as a simple set of sparse commands that augment the XPath statements to provide a core of content string based functionality.

**Figure 4.10.2: Summary of business content mapping script commands**

| Predicate | Parameter(s) | Description |
|---|---|---|
| `@concat(p1,p2)` | `[//memberpath \| string \| predicate()], [//memberpath \| string \| predicate()]` | Combine two strings together. Predicates can be combined to derive resultant string content. |
| `@trim(p1)` | `//memberpath` | Remove trailing and leading white space from content. |
| `@startgrp()` | `[//memberpath]` | Start of loop of recurring content. Optional memberpath reference denotes when 'next record' condition occurs on change of value / occurance in the input structure. |
| `@endgrp()` | `None` | End of loop of recurring content |
| `@multiply(p1,p2)` | `[//memberpath \| string \| predicate()], [//memberpath \| string \| predicate()]` | Compute result of calculation |
| `@divide(p1,p2)` | `[//memberpath \| string \| predicate()], [//memberpath \| string \| predicate()]` | Compute result of calculation |
| `@add(p1,p2)` | `[//memberpath \| string \| predicate()], [//memberpath \| string \| predicate()]` | Compute result of calculation |
| `@subtract(p1,p2)` | `[//memberpath \| string \| predicate()], [//memberpath \| string \| predicate()]` | Compute result of calculation |

**Deleted:** 12/02/03

| Predicate | Parameter(s) | Description |
|---|---|---|
| `@if(p1,p2,p3)` | `Expression, [//memberpath \| predicate()], [//memberpath \| predicate()]` | Logical expression, if the conditional expression is true, then p2, else p3. |
| `@upper(p1)` | `[//memberpath \| string \| predicate()]` | Change all characters to their uppercase equivalent. |
| `@lower(p1)` | `[//memberpath \| string \| predicate()]` | Change all characters to their lowercase equivalent. |
| `@len(p1)` | `[//memberpath \| string \| predicate()]` | Returns length of string item. |
| `@left(p1,p2)` | `[//memberpath \| string \| predicate()],[numeric \| //memberpath \| predicate()]` | Return p2 number of leftmost characters from a string p1. |
| `@right(p1,p2)` | `[//memberpath \| string \| predicate()],[numeric \| //memberpath \| predicate()]` | Return p2 number of rightmost characters from a string p1. |
| `@mid(p1,p2,p3)` | `[//memberpath \| string \| predicate()],[numeric \| //memberpath \| predicate()],[numeric \| //memberpath \| predicate()]` | Return p3 number of characters from a string p1 starting from position p2. |

The section completes the processing requirements for the assembly system; the addendum now provides reference examples.

**Deleted:** 12/02/03

### 4.11.1 Conformance Levels and Feature Sets

One goal of CAM is to provide the means for simple implementations. To facilitate this the implementation has been separated into three levels, where level 1 contains the minimum functionality, level 2 contains extended functionality and level 3 contains advanced features.

To aid implementers and conformance testing the following matrix shows by section those features that apply to each level. Also it should be noted that the CAM header section contains processing rules for header information relating to level control for CAM processor implementations.

**Figure 4.11.1: CAM conformance matrix.**

| Feature | Document reference | Level 1 | Level 2 | Level 3 |
|---|---|---|---|---|
| Header section processor | | required | required | required |
| Structure processor, simple XML | | required | required | required |
| Structure processor, inline predicates | | none | required | required |
| Structure processor for schemas | | none | none | required |
| Structure processor for non-XML targets | | none | none | required |
| Include sub-assembly mechanism | | none | required | required |
| XPath Context rules | | required | required | required |
| XPath lookup() function support | | none | required | required |
| External library functions | | none | required | required |
| Reference section – local definitions | | required | required | required |
| Reference section – external registry | | none | required | required |
| Validation section – simple checks | | none | required | required |
| Validation section – extended checks | | none | required | required |
| Validation section – external functions | | none | required | required |
| External Mapping section | | none | none | required |
| | | | | |

A CAM conformance test suite will be developed and made available from the website.

**Deleted:** 12/02/03

# A Addendum

The addendum contains some sample CAM XML instances, and the formal documented schema structure for CAM. These examples are provided both in the addendum and as standalone items as separate XML instance files[7].

## A1.1 Example of an Address assembly

The first example is a complete assembly bringing together the examples used in each section of this document. The focus is on the address details and the selection and control of the structure content given that address details are highly variant depending on the delivery country.

