

ASC X12

X12.7

Context Inspired Component Architecture (CICA)

Technical Specification AND XML Syntax Representation

ASC X12

- WORKING DRAFT -

ASC X12
- WORKING DRAFT -

TABLE OF CONTENTS

- [1 Purpose and Scope](#) 5
- [2 Reference related Standards](#) 5
- [3 Definitions and Concepts](#) 6
 - [3.1 Basic Structure](#) 6
 - [3.2 BNF Notation](#) 6
 - [3.2.1 Syntactic Entities](#) 6
 - [3.2.2 Defined Constructs](#) 7
 - [3.2.3 Other Inclusions](#) 7
 - [3.2.4 Alternative Definitions](#) 7
 - [3.2.5 Predefined Labels](#) 7
 - [3.2.6 Optional Items](#) 7
 - [3.2.7 Multiple Occurrences](#) 7
 - [3.2.8 Minimums/Maximums](#) 7
 - [3.2.9 Order of Resolution](#) 7
 - [3.3 Other Syntax Notation](#) 8
 - [3.4 Order of Precedence](#) 8
 - [3.5 Terminology](#) 8
- [4 Syntax Neutral CICA Representation](#) 9
 - [4.1 Primitives](#) 9
 - [4.1.1 Primitive Restrictive Properties](#) 11
 - [4.1.2 Superclass properties](#) 18
 - [4.1.3 Subclass properties](#) 19
 - [4.2 Components](#) 20
 - [4.2.1 Superclass properties](#) 21
 - [4.2.2 Subclass properties](#) 22
 - [4.3 Blocks](#) 23
 - [4.3.1 Superclass properties](#) 24
 - [4.3.2 Subclass properties](#) 25
 - [4.4 Assemblies](#) 26
 - [4.4.1 Superclass properties](#) 27
 - [4.4.2 Subclass properties](#) 28
 - [4.5 Templates](#) 29
 - [4.5.1 Superclass Properties](#) 30
 - [4.6 Modules](#) 31
 - [4.6.1 Module Properties](#) 31
 - [4.7 Documents](#) 32
 - [4.7.1 Subclass Properties](#) 33
 - [4.8 Common Symbols](#) 34
 - [4.8.1 requirements flag](#) 34
 - [4.8.2 min occurs](#) 34
 - [4.8.3 max occurs](#) 34
 - [4.8.4 content restriction flag](#) 34
 - [4.8.5 entity type flag](#) 35
 - [4.8.6 identification characteristic flag](#) 35
 - [4.8.7 abstract requirements flag](#) 36
 - [4.8.8 abstract repeatability flag](#) 36
 - [4.8.9 slot detail flag](#) 36
 - [4.8.10 context category](#) 36
 - [4.8.11 context category identification scheme name](#) 37
 - [4.8.12 context category identifier value](#) 37
 - [4.8.13 detail repeat](#) 38

ASC X12
WORKING DRAFT

4.9 [Symbols from Other Sources](#) 38

4.10 [Naming Conventions](#)..... 38

 4.10.1 [Item and Usage Names](#) 38

 4.10.2 [XML Names](#) 38

 4.10.3 [Generic versus Specific Usage Names](#) 39

5 [W3C XML Schema Language Schema CICA Representation](#) 40

 5.1 [Overall Structure](#)..... 40

 5.2 [Document](#)..... 41

 5.3 [Module](#)..... 43

 5.4 [Assembly](#)..... 43

 5.5 [Block](#)..... 44

 5.6 [Component](#)..... 45

 5.7 [Primitives](#)..... 46

 5.7.1 [Primitive Example](#) 48

 5.8 [Content Restriction](#)..... 49

 5.8.1 [Exclusive OR](#)..... 50

 5.8.2 [Inclusive OR](#)..... 51

ASC X12
- WORKING DRAFT -

X12.7 XML SYNTAX

1 Purpose and Scope

The Context Inspired Component Architecture (CICA) offers a method for building electronic business messages using XML. This document provides the guidelines for representing the CICA in (XSD) XML Schema as defined by the W3C XML Schema 1.0.

This document uses as its basis the philosophical foundation and general design principles forwarded in the published technical report, ASC X12 Reference Model for XML Design and its description of the Context Inspired Component Architecture (CICA).

2 Reference related Standards

This standard is used with the ASC X12 family of standards on electronic data interchange and associated standards by other bodies

Standard or Specification	Document Type	Version	Responsible Organization
Extensible Markup Language (XML)	W3C Recommendation	1.0 (Second Edition)	World Wide Web Consortium
XML Schema Part 1: Structures	W3C Recommendation	2 May 2001	World Wide Web Consortium
XML Schema Part 2: Datatypes	W3C Recommendation	2 May 2001	World Wide Web Consortium
ASC X12 Reference Model for XML Design	Technical Report Type II	October 2002	ANSI ASC X12
ebXML Core Components Technical Specification Part 8 of the ebXML Framework	UN/CEFACT Specification	15 November 2003 Version 2.01	United Nations Centre for Trade Facilitation and Electronic Business
RFC 2119 : Key words for use in RFCs to Indicate Requirement Levels	Request for Comment; Category Best Current Practice	March 1997	Internet Engineering Task Force of the Internet Society

Table 1 - Related Standards

3 Definitions and Concepts

3.1 Basic Structure

The Context Inspired Component Architecture presents seven levels of semantic granularity, from the entire document at the top, down to Primitives that contain one discrete piece of data. The CICA architecture is summarized in Table 2. The dividing line in the table below separates the context independent entities above from the context dependent entities below.

Layer	Definition
Primitive	One discrete piece of data within a Component
Component	Finest level of detail that indicates the identity or characteristics of business data to describe parties, resources, events, or locations in a document
Block	Specifies single parties, resources, events, or locations, composed of combinations of identity and characteristics Components
Assembly	Links sets of blocks into coherent collections of data that businesses can reuse as needed
Template	The framework of the document with slots or placeholders for Modules in the message
Module	Adds business context, such as special industry terminology or business process requirements or legal constraints, with the neutral data in the Assembly or Block into meaningful pieces of information to the business partners.
Document	Complete processable message containing data combined with the business context needed by business partners

Table 2 - Basic Structure

3.2 BNF Notation

This standard uses Backus-Naur Form (BNF) notation to clearly and unambiguously define CICA. BNF is described in this section, and used for the following two purposes in the standard:

- To express the logical composition of a construct that contains one or more other constructs
- To describe a document fragment that represents an instance of the construct in a data stream.

3.2.1 Syntactic Entities

Alphanumeric strings that are set in **bold** font denote syntactic entities, or nonterminal symbols. The underscore () character may be used in this representation to separate words. For example:

```

document
module

template
assembly
block

```

```
component
primitive
```

3.2.2 Defined Constructs

The defined construct is on the left side of a statement (a production) and is separated from the defining right side by two colons and an equal sign (:=). For example:

```
construct_one ::= construct_two
```

3.2.3 Other Inclusions

In the statements, spaces between syntactic entities are not significant and may be included for clarity. Meaningful spaces may be included in the statements by enclosing them in quotation marks. Other symbols may be included in quotation marks for clarity. The right side of the statement may be continued onto lower lines and be divided between syntactical entities. Ellipses (...) are used to represent missing items when a logical progression has been established.

3.2.4 Alternative Definitions

When alternative definitions exist, they may be shown separated by a vertical bar (|). This is the equivalent of a logical "OR". For example:

```
letter_or_digit ::= uppercase_letter | digit
```

3.2.5 Predefined Labels

Certain character strings such as predefined labels, literal markup, or punctuation are to be used as given. For clarity they are presented in **RED**. For example:

```
if_statement ::= if boolean_expression then
statement_sequence end if
```

3.2.6 Optional Items

Square brackets enclose optional items. For example:

```
if_statement ::= If boolean_expression then
statement_sequence [ else statement_sequence ] end if
```

3.2.7 Multiple Occurrences

Braces enclose an item which may appear zero or more times. For example:

```
unsigned_integer ::= digit { digit }
```

3.2.8 Minimums/Maximums

The minimum and maximum number of characters that may be used for a syntactic entity is specified by the inclusion of two numbers in parentheses appended to the end of the syntactic construct as (minimum/maximum). The following example is used to represent the X12 EDI data element in the data element dictionary.

```
id(02/03) ::= letter_or_digit letter_or_digit
[ letter_or_digit ]
```

3.2.9 Order of Resolution

When necessary parenthesis () shall be used to specific or clarify the order of resolution. For example:

```
id (02/03) ::= ( letter | digit ) ( letter | digit ) [ letter
| digit ]
```

3.3 Other Syntax Notation

When expressing the properties of an entity, as opposed to its grammatical composition in BNF, the n-tuple notation is used as follows:

```
entity = (property-1, property-2, ..., property-n)
```

Where a property may have values represented by an exclusive choice of two or more entities, it may be represented within the n-tuple by another set of parentheses enclosing the choices, each of which is separated by the vertical bar (|).

```
entity = (property-1, (property-2a | property-2b), ..., property-n)
```

In addition, conventional set notation may be used when it is appropriate in one context to refer to the set as a whole, yet in another context to enumerate the members of the set.

```
set = {element-1, element-2, ..., element-n}
```

NOTE

There is only one set of symbols used in this document. When the same symbol is used in different notations, it represents the same entity. For example, when **module_XML_name** appears in n-tuple property list, it is the same entity as when it is used in the BNF production for the schema fragment.

3.4 Order of Precedence

If there is any conflict between the notational representation of an entity or a concept and the text description, the notational representation takes precedence.

3.5 Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119] as quoted here:

NOTE

The force of these words is modified by the requirement level of the document in which they are used.

1. MUST

This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

2. MUST NOT

This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.

