ISO TC 184/SC4/WG11 N136___ **Date:** 2000-09-29

Supersedes ISO TC 184/SC4/WG N134

ISO 10303-28.Working Draft XML Representation for Data sharing

ABSTRACT: This document describes data sharing messages that can be sent between a server and client using XML. The format of the generated XML has been optimized for readability and processing of STEP Application Interpreted Model data using Xpath. The format is compatible with the requirements of messaging protocols such as SOAP.

KEYWORDS:XML

COMMENTS TO READER: This working draft is being released for review before the Charleston meeting.

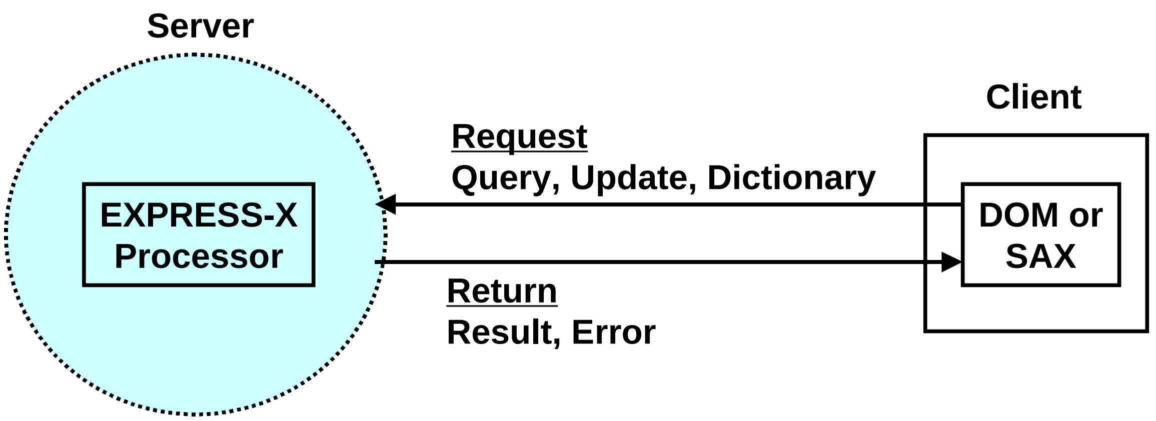
Project Leader: Peter Bergstom	**Document Editor: Martin Hardwick**
Address:	**Address: STEP Tools, Inc. 216 River Street Troy, NY 12180 USA**
Telephone:	**Telephone: +518 687-2848**
Telefacsimile:	**Telefacsimile: +518 687-4420**
Electronic mail:	**Electronic mail:hardwick@steptools.com**

CEB Binding
Draft 3.0

1 Background and context (Informative)

1.1 Purpose of the Binding

The Containment Early Binding (CEB) defines an XML representation for data sharing messages between a STEP server and a client application.



The use of containment gives the binding three qualities:

1. It makes STEP AIM information easier to understand. In many cases, the information can be understood without reference to EXPRESS-G diagrams or mapping tables because the organization of the mapping table is the same as the nesting of the binding.

2. It produces XML data that is easier to process using the Xpath (http://www.w3.org/TR/xpath) language of the Document Object Model (DOM) library and the Simple Access for XML (SAX) library. Xpath has a syntax similar to the notation used to find files in directory systems and like that notation is more powerful when the directories are organized into nested hierarchies.

3. It produces XML data compatible with the encoding conventions of the Simple Object Access Protocol (SOAP) (http://www.w3.org/TR/SOAP/).

To get these qualities the CEB sometimes duplicates items when they are nested. For some scenarios, this duplication may disqualify the CEB from being used as a canonical form for data exchange or data archiving applications. The target application for the CEB is client server data sharing. The CEB describes how to map EXPRESS information into XML elements so that it can be transmitted in messages between a client and server.

The CEB binding maps any EXPRESS defined information into XML, but it has been optimized to suite the style of STEP Application Interpreted Model information. The CEB binding describes elements that can be used as information content for any messaging protocol but it has been optimized for the style described for SOAP.