**Figure A.1.1: Sample CAM template of Address content with embedded context expressions**

```
<!-- Example Assembly for Address and Order items -->
<CAM xmlns:as="http://www.oasis-open.org/committees/cam">
 <AssemblyStructure >
   <Header>
     <CAMlevel value="2"/>
     <Description>WorldCup Soccer Order Transaction</Description>
     <Version>1.20</Version>
     <DateTime>02/12/2003</DateTime>
   </Header>
   <Declaration parameter='DeliveryCountry' default='USA' datatype='string'/>
   <Structure taxonomy='XML'>
   <Items CatalogueRef="2002">
     <SoccerGear>
       <Item as:makeRepeatable="true">
        <RefCode as:makeMandatory="true" as:setLength="10">%%</RefCode>
        <Description>%%</Description>
        <Style>WorldCupSoccer</Style>
        <UnitPrice as:setMask="'999.99','SQL'">%%</UnitPrice>
       </Item>
       <QuantityOrdered as:setMask="'999','SQL'">%%</QuantityOrdered>
       <SupplierID as:makeMandatory="true">%%</SupplierID>
       <DistributorID>%%</DistributorID>
```

[7] Implementers seeking the very latest details should reference the schema and DTD structure for CAM directly from the Internet location for developer's resources and not rely completely on the printed instance, since corrections and extensions to the printed formal published implementation reference documentation can lag behind. Participation in the online technical discussion groups is strongly recommended.

**Deleted:** 12/02/03

```
       <OrderDelivery>Normal</OrderDelivery>
       <DeliveryAddress as:choiceID="USA-Street">
          <FullName>%%</FullName>
          <Street>%%</Street>
          <City>%%</City>
          <State as:setLength="2" as:makeMandatory="true">%%</State>
       </DeliveryAddress>
       <DeliveryAddress as:choiceID="USA-APObox">
          <FullName>%%</FullName>
          <APOBox>%%</APOBox>
          <City>%%</City>
          <State as:setLength="2">%%</State>
          <Country>%%</Country>
       </DeliveryAddress>

       <DeliveryAddress as:choiceID="Canada">
          <PersonName>%%</PersonName>
          <Street1>%%</Street1>
          <Street2>%%</Street2>
          <TownCity>%%</TownCity>
          <PostCode>%%</PostCode>
          <Province>%%</Province>
          <Country>Canada</Country>
       </DeliveryAddress>
     </SoccerGear>
   </Items>
 </Structure>
</AssemblyStructure>

<BusinessUseContext>
   <Rules>
    <default>
      <context> <!-- default structure constraints -->
       <constraint action="makeRepeatable(//SoccerGear)" />
       <constraint action="makeMandatory(//SoccerGear/Items/*)" />
       <constraint action="makeOptional(//Description)" />
       <constraint action="makeMandatory(//Items@CatalogueRef)" />
       <constraint action="makeOptional(//DistributorID)" />
       <constraint action="makeOptional(//SoccerGear/DeliveryAddress)" />
      </context>
```

**Formatted:** French (France)

**Deleted:** 12/02/03

```
    </default>
    <context condition="token='//SoccerGear/SupplierID' and
                          contains(value,'SuperMaxSoccer')">
      <constraint action="makeMandatory(//SoccerGear/DeliveryAddress)"/>
    </context>
    <context condition="token='%DeliveryCountry%' and contains(value,'USA'">
      <constraint
         action="useChoiceByID(//SoccerGear/DeliveryAddress(#USA-Street))"/>
    </context>
    <context condition="token='%DeliveryCountry%' and contains(value,'APO'">
      <constraint
         action="useChoiceByID(//SoccerGear/DeliveryAddress(#USA-APObox))"/>
    </context>
    <context condition="token='%DeliveryCountry%'
                   and contains(value,'CANADA'">
      <constraint
         action="useChoiceByID(//SoccerGear/DeliveryAddress(#Canada))"/>
    </context>
   </Rules>
</BusinessUseContext>

<ContentReference>
   <Addressing>
    <registry name="SGIR" access="registry.sgir.org:1023" method="URL"
              description="Sporting Goods Industry Registry"/>
    <registry name="SGIRWSDL" access="registry.sgir.org:1025" method="WSDL"
              description="Sporting Goods Industry Registry"/>
    <registry name="UN" access="registry.un.org:9090" method="ebXML"
              description="United Nations EDIFACT Registry"/>
    <registry name="UPS" access="registry.ups.com:7001" method="URL"
              description="United Parcels Service Registry"/>
    <registry name="USPS" access="registry.usps.gov:8080" method="URL"
              description="United States Postal Service Registry"/>
    <registry name="Local" access="rdbms.mybusiness.com:4040" method="SQL"
              description="Local Product Database stored procedures"/>
   </Addressing>

   <item type="noun" name="RefCode"
         UIDReference="SGIR010027" taxonomy="UID" registry="SGIR"/>
   <item type="noun" name="Description"
```