3. SHOULD

This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

4. SHOULD NOT

This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

5. MAY

This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option **MUST** be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option **MUST** be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

3.6 Pre-defined Entities

Some entities expressed in BNF in this document derive from entities defined in other documents referenced in this document. In this document, these entities are prefixed with identifiers to make it clear that these entities are defined outside this document. The prefixes and the documents they reference are as follows:

<u>Prefix</u>	<u>Agency</u>	<u>Document</u>
XML_	W3	Extensible Markup Language (XML)
XSD_	W3	XML Schema
CCT_	UN/CEFACT	ebXML Core Components Technical Specification

4 Syntax Neutral CICA Representation

The syntax neutral definitions represent the various syntactic entities of the CICA architecture as they relate to one another.

CICA entities are either context-neutral or context-dependent. The context neutral entities are Primitive, Component, Block, Assembly, and Template. The context-dependent entities are Module and Document. Each of the context-neutral entities has a set of properties that are constant regardless of its usage. In addition, all of these except Template has an additional set of properties that are only relevant when the entity is used within a Module. The usage within the Module may either be as an explicit member of the Module or as a member of one of the Module's members (and so on, recursively down to entities that have no members).

The set of properties defined in an abstract superclass entity are called the constructive details. The constructive details in a concrete subclass are inherited from its abstract superclass parent. For clarity in this document, these inherited constructive details are not shown explicitly in the concrete subclass list of properties. The inherited constructive details can not be modified in a concrete subclass entity. The set of additional properties defined in a concrete subclass entity are called the restrictive details. Different concrete subclasses of a single abstract superclass will differ in restrictive details.

4.1 Primitives

Primitives are the lowest CICA entity and are based on a single CCT (reference section 4.9). Each Primitive has one content component and zero or more supplementary components. Each content and supplementary component may have a number of restrictions placed with the Primitive subclass. The allowable list of restrictions, and the content of each CCT is listed in section 4.1.1.

```
Primitive Abstract Superclass:
primitive = (primitive_name,
            primitive_UID,
            primitive_XML_name,
            primitive_description,
            CCT_cct,
            CCT_member_list)
CCT_member_list ::= content_component
                  supplementary_component{supplementary_component}
content_component ::= CCT_member
supplementary_component ::= CCT_member
CCT_member = (CCT_member_name,
             CCT_member_XML_name,
             CCT_cct,
             CCT_datatype)
Primitive Concrete Subclass:
primitive_subclass = (primitive_subclass_name,
                    primitive_subclass_UID,
                    primitive_subclass_XML_name,
                    primitive_description,
                    CCT_cct,
                    subclass_CCT_member_list)
subclass_CCT_member_list ::= subclass_content_component
                          subclass_supplementary_component{subclass_supplementary_component}
subclass_content_component ::= subclass_CCT_member
subclass_supplementary_component ::= subclass_CCT_member
subclass_CCT_member = (CCT_member_name,
                     CCT_member_XML_name,
                     CCT_member_type_XML_name,
```

```

CCT_cct,
CCT_requirements_flag,
CCT_datatype,
[ {CCT_datatype_restrictions} ])

CCT_datatype_restrictions = {
  CCT_pattern,
  CCT_length,
  CCT_min_length,
  CCT_max_length,
  CCT_min_inclusive,
  CCT_max_inclusive,
  CCT_min_exclusive,
  CCT_max_exclusive,
  CCT_total_digits,
  CCT_fraction_digits}

```

NOTE: Not every data type restriction is permitted for every date type. The restrictions permitted for each data type are detailed in section 4.1.1.

```

[enumerations]

enumerations ::= enumeration_name enumeration {enumeration}
enumeration ::= enumeration_value enumeration_description

```

4.1.1 Primitive Restrictive Properties

Each content component and supplementary component has a specific generic CICA data type. In addition, the value space of each of these may be further restricted by a set of facets. This section describes the generic data types and the facets that may be applied to them. In some cases, the facet may redefine the data type to one whose value space is a subset of the base data type. For example, decimal may be further restricted by assigning a data type of integer.

4.1.1.1 CICA Data Types

The generic, syntax independent data types in CICA, their descriptions, and the value spaces associated with them are listed below.

1. string

A string is a sequence of zero or more characters. The value space of string is the set of finite length sequences of zero or more characters. The definition of character, its value space, and encoding is dependent upon the implementation syntax. For example, in X12 EDI syntax the string data type and character set are as defined in X12.6. In an XML

syntax representation of CICA, a string may consist of zero or more UNICODE characters as specified in the XML Recommendation.

2. **token**

A token is a string of one or more characters, restricted such that there may be no leading, trailing, or embedded spaces or other whitespace characters, where whitespace characters are defined by the specific implementation syntax. (NOTE: This is very similar to the schema language token datatype, except that the CICA token is a single token and a schema language token may be a set of tokens.)

3. **boolean**

A boolean indicates either true or false.

The value space of the boolean data type is as defined for the schema language boolean datatype: “*the value space required to support the mathematical concept of binary-valued logic: {true, false}.*”

4. **integer**

An integer is a whole base ten number with no fractional part and no decimal point. The infinite set { ..., -2, -1, 0, 1, 2, ... }

5. **decimal number**

A decimal number is a base ten number that may have either or both of an integer part and a fractional part. In syntax independent terms, the value space of a decimal number is as defined for the schema language decimal datatype: “set of the values $i \times 10^{-n}$, where i and n are integers such that $n \geq 0$.” The value space of decimal number may be constrained by data type limitations of the implementation syntax.

6. **date**

A date is a calendar date corresponding to an entire twenty-four hour period. The value space of date is as defined for the schema language date datatype: “the set of Gregorian calendar dates as defined in section 5.2.1 of ISO 8601.”

7. **time**

A time is an instant of time that recurs every day. The value space of time is as defined for the schema language time datatype: “the space of time of day values as defined in section 5.3 of ISO 8601.” Note that implementation syntaxes may or may not include a time zone indication in the time data type.

8. **datetime**

A datetime is a specific instance of time. The value space of time is as defined for the schema language dateTime datatype: “the space of Combinations of date and time of day values as defined in section 5.4 of ISO 8601.”

4.1.1.2 CICA Restrictive Facets

A facet is an attribute of an atomic data item (i.e., a content or supplementary component of a Core Component Type), that constrains the value space of the item beyond the constraints of the item’s assigned data type. The facets are listed below with their definitions and data types.

1. **pattern**

The pattern facet is a string of characters, expressed as a regular expression, that limits the value space of the item such that it must conform to a specific pattern. A pattern must be a string and expressed in CICA as a regular expression as specified in Appendix F of Part 2 of the W3C XML Schema Recommendation. Implementation syntaxes may require that patterns be specified in different formats.
2. **length**

The length facet limits the length of the data item to a specific length, where the units of length is dependent on the data type. The value of a length facet must be an integer value greater than or equal to zero.
3. **min_length**

The min_length facet limits the length of the data item to a specific minimum length, where the units of length is dependent on the data type. The value of a min_length facet must be an integer value greater than or equal to zero.
4. **max_length**

The max_length facet limits the length of the data item to a specific maximum length, where the units of length is dependent on the data type. The value of a max_length facet must be an integer value greater than or equal to zero.
5. **min_inclusive**

The min_inclusive facet limits allowable range of values a data item by specifying a minimum value that is included in the range. The data type of min_inclusive is the same as the data type of the item to which it is applied.
6. **max_inclusive**

The max_inclusive facet limits allowable range of values a data item by specifying a maximum value that is included in the range. The data type of max_inclusive is the same as the data type of the item to which it is applied.
7. **min_exclusive**

The min_exclusive facet limits allowable range of values a data item by specifying a minimum value that is not included in the range. The data type of min_exclusive is the same as the data type of the item to which it is applied.
8. **max_exclusive**

The max_exclusive facet limits allowable range of values a data item by specifying a maximum value that is not included in the range. The data type of max_exclusive is the same as the data type of the item to which it is applied.
9. **fraction_digits**

The fraction digits facet limits the maximum number of digits to the right of the decimal point of a decimal number. The value of a fraction_digits facet must be an integer value greater than or equal to zero.

10. enumeration

The enumeration facet limits the value space of a data item by limiting it to a specific, finite set of values. Each member of the set is a two-tuple with **enumeration_value** specifying the allowable value and **enumeration_description** containing the freeform text description of the value. The **enumeration_value** must be of the same data type as the item to which the facet is applied.

enumeration_name ::= XML_string

enumeration_description ::= XML_string

enumeration_value ::= XSD_enumerationValue

4.1.1.3 Supported Facets for each Data Type

This subsection specifies the CICA restrictive facets allowed for each CICA data type.

1. string

- a. length
- b. min_length
- c. max_length
- d. pattern
- e. enumeration

2. token

- a. length
- b. min_length
- c. max_length
- d. pattern
- e. enumeration

3. boolean

none

4. integer

- a. total_digits
- b. fraction_digits
- c. pattern
- d. enumeration
- e. max_inclusive
- f. max_exclusive
- g. min_inclusive
- h. min_exclusive

5. decimal number

- a. total_digits
- b. fraction_digits
- c. pattern
- d. enumeration
- e. max_inclusive
- f. max_exclusive
- g. min_inclusive

- h. min_exclusive
- 6. date
 - a. pattern
 - b. enumeration
 - c. max_inclusive
 - d. max_exclusive
 - e. min_inclusive
 - f. min_exclusive
- 7. time
 - a. pattern
 - b. enumeration
 - c. max_inclusive
 - d. max_exclusive
 - e. min_inclusive
 - f. min_exclusive
- 8. datetime
 - a. pattern
 - b. enumeration
 - c. max_inclusive
 - d. max_exclusive
 - e. min_inclusive
 - f. min_exclusive

length ::= XSD:positiveInteger
min_length ::= XSD:positiveInteger
max_length ::= XSD:positiveInteger
pattern ::= XSD:pattern
enumeration ::= XSD:enumeration
total_digits ::= XSD:positiveInteger
fraction_digits ::= XSD:nonNegativeInteger
max_inclusive ::= XSD:maxInclusive
max_exclusive ::= XSD:maxExclusive
min_inclusive ::= XSD:minInclusive
min_exclusive ::= XSD:minExclusive

4.1.1.4 CICA Data Types for CCT Content and Supplementary Components

The Core Component Types upon which CICA Primitives are based are listed below with the CICA data types of the content and supplementary components. The value spaces of each of these content and supplementary components are constrained as specified in the Core Component Technical Specification. They may be further constrained by the facets appropriate to each data type as listed in the previous subsections.