The CEB binding is divided into two levels. The first level requires the server to process EXPRESS defined information. The second level requires the server to process EXPRESS-X view bindings. In either case the requirement may be met using an EXPRESS or EXPRESS-X processing system, or using an application with implementation logic equivalent to the EXPRESS or EXPRESS-X specifications given in an EXPRESS schema or EXPRESS-X schema view.

1.2 Supported Scenarios

The following scenarios are supported:

— The client requests the server to export information using a STEP-QUERY message. The server returns the requested information as XML elements in a STEP-RESULT message.

— The first scenario with the following addition: The client modifies the XML elements returned in the STEP-RESULT message and sends the modifications back to the server in a STEP-UPDATE message.

— Before the client begins the first or second scenario it checks the server for compatibility with a STEP-DICTIONARY message.

In a Level I server, the client can access data by type or by identifier. In a Level II server, the client can also use EXPRESS-X view bindings to select data by value.

1.3 Design Decisions

The following design decisions were made for this binding:

1.3.1 Data Specification Language

EXPRESS is the data specification language used in this binding. The Document Template Description (DTD) language is not used as a data specification language for the following reasons:

— The DTD language was designed for simple applications. Unfortunately, product data applications are NOT simple so the extra simplicity of the DTD language makes it harder to use for these applications.

— Unlike EXPRESS, the DTD language is unable to scope the names of elements. For applications such as STEP this restriction requires unacceptably long names to be generated to disambiguate different instances of the same name.

— The SOAP protocol does not allow DTD's.

Even though the binding does not describe a DTD, it may be possible for a user to develop a DTD or an XML Schema for the data generated by this binding depending on the EXPRESS model.

1.3.2 Naming

When names have to be generated they always contain the character "-". This character is legal in XML identifiers, but not EXPRESS and EXPRESS_X identifiers.

1.3.3 Organization

Control information is put into XML attributes. EXPRESS and EXPRESS-X information is put into XML elements.

1.3.4 Address Information

The address of the server is assumed to be provided in a message header that is outside the scope of this binding. The following example contains a STEP-QUERY message. This message is encapsulated by a SOAP Envelope that contains the address of the server www.stepserver.com/ap203/tire1. In the example the SOAP envelope is posted to the server using an HTTP POST command. Only the ceb:STEP-QUERY element and its sub-elements are prescribed by this binding.

```
POST /STEP-QUERY HTTP/1.1
Host: www.stepserver.com/ap203/tire1
Content-Type: text/xml
Content-Length: nnnn
SOAPMethodName: urn:iso10303-28:ceb#STEP-QUERY

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
       <SOAP:Body>
          <ceb:STEP-QUERY
           xmlns:ceb=" urn:iso10303-28:ceb" >
                <product/>
          </ceb:STEP-QUERY>
      </SOAP:Body>
</SOAP:Envelope>
```

The return address of the client is also outside the scope of this binding. In the following example a STEP-RESULT message is formulated in response to the above STEP-QUERY message and returned to the client in another SOAP envelope. Only the ceb:STEP-RESULT element and its sub-elements are prescribed by this binding.

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
       <SOAP:Body>
           <ceb:STEP-RESULT xmlns:m="urn:iso10303-28:ceb">
               <Product id=" I-670740">
                     <id>TIRE</id>
                     <name>TIRE</name>
```

```
                      <description>NOT SPECIFIED</description>
                      <frame_of_reference id=” I-670730”
                          xsi:type=” Mechanical_context” >
                            <name></name>
                            <discipline_type>MECHANICAL</discipline_type>
                            <frame_of_reference id=” I-607020”
                               xsi:type=” Application_context” >
                                     <text>CONFIGURATION CONTROLLED 3D
                               DESIGNS OF MECHANICAL PARTS AND
                               ASSEMBLIES</text>
                            </frame_of_reference>
                      </frame_of_reference>
                   </Product>
             </ceb:STEP-RESULT>
        </SOAP:Body>
</SOAP:Envelope>
```

1.3.5 Meta Information

The binding assumes an SDAI Dictionary instance exists for the server. This dictionary instance is accessed using the STEP-DICTIONARY message. STEP-DICTIONARY is a special case of the STEP-QUERY message in which the STEP-QUERY is addressed to the SDAI Dictionary of the server instead of the server itself. The following example requests the server to return information about the schema of the data at www.stepserver.com/ap203/tire1. The EXPRESS model defining an SDAI Dictionary is defined in ISO 10303-22.