**Deleted:** 12/02/03

```
        UIDReference="SGIR010050" taxonomy="UID" registry="SGIR"/>
  <item type="noun" name="Style"
        UIDReference="SGIR010028" taxonomy="UID" registry="SGIR"/>
  <item type="noun" name="SupplierID"
        UIDReference="SGIR010029" taxonomy="UID" registry="SGIR"/>
  <item type="noun" name="CatalogueRef" UIDReference="none" taxonomy="none"
        datatype="text" setlength="4" setmask="'\d\d\d\d',PERL" />
  <item type="noun" name="DistributorID" UIDReference="none" taxonomy="none"
        datatype="text" setlength="30" />
  <item type="noun" name="UnitPrice"
        UIDReference="070010" taxonomy="EDIFACT" registry="UN"/>
  <item type="noun" name="QuantityOrdered"
        UIDReference="070011" taxonomy="EDIFACT" registry="UN"/>
  <item type="noun" name="OrderDelivery"
        UIDReference="UPS050050" taxonomy="UID" registry="UPS"/>
  <item type="defaultAssembly" name="DeliveryAddress"
        UIDReference="USPS090081:01:05" taxonomy="UID" registry="USPS"/>
</ContentReference>
<DataValidations>
  <Conditions
      condition="token='%DeliveryCountry%' and contains(value,'USA'">
      <conditional
            expression="'//UnitPrice' and greater(value,'0.00')"
            syntax="XPath" outcome="fail"
            message="Item price not valid / missing" test="always"/>
      <conditional
            expression="'//RefCode + //UnitPrice' and
            lookup(value,'SGIRWSDL:unitprice_check')" outcome="report"
            message="Unit price value does not match catalog" test="always"/>
      <conditional
            expression="'//SupplierID' and
            lookup(value,'SGIRWSDL:supplierID_check')" outcome="fail"
            message="Unknown Supplier ID" test="always"/>
      <conditional
            expression="'//DistributorID' and
            lookup(value,'SGIRWSDL:distributor_check')" outcome="fail"
            message="Unknown distributor ID" test="postcheck"/>
      <conditional itemRef="//QuantityOrdered" conditioncheck="value"
            expression="'//QuantityOrdered' and
            lookup(value,'LocalSQL:quantityOnHand()')" outcome="report"
```

**Deleted:** 12/02/03

```
             message="Item not available / backordered" test="postcheck"/>
   </Conditions>
</DataValidations>
</AssemblyScript>
```

In this particular example the three different address formats, USA street address, USA APO box and Canadian address are selected depending on the business use context. Notice from the business perspective this effectively controls where physically the company will deliver its products.

See the main document for details on the techniques illustrated in each section of this example. The overall business capability demonstrated is the ability to use a single assembly to manage the content variants for the business process and to tie those to the context variables that determine the actual content structure for a given business scenario.

**Deleted:** 12/02/03

## A1.2 Example of an OAGIS BOD assembly

The next example is for the OAGIS BOD syntax (see http://www.openapplications.org) and shows how the base BOD mechanism expressed simply as a W3C XSD schema fails to cover the business need (see discussion in section 1 – Introduction), while the assembly for the BOD is able to provide the required business context rules and content linkage references completely.

**Figure A.1.2: Sample of a CAM template for OAGIS BOD content**

```
<CAM>
  <!—TBD -->
</CAM>
```

See the main document for details on the techniques illustrated in each section of this example. The overall business capability demonstrated is the ability to use a single assembly to manage the content variants for the business process and to tie those to the context variables that determine the actual content structure for a given business scenario.

**Formatted:** German (Germany)

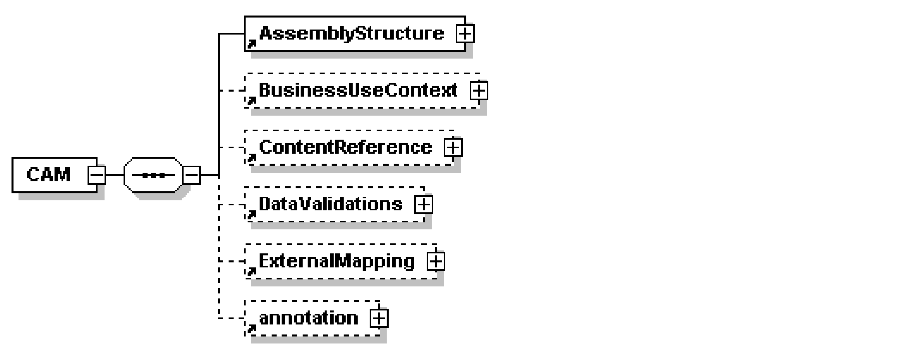
**Deleted:** 12/02/03

## A1.3 CAM schema (W3C XSD syntax)

This section is provided for implementers wishing a formal specification of the XML structure definition for the assembly itself. However specific implementation details not captured by the XSD syntax should be referenced by studying the specification details provided in this document and clarification of particular items can be obtained by participating in the appropriate on-line e-business developer community discussion areas and from further technical bulletins supplementing the base specifications.