1. CCT Amount

- a. Content Component – **decimal**
 - **fraction_digits**
 - **max_inclusive**
 - **min_inclusive**
 - **min_exclusive**
 - **max_exclusive**
 - **total_digits**
 - b. Currency Identifier – **string**
 - **enumeration**
 - c. Currency Code List Version Identifier – **string**
2. CCT BinObj
 - a. Content Component – **string**
 - **min_length**
 - **max_length**
 - b. Format Text – **string**
 - **enumeration**
 - c. MimeCode – **string**
 - **enumeration**
 - d. CharacterSet – **string**
 - **enumeration**
 - e. EncodingCode – **string**
 - **enumeration**
 - f. URI – **string**
 - **enumeration**
 - g. FileNameText – **string**
 - **enumeration**
3. CCT Code
 - a. Content Component – **token**
 - **min_length**
 - **max_length**
 - **pattern**
 - **enumeration**
 - b. ListAgencyIdentifier – **string**
 - **enumeration**
 - c. ListAgencyNameText – **string**
 - **enumeration**
 - d. ListAgencyName – **string**
 - **enumeration**
 - e. ListIdentifier – **string**
 - **enumeration**
 - f. ListSchemeURI – **string**
 - **enumeration**
 - g. ListURI – **string**
 - **enumeration**
 - h. ListVersionIdentifier – **string**
 - **enumeration**
 - i. NameText – **string**
 - **enumeration**
4. CCT DateTime
 - a. Content Component – **datetime**

- **max_inclusive**
 - **min_inclusive**
 - **min_exclusive**
 - **max_exclusive**
 - **pattern**
- b. FormatText – **string**
- **enumeration**
5. CCT Identifier
- a. Content Component – **token**
- **min_length**
 - **max_length**
 - **pattern**
- b. IdSchemeAgency – **string**
- **enumeration**
- c. IdSchemaAgencyNameText – **string**
- **enumeration**
- d. IdSchemelIdentifier – **string**
- **enumeration**
- e. IdSchemeNameText – **string**
- **enumeration**
- f. IdSchemeURI – **string**
- **enumeration**
- g. IdSchemeVersionIdentifier – **string**
- **enumeration**
6. CCT Indicator
- a. Content Component – **token**
- **pattern**
 - **enumeration**
- max of 2 values are allowed
- b. FormatText – **string**
- **enumeration**
- fixed {"numeric", "textual", "binary"}
7. CCT Measure
- a. Content Component – **decimal**
- **total_digits**
 - **fraction_digits**
 - **min_exclusive**
 - **max_exclusive**
 - **min_inclusive**
 - **max_inclusive**
 - **enumeration**
 - **pattern**
- b. UnitCode – **token**
- **enumeration**
- c. UnitCodeListVersion – **string**
- **enumeration**
8. CCT Numeric
- a. Content Component – **decimal**
- **Integer**

Note: a datatype of integer may be chosen to further restrict the content component.

- **total_digits**
 - **fraction_digits**
 - **min_inclusive**
 - **max_inclusive**
 - **min_exclusive**
 - **max_exclusive**
 - **pattern**
- b. FormatText – **token**
- **enumeration**
 - 1 of 3 values (“integer”, “decimal”, “percentage”)
- Note
“Real Number” has been omitted because the content component is defined as decimal
9. CCT Quantity
- a. Content Component – **decimal**
- **total_digits**
 - **fraction_digits**
 - **min_inclusive**
 - **max_inclusive**
 - **min_exclusive**
 - **max_exclusive**
 - **min_length**
 - **max_length**
 - **pattern**
- b. UnitCode – **token**
- **enumeration**
- c. UnitCodeListID – **string**
- **enumeration**
- d. UnitCodeListAgencyID – **string**
- **enumeration**
- e. UnitCodeListAgencyNameText – **string**
- **enumeration**
10. CCT Text
- a. Content Component – **string**
- **token**
 - Note: token may be chosen to further restrict the base string datatype for the content component.
 - **min_length**
 - **max_length**
 - **pattern**
 - **enumeration**

4.1.2 Superclass properties

4.1.2.1 Syntax Independent

- **primitive_name** - The name of the Primitive abstract superclass. MUST conform to naming conventions defined in 4.10.
- **primitive_UID** – The unique identifier assigned to this Primitive.
- **primitive_description** - A description of the content of the Primitive.
- **content_component** - this carries the actual value of a Core Component, as defined in the Core Components Technical Specification.

- supplementary_component - A supplementary component provides additional semantics to qualify the semantics of the content component, as defined in the Core Components Technical Specification.
- CCT_member_name – The name of the content or supplementary component of the Primitive. These are content and supplementary component names specified in the CCT.
- CCT - Core Component Type (reference section 4.8.13)
- datatype - One of the CICA data types defined in section 4.1.1.1.

4.1.2.2 XML Syntax Dependent

- primitive_XML_name - The name of the complexType that represents the Primitive abstract superclass, based on the Block name, removing characters to conform to the rules of nmtoken.

primitive_XML_name ::= XML_nmtoken

- For each supplementary component
CCT_member_XML_name - the name of the supplementary component as used with the Primitive.

CCT_member_XML_name ::= XML_nmtoken

4.1.3 Subclass properties

4.1.3.1 Syntax Independent

- primitive_subclass_name - The name of the Primitive subclass. The name SHALL be formed by appending a two digit sequence number to the superclass primitive_name.
- primitive_subclass_UID – the unique identifier assigned to this Primitive subclass.
- CCT_member_name – The name of the content or supplementary component of the Primitive. These are content and supplementary component names specified in the CCT.
- CCT - Core Component Type (reference section 4.8.13)
- datatype - One of the CICA data types defined in section 4.1.1.1.
- requirements_flag - As defined in section 4.8.1. Indicates the requirement restriction imposed on the member of the Block member constraint, that is, the requirements on the presence of the member within the Component_subclass.

4.1.3.2 XML Syntax Dependent

- primitive_subclass_XML_name - The name of the complexType that represents the Primitive concrete subclass. The name is formed by appending a two digit sequence number to the name of the parent abstract superclass Primitive.

**primitive_subclass_XML_name ::= primitive_XML_name [0,1,2,...,9]
[0,1,2,...,9]**

- CCT_member_XML_name - The XML name of the supplementary component

CCT_member_XML_name ::=
supplementary_component_XML_name

- CCT_member_type_XML_name – The XML name of the simpleType assigned to the content component XML element or supplementary component XML attribute
- **CCT_member_type_XML_name ::=**
content_component_type_XML_name |
supplementary_component_type_XML_name
- content_component_type_XML_name - The XML name for the schema type representing the content component.

For each content component restriction

- datatype_restriction - restrictions allowed on the Primitive's content component as defined in section 4.1.1
- For each supplementary component
- supplementary_component_XML_name - the name of the supplementary component as used with the Primitive
supplementary_component_XML_name ::= XML_nmtoken
- supplementary_component_type_XML_name - the XML name for the schema language type representing the supplementary component.

supplementary_component_type_XML_name ::=
primitive_XML_name_name_of_supplementary_component

primitive_XML_name_name_of_supplementary_component ::=
XML_nmtoken

- For each restriction
 - datatype_restriction - restrictions allowed on the Primitive's supplementary component as defined in section 4.1.1

4.2 Components

Components are composed of two or more Primitives.

Abstract Superclass:

component ::= primitive primitive [{primitive}]

component = (component_name,

component_UID,

component_description,

component_XML_name,

component_member_list)

```
component_member_list ::= component_member component_member
{component_member}
```

```
component_member = (primitive_name,
primitive_usage_name,
primitive_usage_XML_name)
```

Component Concrete Subclass:

```
component_subclass = ( component_subclass_name,
component_subclass_UID,
component_subclass_XML_name,
component_member_constraint_list)
```

```
component_member_constraint is a subclass of component_member
component_member_constraint_list ::= component_member_constraint
{component_member_constraint}
```

```
component_member_constraint = (primitive_subclass_name,
component_member_constraint_usage_name,
component_member_constraint_usage_XML_name,
max_occurs,
min_occurs,
requirements_flag,
content_restriction_flag)
```

4.2.1 Superclass properties

4.2.1.1 Syntax Independent

- **component_name** - The name of the Component abstract superclass. MUST conform to naming conventions defined in section 4.10.
- **component_name ::= XML_string**
- **component_UID** - The unique identifier assigned to the Component.
- **component_description** - A description of the content of the Component.
- For each member of the Component
 - **primitive_usage_name** - Is the name of the member as used within the Component. MUST conform to naming conventions specified in section 4.10.

primitive_usage_name ::= XML_string

- **primitive_name** - Is as defined in section 4.1 on Primitive

4.2.1.2 XML Syntax Dependent

- `component_XML_name` - The name of the complexType that represents the Component abstract superclass, based on the Block name, removing characters to conform to the rules of nmtoken.