```
POST /STEP-DICTIONARY HTTP/1.1
Host: www.stepserver.com/ap203/tire1
Content-Type: text/xml
Content-Length: nnnn
SOAPMethodName: urn:iso10303-28:ceb#STEP-DICTIONARY

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
       <SOAP:Body>
          <ceb:STEP-DICTIONARY
            xmlns:ceb=“ urn:iso10303-28:ceb” >
                  <schema>
          </ceb:STEP-DICTIONARY>
      </SOAP:Body>
</SOAP:Envelope>
```

1.3.6 Error Processing

The binding defines error message for when the server detects an error in a message sent by the client. The mechanism used to deliver the messages is outside the scope of this binding. The following example contains a STEP-ERROR message. This message is encapsulated by a SOAP Envelope. Only the ceb:STEP-ERROR element and its sub-elements are prescribed by this binding.

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1" >
       <SOAP:Body>
           <SOAP:Fault>
```

```
                <faultcode>400</faultcode>
                <faultstring>
                    SOAP Must Understand Error
                </faultstring>
                <runcode>1</runcode>
                <detail xmlns:ceb="urn:iso10303-28:ceb"
                    <ceb:STEP-ERROR>
                         <name>Part</name>
                         <identifier/>
                         <code>10</code>
                         <description>Pick tag not found</description>
                    </ceb:STEP-ERROR>
               </detail>
            </SOAP:Fault>
       </SOAP:Body>
</SOAP:Envelope>
```

1.3.7 AND/OR Inheritance

EXPRESS object models are allowed to use a feature called AND/OR inheritance. The flexibility provided by AND/OR inheritance has allowed STEP to expand into new applications without requiring fundamental changes to the type hierarchies of critical elements already being used in other applications.

The Xpath language uses pattern matching functions to find nested elements. An Xpath application for object model data is easier to write if objects containing the same information contain the same elements. In particular, the same names should be used for the same attribute in the different objects in an inheritance hierarchy because this will allow an application to find the element representing an attribute without concern for the different cases in a hierarchy.

However, if two objects in an AND/OR combination have attributes with the same name then those names have to be disambiguated. This has the potential to cause the name of the attribute that is the target of an Xpath expression to change. For example, if the expression is searching for an attribute called name in an inheritance hierarchy rooted by an object type called X, and an AND/OR instance marries the X type with a type called Y that also defines a name attribute, then the name of name is in conflict between the two types.

This binding reduces this possibility by only placing the types of the instance that are relevant to the type of the attribute into the XML. For example, if an EXPRESS attribute has the type X and an entity instance has the type Y&Z where Y is a subtype of X and Z is not, then the attribute value will be described as a Y instance and the Z value will be ignored.

1.3.8 Post Processing

The form and content of each message described by this binding depends on the requirements of the message and the EXPRESS schema only.

Annex A defines a post-processor to convert the information content defined by the CEB binding into conformance with the conventions used in SOAP 1.0.

2 XML Representation for Data Sharing (Normative)

2.1 Definitions

For the purposes of this part of ISO 10303 the following definitions apply:

2.1.1 Argument tag. An XML element describing an EXPRESS-X view binding argument.

2.1.2 Attribute tag. An XML element describing an EXPRESS attribute instance.

2.1.3 Binding tag. An XML element describing a binding to an EXPRESS-X view.

2.1.4 Configure tag. An XML element describing how the client wants the server to configure EXPRESS and EXPRESS-X data.

2.1.5 Discriminator tag. An attribute tag or argument tag.

2.1.6 Delete tag. An entity tag marked for deletion by the client.

2.1.7 Instance tag. A binding, entity, rule or view tag.

2.1.8 Label tag. An XML element describing the type of a primitive or defined value.

2.1.9 Keyword tag. An XML element describing a parameter for a configure tag.

2.1.10 New tag. An entity tag created by the client.

2.1.11 Pick tag. An XML element describing an entity instance or entity type requested by the client.

2.1.12 Rule tag. An XML element describing the result of evaluating EXPRESS global rule.

2.1.13 Truncated tag. An XML element used to identify a binding, entity or view tag.

2.1.14 Value tag. A binding, entity, label or view tag.

2.1.15 View tag. An XML element describing the value computed by evaluating a binding to an EXPRESS-X view.

2.1.16 Update tag. A entity tag with one or more updated attributes.

2.2 Message Elements

2.2.1 Introduction

This binding puts EXPRESS defined information into XML elements called tags. The following kinds of tags are allowed: Argument tags, Attribute tags, Binding tags, Configuration tags, Delete tags, Entity tags, Keyword Tags, Label tags, New tags, Pick tags, Rule tags, Update tags and View tags. The different kinds of tags are further divided as follows:

— Binding, entity, rule and view tags are called Instance tags.
— Argument and attribute tags are called Discriminator tags.
— Binding, entity, label and view tags are called Value tags.

The following sub-sections each describe the canonical form of a tag.

2.2.2 Entity Tags

Entity tags describe EXPRESS entity instance information. Only attribute tags can be used as a discriminator in entity tags. Entity and label tags are always allowed to describe the values in an entity tag discriminator. Depending on the message, binding tags may also be allowed. The following template shows the canonical organization of an entity tag whose discriminators contains label and entity tags.

```
<entity-tag1 properties>
      <attribute-tag1 properties>PCDATA</attribute-tag1>
      <attribute-tag2 properties>
            <entity-tag2 properties> … </entity-tag2>
      </attribute-tag2>
      <attribute-tag3 properties>
            <label-tag3 properties>PCDATA</label-tag3>
      </attribute-tag3>
      <attribute-tag4 properties>
            <label-tag4 properties> … </label-tag4>
            <entity-tag4 properties> … </entity-tag4>
      </attribute-tag4>
</entity-tag1>
```

In the template attribute-tag1 represents an EXPRESS attribute defined by a value with a simple type, or a defined type of a simple type. Attribute-tag2 represents an EXPESS attribute define by an entity. Attribute-tag3 represents an EXPRESS attribute that requires a label to disambiguate a selection of primitive values. Attribute-tag4 is the most general case. This tag represents an attribute defined by a complex data type containing some combination of aggregates and selections.

2.2.3 Binding Tags

Binding tags describe EXPRESS-X binding calls for Level II Servers. Only argument tags can be used as a discriminator in binding tags. Binding, entity and label tags are allowed to describe the values in a binding tag discriminator. The following template shows the canonical organization of a binding tag.

```
<binding-tag1 properties>
      <argument-tag1 properties>PCDATA</argument-tag1>
      <argument-tag2 properties>
            <entity-tag2 properties> … </entity-tag2>
      </argument-tag2>
      <argument-tag3 properties>
            <label-tag3 properties>PCDATA</label-tag3>
      </argument-tag3>
      <argument-tag4 properties>
            <binding-tag4 properties> … </binding-tag4>
      </argument-tag4>
      <argument-tag5 properties>
            <label-tag5 properties> … </label-tag5>
            <entity-tag5 properties> … </entity-tag5>
            <binding-tag5 properties> … </binding-tag5>
      </argument-tag5>
</binding-tag1>
```

In the template, argument-tag1 describes an argument whose value is represented by a primitive type. Argument-tag2 describes an argument whose value is represented by an entity. Argument-tag3 describes an argument whose value must be disambiguated using a label. Argument-tag4 describes an argument whose value is to be computed by evaluating another binding. Argument-tag5 is the most general case. This tag represents an attribute defined by a complex data type containing some combination of aggregates and selections of values that may be defined by a primitive type, an entity type, or another binding type.