### Figure A1.3: Schema of CAM.xsd

element **CAM**

| | |
|---|---|
| diagram |   CAM<br>AssemblyStructure<br>BusinessUseContext<br>ContentReference<br>DataValidations<br>ExternalMapping<br>annotation |
| children | **AssemblyStructure BusinessUseContext ContentReference DataValidations ExternalMapping annotation** |

source

```
<xs:element name="CAM">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="AssemblyStructure"/>
      <xs:element ref="BusinessUseContext" minOccurs="0"/>
      <xs:element ref="ContentReference" minOccurs="0"/>
      <xs:element ref="DataValidations" minOccurs="0"/>
      <xs:element ref="ExternalMapping" minOccurs="0"/>
      <xs:element ref="annotation" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

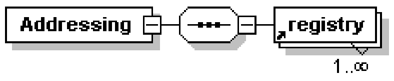
**Formatted:** French (France)

Elements
**Addressing**
**annotation**
**AssemblyStructure**
**BusinessUseContext**
**CAM**
**CAMlevel**
**conditional**
**Conditions**
**constraint**
**ContentAssociation**
**ContentReference**

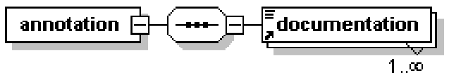
**Deleted:** 12/02/03

**context**
**DataValidations**
**DateTime**
**Declaration**
**default**
**Description**
**documentation**
**ExternalMapping**
**Header**
**InputSource**
**item**
**MapRule**
**OutputStore**
**Owner**
**registry**
**Rules**
**RulesSet**
**Structure**
**Version**

## element **Addressing**

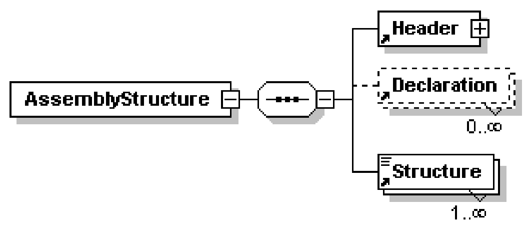
| | |
|---|---|
| diagram |  |
| children | **registry** |
| used by | element **ContentReference** |
| source | `<xs:element name="Addressing">`<br>  `<xs:complexType>`<br>    `<xs:sequence>`<br>      `<xs:element ref="registry" maxOccurs="unbounded"/>`<br>    `</xs:sequence>`<br>  `</xs:complexType>`<br>`</xs:element>` |

## element **annotation**

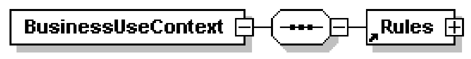
| | |
|---|---|
| diagram |  |
| children | **documentation** |
| used by | element **CAM** |
| source | `<xs:element name="annotation">`<br>  `<xs:complexType>`<br>    `<xs:sequence>`<br>      `<xs:element ref="documentation" maxOccurs="unbounded"/>`<br>    `</xs:sequence>`<br>  `</xs:complexType>`<br>`</xs:element>` |

## element **AssemblyStructure**

**Deleted:** 12/02/03

| | |
|---|---|
| diagram |   AssemblyStructure<br>Header<br>Declaration<br>0..∞<br>Structure<br>1..∞ |
| children | **Header Declaration Structure** |
| used by | element **CAM** |
| source | `<xs:element name="AssemblyStructure">`<br>`  <xs:complexType>`<br>`    <xs:sequence>`<br>`      <xs:element ref="Header"/>`<br>`      <xs:element ref="Declaration" minOccurs="0" maxOccurs="unbounded"/>`<br>`      <xs:element ref="Structure" maxOccurs="unbounded"/>`<br>`    </xs:sequence>`<br>`  </xs:complexType>`<br>`</xs:element>` |

element **BusinessUseContext**

| | |
|---|---|
| diagram |  |
| children | **Rules** |
| used by | element **CAM** |
| source | `<xs:element name="BusinessUseContext">`<br>`  <xs:complexType>`<br>`    <xs:sequence>`<br>`      <xs:element ref="Rules"/>`<br>`    </xs:sequence>`<br>`  </xs:complexType>`<br>`</xs:element>` |

element **CAMlevel**

| | |
|---|---|
| diagram | |
| used by | element **Header** |
| attributes | Name: value; Type: xs:NMTOKEN; Use: required; Default: ; Fixed: ; Annotation: |
| source | `<xs:element name="CAMlevel">`<br>`  <xs:complexType>`<br>`    <xs:attribute name="value" use="required">`<br>`      <xs:simpleType>`<br>`        <xs:restriction base="xs:NMTOKEN">`<br>`          <xs:enumeration value="1"/>`<br>`          <xs:enumeration value="2"/>`<br>`          <xs:enumeration value="3"/>`<br>`        </xs:restriction>` |

**Deleted:** 12/02/03

**Formatted:** French (France)