`component_XML_name ::= XML_nmtoken`

- For each member of the Component
 - `primitive_usage_XML_name` - The name of the element representing the member Primitive as used within the Component.

`primitive_usage_XML_name ::= XML_nmtoken`

4.2.2 Subclass properties**4.2.2.1 Syntax Independent**

- `component_subclass_name` - The name of the Component subclass. The name SHALL be formed by appending a two digit sequence number to the superclass `component_name`.

**`component_subclass_name ::= component_name [0,1,2,...,9]
[0,1,2,...,9]`**

- `component_subclass_UID` - The unique identifier assigned to the Component subclass.
- For each member of the Component
 - `primitive_subclass_name` - Is as defined in section 4.1 on Primitive. MUST be the name of a Primitive subclass that is derived from the abstract superclass Primitive defined in the Primitive abstract superclass member.
 - `component_member_constraint_usage_name` – the name of the usage of the Primitive in the Component. Defaults to the `primitive_usage_name` of the member in the abstract superclass Component but may be overridden.
 - `requirements_flag` - As defined in section 4.8.1. Indicates the requirement restriction imposed on the member of the Block member constraint, that is, the requirements on the presence of the member within the Component_subclass.
 - `min_occurs` - As defined in section 4.8.2. Indicates the minimum number of occurrences permitted for the Block constraint member, that is, the member within the Component_subclass.
 - `max_occurs` - As defined in section 4.8.3. Indicates the maximum number of occurrences permitted for the Component constraint member, that is, the member within the Component_subclass.
 - `content_restriction_flag` - As defined in section 4.8.4. Indicates the content restriction applying to a subset of Primitive members within the Component_subclass.

4.2.2.2 XML Syntax Dependent

- `component_subclass_XML_name` - The name of the complexType that represents the Component concrete subclass. The name is formed by appending a two digit sequence number to the name of the parent abstract superclass Component.

component_subclass_XML_name ::= component_XML_name
[0,1,2,...,9] [0,1,2,...,9]

- For each member of the Component
 - component_member_constraint_usage_XML_name – XML name of the usage of the Primitive that is derived from the component_member_constraint_usage_name according to section 4.10
 - primitive_subclass_XML_name - The name of the complexType that represents the Primitive concrete subclass, as defined in section 4.1.3.

4.3 Blocks

Blocks are composed of one or more identity Primitives or Components and may also contain one or more characteristic Primitives or Components.

An identity Primitive or identity Component provides sufficient information to distinguish a party, location, event, or resource from other party, location, event, or resource. For example, a UPC code uniquely distinguishes one grocery product from another.

A characteristic Primitive or characteristic Component provides descriptive information such as physical or demographic details. For example, color, height, weight, and aroma.

Abstract Superclass:

```
block ::= identity {identity | characteristic}
```

```
identity ::= component | primitive
```

```
characteristic ::= component | primitive
```

```
block = (block_name,
        block_UID,
        block_description,
        block_XML_name,
        block_type_flag,
        block_member_list)
```

```
block_type_flag ::= entity_type_flag
```

```
block_member_list ::= block_member {block_member}
```

```
block_member = (component_name | primitive_name),
               identification_characteristic_flag,
               block_member_usage_name, block_member_usage_XML_name )
```

Block Concrete Subclass:

```

block_subclass = ( block_subclass_name ,
                  block_subclass_UID ,
                  block_subclass_XML_name ,
                  block_member_constraint_list )

block_member_constraint_list ::= block_member_constraint
                               { block_member_constraint }

block_member_constraint is a subclass of block_member
block_member_constraint =
  (( component_subclass_name | primitive_subclass_name ) ,
   block_member_constraint_usage_name ,
   block_member_constraint_usage_XML_name ,
   max_occurs ,
   min_occurs ,
   requirements_flag ,
   content_restriction_flag )

```

4.3.1 Superclass properties

4.3.1.1 Syntax Independent properties

- block_name - The name of the Block abstract superclass. MUST conform to naming conventions defined in 4.10.
block_name ::= XML_string
- block_UID - The unique identifier assigned to the Block.
- block_description - A description of the content of the Block.
- block_type_flag - As defined in section 4.8.5 on entity_type_flag. Indicates the nature of the object described by the Block content.
- For each member of the Block
 - block_member_usage_name - Is the name of the member as used within the Block. MUST conform to naming conventions specified in section 4.10.
 - identification_characteristic_flag - As defined in section 4.8.6. Indicates whether the Block member is an identity or characteristic.
 - component_name - Is as defined in section 4.2 on Component.
 - primitive_name - Is as defined in 4.1 on Primitive

4.3.1.2 XML syntax dependent properties

- block_XML_name - The name of the complexType that represents the Block abstract superclass, based on the Block name, removing characters to conform to the rules of nmtoken.
block_XML_name ::= XML_nmtoken
- block_subclass_UID - The unique identifier assigned to the Block subclass.
- For each member of the Block

- `block_member_usage_XML_name` - The name of the element representing the member Primitive or Component as used within the Block.

`block_member_usage_XML_name ::= XML_nmtoken`

- `block_member_constraint_usage_XML_name` – the name of the usage of the member in the Block. Defaults to the usage name of the member of the abstract superclass Block but may be overridden.

`block_member_constraint_usage_XML_name ::= XML_nmtoken`

4.3.2 Subclass properties

4.3.2.1 Syntax Independent properties

- `block_subclass_name` - The name of the Block subclass. The name SHALL be formed by appending a two digit sequence number to the superclass `Block_name`.

`block_subclass_name ::= block_name [0,1,2,...,9] [0,1,2,...,9]`

- For each member of the Block
 - `block_member_constraint_usage_XML_name` – XML name of the usage of the member of the Block that is derived from the `block_member_constraint_usage_name` according to section 4.10.
 - `component_subclass_name` - Is as defined in section 4.2 on Component. MUST be the name of a Component subclass that is derived from the abstract superclass Component defined in the Block abstract superclass member.
 - `primitive_subclass_name` - Is as defined in 4.1 on Primitive. MUST be the name of a Primitive subclass that is derived from the abstract superclass Primitive defined in the Primitive abstract superclass member.
 - `requirements_flag` - As defined in section 4.8.1. Indicates the requirement restriction imposed on the member of the Block member constraint, that is, the requirements on the presence of the member within the `Block_subclass`.
 - `min_occurs` - As defined in section 4.8.2. Indicates the minimum number of occurrences permitted for the Block constraint member, that is, the member within the `Block_subclass`.
 - `max_occurs` - As defined in section 4.8.3. Indicates the maximum number of occurrences permitted for the Block constraint member, that is, the member within the `Block_subclass`.
 - `content_restriction_flag` - As defined in section 4.8.4. Indicates the content restriction applying to a subset of Component or Primitive members within the `Block_subclass`.

4.3.2.2 XML syntax dependent properties

- `block_subclass_XML_name` - The name of the complexType that represents the Block concrete subclass. The name is formed by appending a two digit sequence number to the name of the parent abstract superclass Block.

**block_subclass_XML_name ::= block_XML_name [0,1,2,...,9]
[0,1,2,...,9]**

- For each member of the Block
 - block_member_usage_XML_name - The name of the element representing the member Primitive or Component as used within the Block.

4.4 Assemblies

Assemblies are used for grouping semantically equivalent syntactic entities. Assemblies are composed of two or more Blocks and/or assemblies.

Abstract Superclass:

```
assembly ::= (assembly | block ) (assembly | block ) {(assembly |
block)}
```

```
assembly = (assembly_name ,
```

```
assembly_UID ,
```

```
assembly_type_flag ,
```

```
assembly_XML_name ,
```

```
assembly_description ,
```

```
assembly_member_list)
```

```
assembly_member_list ::= assembly_member
```

```
assembly_member
```

```
{assembly_member}
```

```
assembly_member = ( (block_name | member_assembly_name ,
```

```
assembly_member_usage_name ,
```

```
assembly_member_usage_XML_name)
```

Assembly Concrete Subclass:

```
assembly_subclass = ( assembly_subclass_name ,
```

```
assembly_subclass_UID ,
```

```
assembly_subclass_XML_name ,
```

```
assembly_member_constraint_list)
```

```
assembly_member_constraint_list ::= assembly_member_constraint
```

```
{assembly_member_constraint}
```

```
assembly_member_constraint is a subclass of assembly_member
```

```
assembly_member_constraint =
```

```
((block_subclass_name | member_assembly_subclass_name) ,
```

```

assembly_member_constraint_usage_name,
assembly_member_constraint_usage_XML_name,
requirements_flag,
min_occurs,
max_occurs,
content_restriction_flag)

```

4.4.1 Superclass properties

4.4.1.1 Syntax Independent

- assembly_name - The name of the Assembly abstract superclass. MUST conform to naming conventions defined in 4.10.
assembly_name ::= XML_string
- assembly_UID - The unique identifier assigned to the Assembly.
- assembly_description - A description of the content of the Assembly.
- member_assembly_name - The assembly_name of an Assembly that is a member of this Assembly.
- assembly_type_flag - As defined in section 4.8.5 on entity_type_flag. Indicates the nature of the object described by the Assembly content.
- For each member of the Assembly:
 - assembly_member_usage_name - Is the name of the member as used within the Assembly. MUST conform to naming conventions specified in section 4.10.
 - member_assembly_name - The assembly_name of an Assembly that is a member of this Assembly.
 - block_name - Is as defined in section 4.3 on Block.