2.2.4 View Tags

A view-tag describes the result of evaluating an EXPRESS-X binding. Only attribute tags can be used as a discriminator in view tags. Entity, label and view tags are allowed to describe the values in a view tag discriminator. The following template shows the canonical organization of a view tag.

```
<view-tag1 properties>
      <attribute-tag1 properties>PCDATA</attribute-tag1>
      <attribute-tag2 properties>
            <entity-tag2 properties> … </entity-tag2>
      </attribute-tag2>
      <attribute-tag3 properties>
            <label-tag3 properties>PCDATA</label-tag3>
      </attribute-tag3>
      <attribute-tag4 properties>
            <view-tag4 properties> … </view-tag4>
      </attribute-tag4>
      <attribute-tag5 properties>
            <label-tag5 properties> … </label-tag5>
            <entity-tag5 properties> … </entity-tag5>
            <view-tag5 properties> … </view-tag5>
      </attribute-tag5>
</view-tag1>
```

2.2.5 Rule Tags

A rule-tag describes the result of evaluating a global rule. A rule tag does not contain any discriminator tags. The following template shows the canonical form of a rule-tag.

```
<rule-tag1>PCDATA</rule-tag1>
```

A rule-tag can only contain a value of type logical. A rule-tag can only appear as a root element.

2.2.6 Configure Tags

A configure tag tells the server how to format information. The following template shows the canonical form of a configure-tag.

```
<configure-tag1 properties>
      <keyword-tag1/>
</configure-tag1>
```

A configure-tag can only appear as a root element.

2.2.7 Pick Tags

A pick tag identifies entity instance information to be returned by the server to the client. The following template shows the canonical form of a pick-tag.

```
<pick-tag1 properties>
      <keyword-tag1/>
</pick-tag1>
```

A pick-tag can only appear as a root element.

2.2.8 Truncated Tags

A truncated tag identifies an entity, binding, or view tag. A truncated tag does not contain any discriminator tags. The identity property of a truncated tag must be that same as the identity property of a known entity, view or binding tag. The following template shows the canonical form of a truncated-tag.

```
<truncated-tag1 properties/>
```

2.2.9 New Tags

A new tag identifies an entity created by the client. A new tag may contain binding tags. A new tag has a status property with the value “new”. The following template shows the canonical form of a new-tag.

```
<entity-tag1 ceb:status=” new”  properties>
      <attribute-tag1 properties>PCDATA</attribute-tag1>
      <attribute-tag2 properties>
            <entity-tag2 properties> … </entity-tag2>
      </attribute-tag2>
      <attribute-tag3 properties>
            <label-tag3 properties>PCDATA</label-tag3>
```

```
        </attribute-tag3>
        <attribute-tag4 properties>
              <binding-tag4 properties> … </binding-tag4>
        </attribute-tag4>
        <attribute-tag5 properties>
              <label-tag5 properties> … </label-tag5>
              <entity-tag5 properties> … </entity-tag5>
              <binding-tag5 properties> … </entity-tag5>
        </attribute-tag4>
  </entity-tag1>
```

2.2.10 Delete Tags

A delete tag identifies an entity that the client wants deleted. A delete tag has a status property with the value “delete”. A delete tag may contain any kind of tag.

```
  <entity-tag1 ceb:status=” delete”  properties>
        <tag>
  </entity-tag1>
```

2.2.11 Update Tags

An update tag identifies an entity that the client wants updated. A delete tag has a status property with the value “update”. At least one attribute in the update tag must have a type value of update.

```
  <entity-tag1 ceb:status=” update”  properties>
        <attribute-tag1 ceb:type=” update” properties>
              …
        </attribute-tag1>
  </entity-tag1>
```

2.3 Message Definition

2.3.1 Introduction

This specification defines information content for four messages that can be sent between a server and a client. Each message is represented as an XML element. The contents of each message are also represented by XML elements. When necessary the immediate sub-elements of each message are called root elements to distinguish them from the other elements.

2.3.2 Name Space

The following name space shall be defined as an XML attribute of each message element.

```
xmlns:ceb=“ urn:iso10303-28:ceb”
```

The following error message shall be used if the client sends the server a message that does not define this name space.