```
    </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>
```

element **conditional**

| | |
|---|---|
| diagram | |
| used by | element **Conditions** |

attributes

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| expression | xs:string | required | | | |
| syntax | xs:NMTOKEN | | | | |
| outcome | xs:NMTOKEN | required | | | |
| message | xs:string | | | | |
| test | xs:NMTOKEN | required | | | |

source

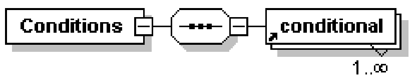
```
<xs:element name="conditional">
 <xs:complexType>
  <xs:attribute name="expression" type="xs:string" use="required"/>
  <xs:attribute name="syntax">
   <xs:simpleType>
    <xs:restriction base="xs:NMTOKEN">
     <xs:enumeration value="XPath"/>
     <xs:enumeration value="JavaScript"/>
     <xs:enumeration value="VB"/>
     <xs:enumeration value="Perl"/>
     <xs:enumeration value="Other"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="outcome" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:NMTOKEN">
     <xs:enumeration value="fail"/>
     <xs:enumeration value="ignore"/>
     <xs:enumeration value="report"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="message" type="xs:string"/>
  <xs:attribute name="test" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:NMTOKEN">
     <xs:enumeration value="always"/>
     <xs:enumeration value="postcheck"/>
     <xs:enumeration value="precheck"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>
```

**Formatted:** French (France)

**Formatted:** French (France)

**Formatted:** French (France)

element **Conditions**

| | |
|---|---|
| diagram |  |

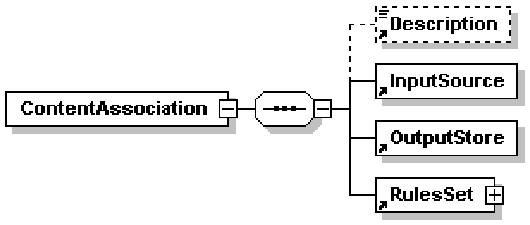
**Deleted:** 12/02/03

| | |
|---|---|
| children | **conditional** |
| used by | element **DataValidations** |
| attributes | Name Type Use Default Fixed Annotation<br>condition xs:string |
| source | `<xs:element name="Conditions">`<br>`  <xs:complexType>`<br>`    <xs:sequence>`<br>`      <xs:element ref="conditional" maxOccurs="unbounded"/>`<br>`    </xs:sequence>`<br>`    <xs:attribute name="condition" type="xs:string"/>`<br>`  </xs:complexType>`<br>`</xs:element>` |

## element constraint

| | |
|---|---|
| diagram | |
| used by | elements **context default** |
| attributes | Name Type Use Default Fixed Annotation<br>condition xs:string<br>action xs:string required |
| source | `<xs:element name="constraint">`<br>`  <xs:complexType>`<br>`    <xs:attribute name="condition" type="xs:string"/>`<br>`    <xs:attribute name="action" type="xs:string" use="required"/>`<br>`  </xs:complexType>`<br>`</xs:element>` |

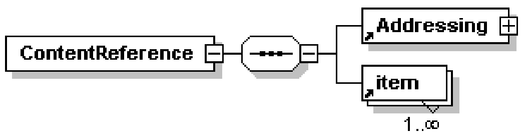
## element ContentAssociation

| | |
|---|---|
| diagram |  ContentAssociation<br>Description<br>InputSource<br>OutputStore<br>RulesSet |
| children | **Description InputSource OutputStore RulesSet** |
| used by | element **ExternalMapping** |
| source | `<xs:element name="ContentAssociation">`<br>`  <xs:complexType>`<br>`    <xs:sequence>`<br>`      <xs:element ref="Description" minOccurs="0"/>`<br>`      <xs:element ref="InputSource"/>`<br>`      <xs:element ref="OutputStore"/>`<br>`      <xs:element ref="RulesSet"/>`<br>`    </xs:sequence>`<br>`  </xs:complexType>`<br>`</xs:element>` |

**Formatted:** French (France)

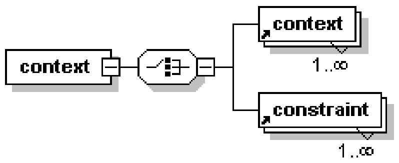
**Deleted:** 12/02/03

element **ContentReference**

| | |
|---|---|
| diagram |  |
| children | **Addressing item** |
| used by | element **CAM** |

source
```
<xs:element name="ContentReference">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Addressing"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