4.4.1.2 XML Syntax Dependent

- assembly_XML_name - The name of the complexType that represents the Assembly abstract superclass, based on assembly_name, removing characters to conform to the rules of nmtoken.

assembly_XML_name ::= XML_nmtoken

- assembly_subclass_UID - The unique identifier assigned to the Assembly subclass.
- assembly_member_constraint_usage_XML_name – XML name of the Assembly member

assembly_member_constraint_usage_XML_name ::= XML_nmtoken

- For each member of the Assembly:
 - assembly_member_usage_XML_name - The name of the element representing the member Block or Assembly as used within the Assembly.

assembly_member_usage_XML_name ::= XML_nmtoken

4.4.2 Subclass properties

4.4.2.1 Syntax independent

- `assembly_subclass_name` - The name of the Assembly subclass. The name SHALL be formed by appending a two digit sequence number to the superclass `assembly_name`.

**`assembly_subclass_name ::= assembly_name [0,1,2,...,9]
[0,1,2,...,9]`**

- For each Assembly member of the Assembly
 - `member_assembly_subclass_name` - The `assembly_subclass_name` of a concrete subclass that is a member of the concrete subclass of the Assembly. MUST be the name of a concrete subclass of the abstract superclass named by `member_assembly_name`.
- For each Block member of the Assembly
 - `block_subclass_name` - Is as defined in section 4.3 on Block. MUST be the name of a Block subclass that is derived from the abstract superclass Block defined in the Assembly abstract superclass member.
- For each member of the Assembly
 - `assembly_member_constraint_usage_name` – the name of the usage of the member in the Assembly. Defaults to the usage name of the member in the abstract superclass Assembly, but may be overridden.
 - `requirements_flag` - As defined in section 4.8.1. Indicates the requirement restriction imposed on the member of the Assembly member constraint, that is, the requirements on the presence of the member within the Assembly_subclass.
 - `min_occurs` - As defined in section 4.8.2. Indicates the minimum number of occurrences permitted for the Assembly constraint member, that is, the member within the Assembly_subclass.
 - `max_occurs` - As defined in section 4.8.3. Indicates the maximum number of occurrences permitted for the Assembly member constraint, that is, the member within the Assembly_subclass.
 - `content_restriction_flag` - As defined in section 4.8.4. Indicates the content restriction applying to a subset of members within the Assembly_subclass.

4.4.2.2 XML syntax dependent properties

- `assembly_XML_name` - The name of the complexType that represents the Assembly abstract superclass, based on `assembly_name`, removing characters to conform to the rules of nmtoken.
- For each member of the Assembly
 - `assembly_member_constraint_usage_XML_name` – the XML name of the usage of the member of the Assembly that is derived from the `assembly_member_constraint_usage_name` according to section 4.10.
 - `assembly_subclass_XML_name` - The name of the complexType that represents the Assembly concrete subclass. The name is

formed by appending a two digit sequence number to the name of the parent abstract superclass Assembly, assembly_XML_name.

- **assembly_subclass_XML_name ::= assembly_XML_name [0,1,2,...,9] [0,1,2,...,9]**

4.5 Templates

A template is the basis from which documents are created. Templates are an abstract syntactic entity. Templates do not have all internal syntactic entities referenced down to primitives. (i.e. Slots are abstract sub-entities)

Abstract Superclass:

```
template = (template_name,
template_UID,
template_description,
template_family,
business_process { business_process },
business_process_family { business_process_family },
business_sub_process { business_sub_process },
business_sub_process_family { business_sub_process_family },
triggering_event_description,
slot_list)
```

```
slot_list ::= slot_entry[{slot_entry}]
```

```
slot_entry = (slot_name,
slot_UID,
slot_XML_name,
slot_purpose,
slot_detail_flag,
slot_type_flag,
slot_module_list,
abstract_requirements_flag,
abstract_repeatability_flag)
```

```
slot_type_flag ::= entity_type_flag
```

```
slot_module_list ::= [{module_entry}]
```

```
module_entry = (module_name,
context_category_value_list)
```

```
context_category_value_list ::= [{context_category_value_tuple}]
```

```
context_category_value_tuple ::= context_category
context_category_identification_scheme_name
context_category_identifier_value
```

4.5.1 Superclass Properties

4.5.1.1 Syntax Independent properties

- `template_name` - The name of the template abstract superclass. MUST conform to naming conventions defined in 4.10.
- `template_UID` - The unique identifier assigned to the template.
- `template_description` - A description of the content of the template.
- `template_family` - The name of a family of templates which this template is a member.
- `business_process` - The name of a business process of which this template is a member. MUST be a `context_category_identifier_value` from the UN/CEFACT Catalogue of Common Business Processes.
- `business_process_family` - The name of a business process family of which this template is a member. MUST be a `context_category_identifier_value` from the UN/CEFACT Catalogue of Common Business Processes.
- `business_sub_process` - The name of a business sub process of which this template is a member. MUST be a `context_category_identifier_value` from the UN/CEFACT Catalogue of Common Business Processes.
- `business_sub_process_family` - The name of a business sub process family of which this template is a member. MUST be a `context_category_identifier_value` from the UN/CEFACT Catalogue of Common Business Processes.
- `triggering_event_description` - A description of an event, or set of events, that cause generation of a Document using this template
- `context_category` - As defined in section 4.8.10. Indicates the set of contexts in which the template can be applied.
- For each slot in the Template
 - `slot_name` - Is the name of the slot member as used within the template. MUST conform to naming conventions specified in section 4.10.
 - `slot_UID` - The unique identifier assigned to the slot.
 - `slot_purpose` - A description of the purpose fulfilled by a slot in the template
 - `slot_detail_flag` - As defined in section 4.8.9. Indicates if the slot is part of the repeating detail of the template
 - `abstract_requirements_flag` - As defined in section 4.8.7. Indicates the requirement restriction imposed on the slot of the template, that is, the requirements on the presence of of the slot within the template.
 - `abstract_repeatability_flag` - As defined in section 4.8.8. Indicates the repeatability restriction imposed on the slot of the template, that is, if the slot can be repeated within the template. This repeatability is independent of the detail repeating aspect.
 - `context_category` - As defined in section 4.8.10. Indicates the set of contexts in which the template can be applied.
 - `slot_type_flag` - A slot primarily describes a person, resource, event, or location. Valid values for `slot_type_flag` are described in section 4.8.5 on `entity_type_flag`.

4.5.1.2 XML syntax dependent properties

- `xlot_XML_name` - The XML Slot Name is used for the name of the XML element representing the slot in an instance of the XML representation of the document. The XML Slot Name SHALL be based on the `slot_name`, modified as appropriate to conform to the requirements of `nmtoken`.
- `slot_XML_name ::= XML_nmtoken`

4.6 Modules

A Module is a syntactic entity that fills slots. Each Module is composed of one or more assemblies and/or Blocks. Modules are concrete subclasses, their abstract superclass parents are Slots in Templates. Unlike the other subclass to superclass relationships in the CICA architecture, the construction of the child is not determined by its parent. The primary requirement is that each Module must fulfill the purpose defined for the Slot.

Concrete Class:

```
module ::= assembly_subclass | block_subclass [{assembly_subclass |
block_subclass }]
```

```
module = (module_name,
  module_UID,
  module_XML_name,
  module_description,
  module_type_flag,
  module_node_list,
  context_category_value_list,
  responsible_subcommittee_name)
```

```
module_node_list ::= module_node {module_node}
```

```
module_node = ( (assembly_subclass_name | block_subclass_name),
  module_node_usage_name,
  module_node_usage_XML_name,
  requirements_flag,
  min_occurs,
  max_occurs,
  content_restriction_flag)
```

4.6.1 Module Properties

4.6.1.1 Syntax Independent properties

- `module_name` - The name of the Module concrete superclass. MUST conform to naming conventions defined in 4.10.
- `module_UID` - The unique identifier assigned to the Module.
- `module_description` - Description for the Module.

- `responsible_subcommittee_name` - The name of the ASC X12 subcommittee responsible for the maintenance of the document.
- `module_type_flag` - A slot primarily describes a person, resource, event, or location. Valid values for `module_type_flag` are described in section 4.8.5 on `entity_type_flag`.
- `context_category` - As defined in section 4.8.10. Indicates the set of contexts in which the Module applies.
- For each member of the Module
 - `module_node_usage_name` - Is the name of the Module member as used within the Module. MUST conform to naming conventions specified in section 4.10.
 - `requirements_flag` - As defined in section 4.8.1. Indicates the requirement restriction imposed on the member of the Module member constraint, that is, the requirements on the presence of the member within the Module concrete subclass.
 - `min_occurs` - As defined in section 4.8.2. Indicates the minimum number of occurrences permitted for the Module constraint member, that is, the member within the Module concrete subclass.
 - `max_occurs` - As defined in section 4.8.3. Indicates the maximum number of occurrences permitted for the Module constraint member, that is, the member within the Module concrete subclass.
 - `content_restriction_flag` - As defined in section 4.8.4. Indicates the content restriction applying to a subset of Block or Assembly members within the Module.

4.6.1.2 XML syntax dependent properties

- `module_XML_name` – The name of the complexType representing the Module. The `module_XML_name` is derived from `module_name` according to the rules described in section 4.10.2

`module_XML_name ::= XML_nmtoken`

- `module_node_usage_XML_name` - The name of the element representing the member Assembly subclass name or Block subclass name as used within the Module.

`module_node_usage_XML_name ::= XML_nmtoken`

4.7 Documents

Documents are completed frameworks, or templates. A Document is defined when a template has all its slots filled with Modules. A document has all internal syntactic entities resolved down to Primitive elements.