Error:

Code:	1	Message:	Undefined name space

2.3.3 STEP-QUERY message

The STEP-QUERY message shall be sent from the client to the server. The following element shall enclose a STEP-QUERY message.

```
<ceb:STEP-QUERY xmlns:ceb=“ urn:iso10303-28:ceb” >
 <binding-tag>
 <configure-tag>
 <pick-tag>
</ceb:STEP-QUERY>
```

A STEP-QUERY message shall contain configure, pick and binding tags only. The following error message shall be used if the client sends the server a STEP-QURY containing a tag that is not a binding, configure or pick tag.

Error:

Code:	2	Message:	Tag not allowed in STEP-QUERY

2.3.4 STEP-DICTIONARY message

The STEP-DICTIONARY message is a special case of the STEP-QUERY message in which the message is addressed to the SDAI Dictionary that defines the data set instead of the server itself.

```
<ceb:STEP-DICTIONARY xmlns:ceb=“ urn:iso10303-28:ceb” >
…
</ceb:STEP-DICTIONARY>
```

2.3.5 STEP-RESULT message

The STEP-RESULT message shall be sent from the server to the client. The server shall use a STEP-RESULT message to describe values in the server. The following element shall enclose a STEP-RESULT message.

```
<ceb:STEP-RESULT xmlns:ceb=“ urn:iso10303-28:ceb” >
 <entity-tag>
 <rule-tag>
 <view-tag>
</ceb:STEP-RESULT>
```

A STEP-RESULT message shall contain entity, rule and view tags only.

2.3.6 STEP-UPDATE message

The STEP-UPDATE message shall be sent from the client to the server. The following element shall enclose a STEP-UPDATE message.

```
<ceb:STEP-UPDATE xmlns:ceb=” urn:iso10303-28:ceb” >
…
</ceb:STEP-UPDATE>
```

A STEP-UPDATE message may contain any type of element. However, the server shall process the elements that represent delete, new and update tags only.

2.3.7 STEP-ERROR message

The STEP-ERROR message shall be sent from the server to the client when the server is unable to process a request made by the client. The following element shall enclose a STEP-ERROR message.

```
<ceb:STEP-ERROR xmlns:ceb=" urn:iso10303-28:ceb" >
 <name>PCDATA</name>
 <identifier>PCDATA</identifier>
 <code>PCDATA</code>
 <descripton>PCDATA</description>
</ceb:STEP-ERROR>
```

A STEP-ERROR message shall contain four sub-elements defined as PCDATA. The first sub-element shall be called "name" and contain the name of the tag in the original message that caused the error. The second sub-element shall be called "identifier" and contain the instance identifier of the tag that caused the error if the tag has an instance identifier and no value otherwise. The third sub-element shall be called "code" and define a code for the error as defined in this specification. The fourth sub-element shall be called "description" and contain a textual description of the error as defined in this specification.

2.4 Message Content

2.4.1 Tag Names

An argument-tag name shall be any name followed by a "-" and a non-empty string of digits (0-9).

An attribute-tag name shall be the same as the EXPRESS name of the attribute with all characters converted to lower case.

A binding-tag name shall be the same as the EXPRESS-X name of the view and the case of the characters shall be ignored.

A configure-tag name shall be the same as the name of the entity-tag, view-tag or rule-tag that it configures and the case of the characters shall be ignored..

A delete-tag name shall be the same as the name of an entity-tag and the case of the characters shall be ignored.

An entity-tag name shall be the same as the EXPRESS name of the entity with the first character converted to upper case and the subsequent characters converted to lower case.

A keyword-tag name shall be one the fixed values described in Section ?.?.?.? view and the case of the characters shall be ignored.

A label-tag name shall be the same as the EXPRESS name of the type with all characters converted to lower case unless the type is BINARY, BOOLEAN, LOGICAL, INTEGER, NUMBER, REAL or STRING in which case all the characters shall be converted to upper case.

A new-tag name shall be the same as the name of an entity-tag and the case of the characters shall be ignored.

A pick-tag name shall be the same as the name of the entity-tag that it picks and the case of the characters shall be ignored..

A view-tag name shall be the same as the EXPRESS-X name of the view with all the characters converted to upper case.