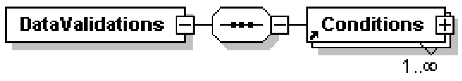
element **context**

| | |
|---|---|
| diagram |  |
| children | **context constraint** |
| used by | elements **context default Rules** |
| attributes | Name: condition; Type: xs:string; Use: required; Default: ; Fixed: ; Annotation: |

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| condition | xs:string | required | | | |

source
```
<xs:element name="context">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="context" maxOccurs="unbounded"/>
      <xs:element ref="constraint" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="condition" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

element **DataValidations**

| | |
|---|---|
| diagram |  |
| children | **Conditions** |
| used by | element **CAM** |

source
```
<xs:element name="DataValidations">
  <xs:complexType>
```

**Deleted:** 12/02/03

| | |
|---|---|
| | `<xs:sequence>` `<xs:element ref="Conditions" maxOccurs="unbounded"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>` |

element **DateTime**

| | |
|---|---|
| diagram | |
| type | **xs:string** |
| used by | element **Header** |
| source | `<xs:element name="DateTime" type="xs:string"/>` |

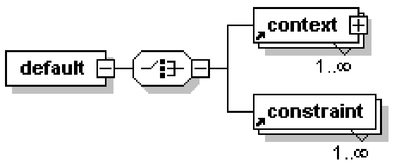
element **Declaration**

| | |
|---|---|
| diagram | |
| used by | element **AssemblyStructure** |
| attributes | see below |
| source | see below |

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| parameter | xs:string | required | | | |
| values | xs:string | | | | |
| default | xs:string | | | | |
| datatype | xs:string | | | | |

```
<xs:element name="Declaration">
  <xs:complexType>
    <xs:attribute name="parameter" type="xs:string" use="required"/>
    <xs:attribute name="values" type="xs:string"/>
    <xs:attribute name="default" type="xs:string"/>
    <xs:attribute name="datatype" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

element **default**

| | |
|---|---|
| diagram |  |
| children | **context constraint** |
| used by | element **Rules** |
| source | see below |

```
<xs:element name="default">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="context" maxOccurs="unbounded"/>
      <xs:element ref="constraint" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:complexType>
```

**Deleted:** 12/02/03

```
</xs:element>
```

element **Description**

| | |
|---|---|
| diagram | |
| type | **xs:string** |
| used by | elements **ContentAssociation Header** |

source

```
<xs:element name="Description" type="xs:string"/>
```

element **documentation**

| | |
|---|---|
| diagram | |
| type | extension of **xs:string** |
| used by | element **annotation** |

attributes

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| type | xs:NMTOKEN | required | | | |

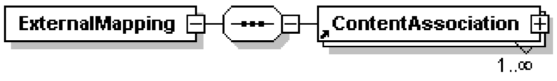
source

```
<xs:element name="documentation">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="xs:string">
    <xs:attribute name="type" use="required">
     <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
       <xs:enumeration value="description"/>
       <xs:enumeration value="note"/>
       <xs:enumeration value="license"/>
       <xs:enumeration value="usage"/>
       <xs:enumeration value="other"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

**Formatted:** French (France)

**Formatted:** French (France)

element **ExternalMapping**

| | |
|---|---|
| diagram |  |
| children | **ContentAssociation** |
| used by | element **CAM** |

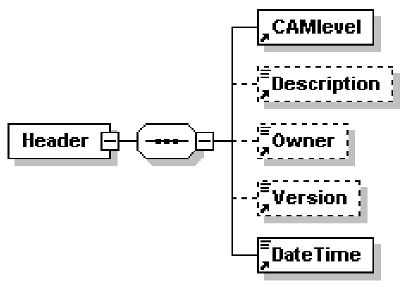
source

```
<xs:element name="ExternalMapping">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="ContentAssociation" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
```

**Deleted:** 12/02/03

| | |
|---|---|
| | `</xs:element>` |

element **Header**

| | |
|---|---|
| diagram |   Header<br>CAMlevel<br>Description<br>Owner<br>Version<br>DateTime |
| children | **CAMlevel Description Owner Version DateTime** |
| used by | element **AssemblyStructure** |
| source | see below |