Concrete Subclass:

```
document ::= module {module}
```

```
document = (document_name ,  
  document_UID ,  
  document_XML_name ,  
  document_description ,
```

```

    template_name ,
    context_category_value_list ,
    responsible_subcommittee_name ,
    slotted_module_list ,
    detail_repeat )

slotted_module_list ::= slotted_module_node {slotted_module_node }

slotted_module_node = ( module_name ,
    module_usage_name ,
    module_usage_XML_name ,
    requirements_flag ,
    min_occurs ,
    max_occurs )

```

4.7.1 Subclass Properties

4.7.1.1 Syntax Independent properties

- document_name - The name of the document concrete subclass. MUST conform to naming conventions defined in 4.10.
- document_UID - The unique identifier assigned to the document.
- document_description - Description for the document.
- template_name - The name of the template abstract superclass that is the framework for this document.
- context_category - As defined in section 4.8.10. Indicates the set of contexts in which the document applies.
- responsible_subcommittee_name - The name of the ASC X12 subcommittee responsible for the maintenance of the document.
- detail_repeat - As defined in section 4.8.13. Indicates the number of times that the detail area may repeat.
- For each Module in the Document
 - module_name - As defined in 4.6.1.1
 - module_usage_name - The name of the Module as used in the document. Defaults to the slot name, but may be over-ridden.
 - requirements_flag - As defined in section 4.8.1. Indicates the requirement restriction imposed on the Module.
 - min_occurs - As defined in section 4.8.2. Indicates the minimum number of occurrences permitted for the Module.
 - max_occurs - As defined in section 4.8.3. Indicates the maximum number of occurrences permitted for the Module.

4.7.1.2 XML syntax dependent properties

- document_XML_name - The name of the top level element representing the document.

document_XML_name ::= XML_nmtoken

- module_usage_name - The name of the Module in the document.
- module_usage_XML_name - The name of the element representing the Module in the document.

module_usage_XML_name ::= XML_nmtoken

4.8 Common Symbols

These are symbols referenced throughout the document. Their definition is provided here by describing them in terms of their value space and lexical space.

4.8.1 requirements_flag

4.8.1.1 Value Space

Indicates that the presence of the member within its parent entity is either mandatory, optional, or not-used. The value of the requirements flag MUST be consistent with that of the min occurs property, i.e., the requirements flag MUST indicate if mandatory if the min occurs is greater than one, and vice versa, and the requirements flag MUST indicate optional if the min occurs is equal to zero.

4.8.1.2 Lexical Space

requirements_flag ::= M | O | N

where the character 'M' represents mandatory, 'O' represents optional, and 'N' indicates the member is not used.

Note: requirements_flag is only used on the members of the concrete subclasses Component_subclass, Block_subclass, Assembly_subclass, and Module.

4.8.2 min_occurs

4.8.2.1 Value Space

A non-negative integer.

4.8.2.2 Lexical Space

min_occurs ::= [0,1,2, ...]

Note: min_occurs is used on the members of the concrete subclasses Component_subclass, Block_subclass, Assembly_subclass, and Module.

4.8.3 max_occurs

4.8.3.1 Value Space

A positive integer or positive infinity.

4.8.3.2 Lexical Space

max_occurs ::= [1,2,3,...] | unbounded

where the string "unbounded" represents positive infinity.

Note: max_occurs is used on the members of the concrete subclasses Component_subclass, Block_subclass, Assembly_subclass, and Module.

4.8.4 content_restriction_flag

4.8.4.1 Value Space

Identifies that the member is an element of a subset of the parent entity's members sharing content restriction. A parent entity MUST have no more than one such content-restricted subset from the set of its members. The content restriction is one of:

1. Exclusive OR, meaning one and only one member of the subset may be present
2. Inclusive OR, meaning at least one of one member of the subset may be present
3. No content restriction

The value of the content restriction flag **MUST** be consistent with the value of the requirements flag. If there is a content restriction, the requirements flag **MUST** indicate the member is optional.

4.8.4.2 Lexical Space

content_restriction_flag ::= X | A | Null

where the character 'X' represents an Exclusive OR, 'A' represents an Inclusive OR, and the absence of either represents no content restriction.

Note: content_restriction_flag is used on Component, Block, Assembly, Module.

4.8.5 entity_type_flag

4.8.5.1 Value Space

Identifies the type of a CICA construct. The entity_type_flag is one of:

1. Party
2. Location
3. Event
4. Resource

4.8.5.2 Lexical Space

entity_type_flag ::= P | L | E | R

Where the character 'P' represents a Party, 'L' represents a Location, 'E' represents an Event, and 'R' represents a Resource.

Note: entity_type_flag is only used on the abstract superclass of the corresponding CICA construct.

4.8.6 identification_characteristic_flag

4.8.6.1 Value Space

Indicates the nature of a Block member. The identification characteristic flag is one of:

1. Identification
2. Characteristic

4.8.6.2 Lexical Space

identification_characteristic_flag ::= I | C

where the character 'I' represents identification and 'C' represents characteristic.

Note: identification_characteristic_flag is only used on the members of the abstract superclass Block

4.8.7 abstract_requirements_flag

4.8.7.1 Value Space

Indicates that the presence of the member within its parent entity is either mandatory or optional. The value of the requirements flag **MUST** be consistent with that of the min occurs property, i.e., the requirements flag **MUST** indicate mandatory if the min occurs is greater than one, and vice versa, and the requirements flag **MUST** indicate optional if the min occurs is equal to zero.

4.8.7.2 Lexical Space

abstract_requirements_flag ::= M | O

where the character 'M' represents mandatory and 'O' represents optional.

Note: **abstract_requirements_flag** is only used on the members of the abstract superclass template

4.8.8 abstract_repeatability_flag

4.8.8.1 Value Space

Indicates that the member can be present one or multiple times. In either case, the presence of one member (or multiple members) **MAY** be affected by a Requirement Flag also associated with the member. The repeatability flag is one of:

1. Singular, meaning the member may appear once and only once.
2. Multiple, meaning the member may appear one or more times

4.8.8.2 Lexical Space

abstract_repeatability_flag ::= S | M

where the character 'S' represents Singular, and 'M' represents multiple

Note: **abstract_repeatability_flag** is only used on the members of the abstract superclass template

4.8.9 slot_detail_flag

4.8.9.1 Value Space

Indicates if the slot is part of the detail section of the template. All Modules that fill slots with the **slot_detail_flag** set to **detail** will repeat as a group.

4.8.9.2 Lexical Space

slot_detail_flag ::= D | null

null - represents the absence of data.

Note: **slot_detail_flag** is only used on the members of the abstract superclass template

4.8.10 context_category

4.8.10.1 Value Space

These are the eight approved context categories defined in section 6.2.2 of the CCTS :

Business Process
 Product Classification
 Industry Classification
 Geopolitical
 Official Constraints
 Business Process Role
 Supporting Role
 System Capabilities

4.8.10.2 Lexical Space

context_category ::= BP | PC | IC | GP | OC | BR | SR | SC

Where the codes represent the following values:

BP Business Process
 PC Product Classification
 IC Industry Classification
 GP Geopolitical
 OC Official Constraints
 BR Business Process Role
 SR Supporting Role
 SC System Capabilities

4.8.11 context_category_identification_scheme_name

4.8.11.1 Value Space

The name of the identification scheme as specified in section 6.6.2 of the CCTS.

context_category_identification_scheme_name ::= CCT_identification_scheme

4.8.11.2 Lexical Space

NOTE: This space contains the names of the identification schemes referenced in the CCTS. However, some of the context categories do not identify identification schemes. So, this lexical space only defines the known entries, and other entries are possible as indicated by the wild card asterisk.

"UN/CEFACT Catalogue of Common Business Processes" |
 "Universal Standard Product and Service Specification (UNSPSC)" |
 "Standard International Trade Classification (SITC Rev .3)" |
 "Harmonized Commodity Description and Coding System (HS)" |
 "Classification Of the purposes of non Profit Institutions serving households (COPI)" |
 "International Standard Industrial Classification (ISIC)" |
 "Universal Standard Product and Service Specification (UNSPSC) Global" |
 "ISO 3166" |
 "UN/EDIFACT Code List for DE 3035 Party Roles" |
 *

4.8.12 context_category_identifier_value

The value space and lexical space are as defined in the relevant scheme.

context_category_identifier_value ::= CCT_identifier_value

4.8.13 detail_repeat

4.8.13.1 Value Space

A positive integer or positive infinity.

4.8.13.2 Lexical Space

detail_repeat ::= [1,2,3,...] | **unbounded**

where the string "unbounded" represents positive infinity.

Note: **detail_repeat** is used on the document.

4.9 Symbols from Other Sources

This subsection provides the source definition for symbols used in this document but whose definitions are specified in other documents. The documents listed in the Source column are fully specified in section 2.

Symbol	Source	Term or Definition in Source
Nmtoken	Extensible Markup Language (XML)	NMTOKEN
CCT	ebXML Core Components Technical Specification	One of the Approved Core Component Types as listed in table 8.1

4.10 Naming Conventions

4.10.1 Item and Usage Names

In general, item names (such as `assembly_name` and `block_name`) and usage names (such as `block_usage_name`) shall conform to the rules for constructing the "property term" as defined in the UN/CEFACT ebXML Core Components Technical Specification. The item name SHOULD in most cases be more general than a usage name, since the usage name represents an instance of the item when used in a larger construct. In most cases, the property term SHOULD correspond to a common business term.

Item names, except for Primitive names, SHALL have the item's CICA type name appended. For example, a Block that conveyed an individual's name would be "Person Name Block". Primitives SHALL have the CCT representation term appended.

Usage names SHALL be a suitable property term with nothing appended, with one exception. Primitives that represent codes, and coded supplementary components within Primitives, shall have "Code" appended to the name.