A rule-tag name shall be the same as the name of the global rule with all the characters converted to lower case.

A truncated-tag name shall be the same as the name of the tag it represents.

An update-tag name shall be the same as the name of an entity-tag and the case of the characters shall be ignored.

2.4.2 Instance Tag Properties

2.4.2.1 Introduction

Instance tag properties shall be XML attributes of the XML element that represents the tag.

The following error message shall be used if the client sends the server a message containing an instance tag with an XML attribute belonging to the CEB name space but defining an unknown property.

Error:

Code:	3	Message:	Unknown instance tag property

2.4.2.2 Copies Property

The Copies property shall describe the number of other entity-tags in the same message that have the same ceb:id and contain one or more of the attributes described in this entity-tag. The copies property shall be written as an XML attribute with the name ceb:copies. The default value of ceb:copies is “1”. The ceb:copies attribute is not required if it has this default value.

> NOTE – The copies attribute warns the client that the given number of other XML elements in the message contain duplicates of values in this element.

2.4.2.3 Partial Property

The partial property shall indicate that an entity-tag only contains a partial description of this entity instance. If the value of ceb:partial is true then the instance is an AND/OR entity and the entity tag only contains that part of the entity instance necessary to describe the attribute of the entity instance parent that contain this instance.

The partial property shall be written as an XML attribute with the name ceb:partial. The default value of ceb:partial is “false”. The ceb:partial attribute is not required if it has this default value.

2.4.2.4 Reference Property

The references property shall indicate that there are unknown entity instances in the server that reference the entity instance described by this tag. These entity instances are unknown because they are not represented by an entity tag in this message.

The name of the XML attribute representing this property shall be ceb:references. The default value for this property is “false”. The property is not required if it has this default value.

> NOTE – If the value of ceb:references is true then an update to the values in the entity-tag may cause a side effect in the database.

2.4.2.5 Status Property

The status property shall be used to show the status of a binding-tag, configure-tag, entity-tag, label-tag, truncated-tag or view-tag. The name of the XML attribute representing the property shall be “ceb:status” and its value shall be one of the strings described below.

— **truncated**. A status of “truncated” shall show that this is a truncated-tag.

> NOTE – Entity, view and binding tags may be truncated to meet the requirements of a message, to break recursive cycles when entities, bindings or views contain themselves as attributes or as a means to reference an entity, binding or view whose value is already known by the client or server. In all cases the value and meaning of the tag can be deduced from other data in the XML message or the server.

— **new**. A status of “new” shall show that is a new-tag.

— **delete**. A status of “delete” shall show that this is a delete-tag.

— **update**. A status of “update” shall show that this is an update-tag.

— **entity**. A status of “entity” shall show that this is an entity-tag.

— **binding**. A status of “binding” shall show that this is a binding-tag.

— **view**. A status of “view” shall show that this is a view-tag.

— **configure**. A status of “configure” shall show that this is a configure-tag.

— **pick**. A status of “pick” shall show that this is a configure-tag.

— **rule**. A status of “global rule” shall show that this is a rule-tag.

If a tag describes a root element and it does not have a status value, then the default status of that tag shall be “pick” for a STEP-QUERY message and “entity” for STEP-RESULT and STEP-UPDATE messages. If a tag describes a NON-root element and it does not have a status value then the default status of that tag shall be “entity”.

The following error message shall be used if the client sends the server a message containing a tag with a status value other than those listed above

Error:
Code: 4 Message: Unknown status value

2.4.2.6 Instance Identifier Property

The name of the XML attribute representing the instance identifier property shall be ceb:id.

An EXPRESS entity instance belonging to the server shall be written into the message content zero, once or many times depending on the requirements of the message. Each copy shall have the same instance identifier value. This instance identifier shall begin with the character “I”. The identifier shall be written as an XML attribute of each entity-tag or truncated-tag representing the entity instance.

An entity-tag, binding-tag, truncated-tag, configure-tag or pick-tag constructed by the client may be given an instance identifier property. If two client constructed tags have the same instance identifier value then they shall represent the same instance. The value of the instance identifier shall not begin with the character “I”.