```
<xs:element name="Header">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="CAMlevel"/>
      <xs:element ref="Description" minOccurs="0"/>
      <xs:element ref="Owner" minOccurs="0"/>
      <xs:element ref="Version" minOccurs="0"/>
      <xs:element ref="DateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **InputSource**

| | |
|---|---|
| diagram | |
| used by | element **ContentAssociation** |

attributes

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| type | xs:NMTOKEN | | | | |
| location | xs:string | | | | |

source

```
<xs:element name="InputSource">
  <xs:complexType>
    <xs:attribute name="type">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="SQL"/>
          <xs:enumeration value="XML"/>
          <xs:enumeration value="EDI"/>
          <xs:enumeration value="TXT"/>
          <xs:enumeration value="ODBC"/>
          <xs:enumeration value="OTHER"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="location" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

**Deleted:** 12/02/03

## element **item**

| | |
|---|---|
| diagram | |
| used by | element **ContentReference** |

attributes

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| type | xs:NMTOKEN | required | | | |
| name | xs:string | | | | |
| UIDReference | xs:string | required | | | |
| taxonomy | xs:string | required | | | |
| registry | xs:string | | | | |
| datatype | xs:string | | | | |
| setlength | xs:string | | | | |
| setmask | xs:string | | | | |

source

```
<xs:element name="item">
  <xs:complexType>
    <xs:attribute name="type" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="noun"/>
          <xs:enumeration value="corecomponent"/>
          <xs:enumeration value="BIE"/>
          <xs:enumeration value="BRIC"/>
          <xs:enumeration value="defaultAssembly"/>
          <xs:enumeration value="identifier"/>
          <xs:enumeration value="verb"/>
          <xs:enumeration value="schema"/>
          <xs:enumeration value="documentation"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="UIDReference" type="xs:string" use="required"/>
    <xs:attribute name="taxonomy" type="xs:string" use="required"/>
    <xs:attribute name="registry" type="xs:string"/>
    <xs:attribute name="datatype" type="xs:string"/>
    <xs:attribute name="setlength" type="xs:string"/>
    <xs:attribute name="setmask" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

**Formatted:** French (France)

## element **MapRule**

| | |
|---|---|
| diagram | |
| used by | element **RulesSet** |

attributes

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| output | xs:string | required | | | |
| input | xs:string | required | | | |

source

```
<xs:element name="MapRule">
  <xs:complexType>
    <xs:attribute name="output" type="xs:string" use="required"/>
    <xs:attribute name="input" type="xs:string" use="required"/>
```

**Deleted:** 12/02/03

```
  </xs:complexType>
</xs:element>
```

### element **OutputStore**

| | |
|---|---|
| diagram | |
| used by | element **ContentAssociation** |

attributes

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| type | xs:NMTOKEN | | | | |
| location | xs:string | | | | |