Usage names of supplementary components within Primitives SHALL be as defined in the corresponding Core Component Type, as defined in the UN/CEFACT ebXML Core Components Technical Specification

4.10.2 XML Names

Two types of XML names are required:

1. Element and attribute names - These represent usage names.

2. Simple and complex type names - These represent item names as well as intermediate, generic type names for Primitive content and supplementary components required when following certain conventions for using schema language.

Usage names for XML are constructed from the base corresponding CICA usage name. For Assembly, Block, Component, and Primitive, the XML type name is constructed from the base corresponding item name. These XML names SHALL be constructed from the base names according to the following algorithm:

1. Remove spaces between words
2. Capitalize the initial letter of each word, and set the remaining letters to lower case. (This is commonly called upper camel case)
3. If a word in the usage name appears on the list of X12 list of approved abbreviations, replace the word with the corresponding abbreviation, including the specified capitalization of the letters in the abbreviation.

The XML type name of the content component of a Primitive SHALL be constructed according to the following production:

content_component_type_XML_name ::= primitive_XML_name_Content

The XML type name of a supplementary component of a Primitive SHALL be constructed according to the following production:

supplementary_component_type_XML_name ::= primitive_XML_name_CCT_member_XML_name

primitive_XML_name_CCT_member_XML_name ::= XML_nmtoken

All XML usage and type names MUST satisfy the production of NMTOKEN, as specific in the W3C XML Recommendation, Version 1.0.

4.10.3 Generic versus Specific Usage Names

In X12 EDI syntax it is common to use a qualifier and value pair to fully specify the semantics of an item of data. For example, in the N1 Name segment, Data Element 66, Identification Code Qualifier, in N1_03, specifies the type of identification code that appears in Data Element 67, Identification Code, in N1_04. So, D-U-N-S Number is conveyed by placing a "1" in N1_03, and the DUNS number in N1_04.

In contrast, CICA is intended to support a greater degree of semantic specificity in usage names. However, a generic usage name with qualifier/value scheme is supported for instances where message designers determine that a generic usage name satisfies business requirements better than a specific usage name. Reasons for this determination may include, but may not be limited to, a large set of specific usage names, a dynamic set of usage semantics, or qualifiers based on existing external code lists.

The following general principle SHALL apply when a Primitive is used more than once within the Block and each usage has different semantics: A specific usage name for each usage SHOULD be assigned if there are less than ten different semantic usages. If there are ten or more different semantic usages, a generic usage name

with a qualifying required supplementary component MAY be used instead of specific usage names.

5 W3C XML Schema Language Schema CICA Representation

A CICA document is represented in W3C XML Schema Language by a single schema document that defines XSD constructs that correspond only to the document, Module, and concrete subclasses for the referenced assemblies, Blocks, Components, and Primitives. The following subsections describe the specific XSD representation of each of these CICA entities. In addition, subsection 5.1 specifies the overall structure of a schema for a CICA document

The relevant XSD fragments are represented in BNF notation, using literal values for XSD syntax and terminal symbols defined in section 4 for CICA entities. Where necessary, additional terminal symbols required by XSD are defined in this section.

5.1 Overall Structure

This subsection defines the XSD declarations for items such as the schema root document element with associated attributes and documentation, required both by XSD for a compliant schema document and by X12's use of XSD. A CICA document SHALL be represented in W3C XML Schema Language (or XSD) by a single, fully self-contained schema document. The name of the schema document, usually as stored on a disk drive or referenced as part of a URL, SHALL be the XML name of the document with ".xsd" appended.

```

CICA_schema ::=
<?xml version ="1.0" encoding="utf-8"?>
<xs:schema version="X12_version" ↵
xmlns:xs="http://www.w3.org/2001/XMLSchema" ↵
xmlns="X12_namespace" targetNamespace="X12_namespace" ↵
elementFormDefault="unqualified" attributeFormDefault="unqualified">
CICA_document_definition
CICA_module_definition {CICA_module_definition}
{CICA_assembly_definition}
CICA_block_definition {CICA_block_definition}
{CICA_component_definition}
CICA_primitive_definition {CICA_primitive_definition}
</xs:schema>

```

The following symbols are defined for this production:

- **CICA_schema** – a CICA document represented in W3C XSD schema.
- **X12_version** - This is the coded X12 version and release, as identified by Data Element 480, the Version/Release/Industry Identifier Code, from the X12 Data Element Dictionary. However, only the six digit major version/minor release format is permitted.
X12_version ::= [0,1,2,...,9] [0,1,2,...,9] [0,1,2,...,9] [0,1,2,...,9] [0,1,2,...,9] [0,1,2,...,9]
- **X12_namespace** - This is a Uniform Resource Name. The production is as follows:
X12_namespace ::= urn:x12:schemas:X12_version

Other symbols are defined in the succeeding sections.

5.2 Document

A CICA document is represented in XSD with an element declaration an anonymous, in-line complexType declaration. The complexType declaration has a sequence content model as defined in the W3C XML Schema Recommendation, Part 1. Each of the member elements of the sequence corresponds to a Module.

```

CICA_document_definition ::=
  <xs:element name="document_XML_name">
    <xs:complexType>
      <xs:annotation>
        <xs:documentation>
          document_documentation
        </xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element name="module_usage_XML_name" ↵
          type="module_XML_name" ↵
          minOccurs="min_occurs" ↵
          maxOccurs="max_occurs"/>
        [
          <xs:element name="detail" ↵
            maxOccurs="detail_repeat">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name= ↵
                    "slotmodule_usage_name"
                      type="module_XML_name" ↵
                      minOccurs="min_occurs" ↵
                      maxOccurs="max_occurs"/>
                  {
                    <xs:element name= ↵
                      "slotmodule_usage_name"
                        type="module_XML_name" ↵
                        minOccurs="min_occurs" ↵
                        maxOccurs="max_occurs"/>
                  }
                </xs:sequence>
              </xs:complexType>
            </xs:element>

```

```
    ]  
    {<xs:element name="slotmodule_usage_name" ↵  
      type="module_XML_name" ↵  
      minOccurs="min_occurs" ↵  
      maxOccurs="max_occurs"/>  
    }  
  <xs:sequence>  
  </xs:complexType>  
</xs:element>
```

The following symbols are defined for this production:

- **document_documentation** - The specific contents of the documentation element are beyond the scope of this document at this time. It MAY be filled as standards developers best determine at time of schema generation

document_documentation ::= XML_string

- **slotmodule_usage_name** - the name of the Module, from the slot XML name defined on the template or the Module XML named defined by the document.

slotmodule_usage_name ::= module_usage_name | slot_XML_name

- **detail** - the element name for the detail section of the document.

detail ::= Detail

ASC X12
- WORKING DRAFT -

5.3 Module

A CICA Module is represented in XSD with a complexType declaration. The complexType declaration has a sequence content model as defined in the W3C XML Schema Recommendation, Part 1. Each of the member elements of the sequence corresponds to an Assembly or Block.

```

CICA_module_definition ::=
  <xs:complexType name="module_XML_name">
    <xs:annotation>
      <xs:documentation>
        module_documentation
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="module_node_usage_XML_name" ↵
        type="assembly_subclass_XML_name | ↵
          block_subclass_XML_name" ↵
        minOccurs="min_occurs" ↵
        maxOccurs="max_occurs"/>
      {
        <xs:element name="module_node_usage_XML_name" ↵
          type="assembly_subclass_XML_name | ↵
            block_subclass_XML_name" ↵
          minOccurs="min_occurs" ↵
          maxOccurs="max_occurs"/>
      }
    </xs:sequence>
  </xs:complexType>

```

The following symbols are defined for this production:

- **module_documentation** - The specific contents of the documentation element are beyond the scope of this document at this time. It MAY be filled as standards developers best determine at time of schema generation

module_documentation ::= XML_string

5.4 Assembly

A CICA Assembly is represented in XSD with a complexType declaration. The complexType declaration has a sequence content model as defined in the W3C XML Schema Recommendation, Part 1. Each of the member elements of the sequence corresponds to an Assembly or Block.

```

CICA_assembly_definition ::=
  <xs:complexType name="assembly_subclass_XML_name">
    <xs:annotation>

```

```

    <xs:documentation>
        assembly_documentation
    </xs:documentation>
</xs:annotation>
<xs:sequence>
    <xs:element ↓
name="assembly_member_constraint_usage_XML_name" ↓
        type="assembly_subclass_XML_name | ↓
        block_subclass_XML_name" ↓
        minOccurs="min_occurs" ↓
        maxOccurs="max_occurs"/>
    {
        <xs:element ↓
name="assembly_member_constraint_usage_XML_name" ↓
        type="assembly_subclass_XML_name | ↓
        block_subclass_XML_name" ↓
        minOccurs="min_occurs" ↓
        maxOccurs="max_occurs"/>
    }
</xs:sequence>
</xs:complexType>

```

The following symbols are defined for this production:

- **assembly_documentation** - The specific contents of the documentation element are beyond the scope of this document at this time. It MAY be filled as standards developers best determine at time of schema generation

assembly_documentation ::= XML_string

5.5 Block

A CICA Block is represented in XSD with a complexType declaration. The complexType declaration has a sequence content model as defined in the W3C XML Schema Recommendation, Part 1. Each of the member elements of the sequence corresponds to a Primitive or Component

```

CICA_block_definition ::=
    <xs:complexType name="block_subclass_XML_name">
        <xs:annotation>
            <xs:documentation>
                block_documentation
            </xs:documentation>
        </xs:annotation>
    </xs:sequence>

```

```

    <xs:element name=↵
"block_member_constraint_usage_XML_name"↵
    type="component_subclass_XML_name | ↵
    primitive_subclass_XML_name" ↵
    minOccurs="min_occurs" ↵
    maxOccurs="max_occurs"/>
    {
    <xs:element ↵
name="block_member_constraint_usage_XML_name"
    type="component_subclass_XML_name | ↵
    primitive_subclass_XML_name" ↵
    minOccurs="min_occurs" ↵
    maxOccurs="max_occurs"/>
    }
<xs:sequence>
</xs:complexType>

```

- **block_documentation** - The specific contents of the documentation element are beyond the scope of this document at this time. It MAY be filled as standards developers best determine at time of schema generation.

block_documentation ::= XML_string

5.6 Component

A CICA Component is represented in XSD with a complexType declaration. The complexType declaration has a sequence content model as defined in the W3C XML Schema Recommendation, Part 1. Each of the member elements of the sequence corresponds to a Primitive.

```

CICA_component_definition ::=
<xs:complexType name="component_subclass_XML_name">
<xs:annotation>
<xs:documentation>
    component_documentation
</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element ↵
name="component_member_constraint_usage_XML_name" ↵
    type="primitive_subclass_XML_name" ↵
    minOccurs="min_occurs" ↵
    maxOccurs="max_occurs"/>

```

```

<xs:element ↵
name="component_member_constraint_usage_XML_name" ↵
    type="primitive_subclass_XML_name" ↵
    minOccurs="min_occurs" ↵
    maxOccurs="max_occurs"/>
}
<xs:element ↵
name="component_member_constraint_usage_XML_name" ↵
    type="primitive_subclass_XML_name" ↵
    minOccurs="min_occurs" ↵
    maxOccurs="max_occurs"/>
}
<xs:sequence>
</xs:complexType>
    
```

The following symbols are defined for this production:

- **component_documentation** - The specific contents of the documentation element are beyond the scope of this document at this time. It MAY be filled as standards developers best determine at time of schema generation

component_documentation ::= XML_string

- **component_member_constraint_usage_XML_name –**

component_member_constraint_usage_XML_name ::= XML_nmtoken

5.7 Primitives

Primitives are represented in Schema as a ComplexType. Each Primitive is derived from a CCT. Each CCT content and supplementary component is represented by a SimpleType definition with applicable restrictions (reference section 4.1.1). Each SimpleType will be generated based on the XSD data type.

Table 5.1 lists the CICA data types and the corresponding schema language data types.

datatype ::= XSD_datatype, substituted according to table 5.7.1

CICA Data Type	Schema Language datatype
string	String
token	Token
boolean	Boolean
Integer	Integer
decimal_number	decimal_number
date	Date
time	Time
date_time	dateTime

Table 5.7.1

Table 5.7.2 lists the CICA facets and the corresponding schema language facets.

CICA Facet	Schema language facet
pattern	Pattern
length	Length
min_length	minLength
max_length	maxLength
min_inclusive	minInclusive
max_exclusive	maxExclusive
min_exclusive	minExclusive
max_inclusive	maxInclusive
total_digits	totalDigits
fraction_digits	fractionDigits
enumeration	enumeration

Table 5.7.2

Each CCT Content Component is represented in schema as:

```
<xs:simpleType name = "content_component_type_XML_name">
  <xs:restriction base="xs:XSD_datatype">
    [{ constraining_facet }]
  </xs:restriction>
</xs:simpleType>
```

Each CCT Supplementary Component is represented in schema as:

```
<xs:simpleType name = "supplementary_component_type_XML_name">
  <xs:restriction base="xs:XSD_datatype">
    [{ constraining_facet }]
  </xs:restriction>
</xs:simpleType>
```

ASC X12
- WORKING DRAFT -

The allowable constraining facets are represented as:

```

constraining_facet ::= <xs:pattern value="pattern"> |
<xs:length value="length"> |
<xs:minLength value="min_length"> |
<xs:maxLength value="max_length"> |
<xs:minInclusive value="min_inclusive"> |
<xs:maxInclusive value="max_inclusive"> |
<xs:minExclusive value="min_exclusive"> |
<xs:maxExclusive value="max_exclusive"> |
<xs:enumeration value="enumeration_value">

```

The Primitive is represented in schema as a complexType referencing the simpleType declarations above as allowed (reference section 4.1.1).

```

CICA_primitive_definition ::=
<xs:complexType name = "primitive_subclass_XML_name">
  <xs:annotation>
    <xs:documentation>
      primitive_documentation
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="content_component_type_XML_name">
      <xs:attribute ↓
name="supplementary_component_XML_name" ↓
type="supplementary_component_type_XML_name"/>
      <!--...-->
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

The following symbols are defined for this production:

- **primitive_documentation** - The specific contents of the documentation element are beyond the scope of this document at this time. It MAY be filled as standards developers best determine at time of schema generation

5.7.1 Primitive Example

A Primitive of CCT **Amount** that is restricted to US and EU dollars, with fractional digits set to 2, and currencycode set to a min length of 1 and a max length of 2 would be created as follows:

Eg. <paidamount currency="us" currencycode="xx">6.00</paidamount>

```

<xs:element name = "PaidAmount" type = "WOD_Amount1">
</xs:element>
<xs:simpleType name = "Amount1_CurrencyID">
  <xs:restriction base = "xs:string">
    <xs:minLength value = "1"/>
    <xs:maxLength value = "2"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Amount1_CurrencyCodeListVersionID">
  <xs:restriction base = "xs:string">
    <xs:enumeration value="US">
    <xs:enumeration value="EU">
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Amount1_Content">
  <xs:restriction base = "xs:decimal">
    <xs:fractionDigits value = "2"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name = "Amount1">
  <xs:simpleContent>
    <xs:extension base="Amount1_Content">
      <xs:attribute name="currency" type=" ↴
Amount1_CurrencyID"/>
      <xs:attribute name="currencycode" type=" ↴
Amount1_CurrencyCodeListVersionID"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

5.8 Content Restriction

This subsection specifies the schema representation of CICA entities that are have a **content_restriction_flag** (as specified in section 4.8.4) with a value of "X" or "A", indicating an exclusive OR or an inclusive OR.

This subsection applies to CICA Component, Block, Assembly, and Module, and modifies the schema language representation specified in the previous sections. In general, all of the child members of these are included in a schema language group element, with choice elements applied to create appropriate choice content models.

The member elements of the construct are divided into two groups, **restricted_members** containing the members that have a content_restriction_flag flag value of “X” or “A”, and **unrestricted_members** that have a content_restriction_flag value of not present. The schema language sequence element in each of the productions is replaced as shown in sections 5.8.1 and 5.8.2. For both these sections, the **unrestricted_members** symbol is as defined in the previous section, repeated here in a generalized form for all subject CICA constructs.

```

unrestricted_members ::=
<xs:element name="member_constraint_usage_XML_name" ↵
    type="subclass_XML_name" ↵
    minOccurs="min_occurs" ↵
    maxOccurs="max_occurs"/>
{
<xs:element name="member_constraint_usage_XML_name" ↵
    type="subclass_XML_name" ↵
    minOccurs="min_occurs" ↵
    maxOccurs="max_occurs"/>
}

member_constraint_usage_XML_name ::=
module_member_constraint_usage_XML_name |
assembly_member_constraint_usage_XML_name |
block_member_constraint_usage_XML_name

module_member_constraint_usage_XML_name ::= XML_nmtoken

subclass_XML_name ::=
assembly_subclass_XML_name |
block_subclass_XML_name |
component_subclass_XML_name |
primitive_subclass_XML_name

restricted_members ::= exclusive_or_members | inclusive_or_members

```

5.8.1 Exclusive OR

The schema language representation of the CICA Exclusive OR content restriction is shown below:

```

exclusive_or_members ::=
<xs:choice>
  <xs:element name="member_constraint_usage_XML_name" ↵
    type="subclass_XML_name"/>
  <xs:element name="member_constraint_usage_XML_name" ↵
    type="subclass_XML_name"/>
  {
  <xs:element name="member_constraint_usage_XML_name" ↵
    type="subclass_XML_name"/>
  }
</xs:choice>

```

5.8.2 Inclusive OR

The schema language representation for the CICA inclusive OR content restriction is shown below, where there are from 1 to n members subject to the restriction. These members are designated below as **member_1** through **member_n**. The order of the members as listed in the syntax independent CICA representation defines a logical sequence used throughout the productions in this subsection.

In general, there is one parent choice element and n child sequence elements. For each of the n member elements there is a sequence element that has that member element as the first child element, with a `min_occurs` facet of 1, and all of the remaining elements in the sequence element having a `min_occurs` facet value of zero. After this member of the logical sequence has appeared as the initial child element of a sequence element, it is omitted from the remaining sequence elements.

```

inclusive_or_members ::=
<xs:choice>
  <xs:sequence>
    <xs:element name="member_1" ↵
      type="subclass_XML_name"/>
    <xs:element name="member_2" ↵
      type="subclass_XML_name" ↵
      minOccurs="0"/>
    ...
    <xs:element name="member_n" ↵
      type="subclass_XML_name" ↵
      minOccurs="0"/>
  </xs:sequence>
  <xs:sequence>
    <xs:element name="member_2" ↵
      type="subclass_XML_name"/>
    ...

```

```
<xs:element name="member_n" ↵
    type="subclass_XML_name" ↵
    minOccurs="0"/>
</xs:sequence>
<xs:sequence>
    <xs:element name="member_n" ↵
        type="subclass_XML_name"/>
</xs:sequence>
</xs:choice>
```

NOTE: The ellipses (...) in this production represent a progression. If there are only two members, the production is modified appropriately to include only two sequence elements corresponding to member_1 and member_2.

```
member_1 ::= member_constraint_usage_XML_name
member_2 ::= member_constraint_usage_XML_name
...
member_n ::= member_constraint_usage_XML_name
```

ASC X12

- WORKING DRAFT -