source

```
<xs:element name="OutputStore">
  <xs:complexType>
    <xs:attribute name="type">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="SQL"/>
          <xs:enumeration value="XML"/>
          <xs:enumeration value="EDI"/>
          <xs:enumeration value="TXT"/>
          <xs:enumeration value="ODBC"/>
          <xs:enumeration value="OTHER"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="location" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

### element **Owner**

| | |
|---|---|
| diagram | |
| type | **xs:string** |
| used by | element **Header** |
| source | `<xs:element name="Owner" type="xs:string"/>` |

### element **registry**

| | |
|---|---|
| diagram | |
| used by | element **Addressing** |

attributes

| Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|
| name | xs:string | required | | | |
| access | xs:string | required | | | |
| method | xs:NMTOKEN | required | | | |
| description | xs:string | | | | |

source

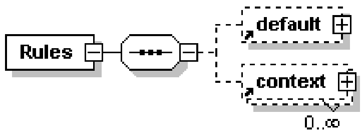
```
<xs:element name="registry">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required"/>
```

**Deleted:** 12/02/03

| | |
|---|---|
| | `<xs:attribute name="access" type="xs:string" use="required"/>`<br>`<xs:attribute name="method" use="required">`<br>`<xs:simpleType>`<br>`<xs:restriction base="xs:NMTOKEN">`<br>`<xs:enumeration value="URL"/>`<br>`<xs:enumeration value="http"/>`<br>`<xs:enumeration value="SOAP"/>`<br>`<xs:enumeration value="ebXML"/>`<br>`<xs:enumeration value="UDDI"/>`<br>`<xs:enumeration value="Other"/>`<br>`</xs:restriction>`<br>`</xs:simpleType>`<br>`</xs:attribute>`<br>`<xs:attribute name="description" type="xs:string"/>`<br>`</xs:complexType>`<br>`</xs:element>` |

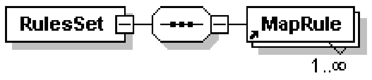
**Formatted:** French (France)

element **Rules**

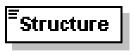
| | |
|---|---|
| diagram |  |
| children | **default context** |
| used by | element **BusinessUseContext** |
| source | `<xs:element name="Rules">`<br>`<xs:complexType>`<br>`<xs:sequence>`<br>`<xs:element ref="default" minOccurs="0"/>`<br>`<xs:element ref="context" minOccurs="0" maxOccurs="unbounded"/>`<br>`</xs:sequence>`<br>`</xs:complexType>`<br>`</xs:element>` |

**Formatted:** French (France)

element **RulesSet**

| | |
|---|---|
| diagram |  |
| children | **MapRule** |
| used by | element **ContentAssociation** |
| source | `<xs:element name="RulesSet">`<br>`<xs:complexType>`<br>`<xs:sequence>`<br>`<xs:element ref="MapRule" maxOccurs="unbounded"/>`<br>`</xs:sequence>`<br>`</xs:complexType>`<br>`</xs:element>` |

element **Structure**

| | |
|---|---|
| diagram |  |

**Deleted:** 12/02/03

| | |
|---|---|
| used by | element **AssemblyStructure** |

| attributes | Name | Type | Use | Default | Fixed | Annotation |
|---|---|---|---|---|---|---|
| | ID | xs:string | | | | |
| | reference | xs:string | | | | |
| | taxonomy | xs:NMTOKEN | required | | | |

source

```
<xs:element name="Structure">
  <xs:complexType mixed="true">
    <xs:attribute name="ID" type="xs:string"/>
    <xs:attribute name="reference" type="xs:string"/>
    <xs:attribute name="taxonomy" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="UID"/>
          <xs:enumeration value="XSD"/>
          <xs:enumeration value="DTD"/>
          <xs:enumeration value="RNG"/>
          <xs:enumeration value="XML"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

**Formatted:** French (France)

element **Version**

| | |
|---|---|
| diagram | |
| type | **xs:string** |
| used by | element **Header** |
| source | `<xs:element name="Version" type="xs:string"/>` |

## A1.4 Implementing RELAX NG with CAM

This section is focused on providing an open source implementation of CAM in combination with OASIS RELAX NG syntax support. The initial approach is aimed at providing a conformance level 1 implementation of CAM. The basic functionality will be obtained as much as possible by re-using existing RELAX NG tools. The strategy is to provide a pre-processor that can consume CAM XML syntax and then emit a valid RELAX schema that matches the structure and rules documented in the CAM syntax. For more current information on this sub-team activity, please visit the OASIS CAM TC website.

## A1.5 Business Process Mechanism (BPM) Context Support

This section describes the mechanism for providing context variables between the CAM processor and the remainder of the eBusiness architecture stack (see figure 2.7.2).

**Deleted:** 12/02/03

The CAM template provides the %parameter% mechanism to accept values from external processes. However the need is to provide a consistent mechanism in XML syntax for the propagation and specifying of context variables and their values throughout the components that make up the architecture stack.

Figure A1.5.1 shows a basic XML structure for carrying such values and it is anticipated that further development of this will continue with other OASIS TC groups to reach agreement on exact details of this mechanism.

Figure A1.5.1 XML structure for eBusiness context variable exchange.

```
<ebContext interchangeID='123456789' BPMref='ABC123456:01' CPAref='ABC012345'>
<variables>
   <variable name="Country"      value="USA"/>
   <variable name="itemType"     value="nonperishable"/>
   <variable name="partnerType"  value="wholesale"/>
</variables>
</ebContext>
```

**Deleted:** 12/02/03

## 5.0 References

- XML Path Language (XPath) specifications document, version 1.0, W3C Recommendation 16 November 1999, http://www.w3.org/TR/xpath/

- Extensible Markup Language (XML) specifications document, version 1.1, W3C Candidate Recommendation, 15 October 2002, http://www.w3.org/TR/xml11/

- XNL: Specifications and Description Document, OASIS CIQ TC, http://www.oasis-open.org/committees/ciq

- XAL: Specifications and Description Document, OASIS CIQ TC, http://www.oasis-open.org/committees/ciq

- ISO 16642 – Representing data categories http://www.loria.fr/projets/TMF/

**Deleted:** 12/02/03