An identifier value of “-“ shall be used to represent an unknown random value that is unique to this instance and cannot be referenced or duplicated by any other tag including those that also have a identifier value of “-“.

The following error message shall be used if the client sends the server a message containing a tag with an unknown instance identifier.

Error:
Code: 5 Message: Unknown instance identifier

2.4.2.7 Identifier Space Property

The name of the XML attribute representing the identifier space property shall be ceb:is.

An identifier space shall be defined for entity-tags and binding-tags that represent new entities to disambiguate the identifiers of elements created by multiple applications.

A ceb:is shall exist for the scope of the entity-tag or binding-tag that declares the name space. If a nested tag declares an identifier space then that nested space shall be used for the scope of the nested tag and the containing scope shall resume at the end of the nested tag. The initial identifier space for a message shall be given the name “default”.

If two instance identifiers have the same value and that value is not “-“ and they belong to the same identifier space then the tags containing those instance identifiers shall represent the same item in the server or client.

2.4.3 Discriminator Properties

2.4.3.1 Introduction

Discriminator tag properties shall be XML attributes of the XML element that represents the tag.

The following error message shall be used if the client sends the server a message containing a discriminator tag with an XML attribute belonging to the CEB name space but defining an unknown property.

Error:			
Code:	6	Message:	Unknown discriminator tag property

2.4.3.2 Type Property

The type property shall be used to show the type of an attribute-tag, or argument-tag. The name of the XML attribute representing the property shall be “ceb:type” and its value shall be one of the strings described below.

— **explicit**. A type value of “explicit” shall indicate that this is an EXPRESS or EXPRESS-X explicit attribute.

— **optional**. A type value of “optional” shall indicate that this is an EXPRESS or EXPRESS-X explicit attribute with an optional value.

— **derived**. A type value of “derived” shall indicate that this is an EXPRESS derived attribute value.

— **inverse**. A type value of “inverse” shall indicate that this is an EXPRESS inverse attribute value.

— **unique**. A type value of “unique rule” shall indicate that this is an EXPRESS unique rule value.

— **where**. A type value of “where rule” shall indicate that this is an EXPRESS where rule value.

— **defined type**. A type value of “type rule” shall indicate that this is an EXPRESS defined type rule value.

— **update**. A type value of “update” shall indicate that this is an EXPRESS explicit attribute whose value has been modified by the client.

— **argument**. A type value of “argument” shall indicate that this is an argument tag for an EXPRESS-X binding.

If a discriminator tag belongs to an entity-tag and it does not have a type value, then the default type of that tag shall be “explicit”. If a discriminator tag belongs to a binding-tag and it does not have a type value, then the default type of that tag shall be “argument”. If a discriminator tag belongs to a view-tag and it does not have a type value, then the default type of that tag shall be “explicit”.

The following error message shall be used if the client sends the server a message containing a tag with a type value other than those listed above

Error:			
Code:	7	Message:	Unknown type value

2.4.4 Value Tags

2.4.4.1 Introduction

Value tags shall be used to describe values for the attributes of entity and view tags and the arguments of binding tags.

The following error message shall be used if the client sends the server a message containing a value tag with an XML attribute belonging to the CEB name space but defining an unknown property. If the tag is also an instance tag then this message shall take precedence.

Error:			
Code:	8	Message:	Unknown value tag property

2.4.4.2 Null Values

The following XML element shall be used to indicate that a value is null.

```
<no-value>
```

2.4.4.3 Basic Values

If the value of the attribute is defined by a standard type, or a defined type of a standard type or an enumerated type then it shall be written as an XML element using the formatting rules defined for the Object Serialization Early Binding (OSEB).

The following error message shall be used if the client sends the server a value tag containing an illegal basic value.

Error:			
Code:	9	Message:	Illegal syntax for basic value

2.4.4.4 Nested Entity Values

If the value of the attribute is defined by a nested entity instance then:

— If the message does not allow the nested entity instance to be written then it shall be written as a truncated tag.
— If the message allows the nested entity instance to be written but another XML element for this instance is currently being written (i.e the end-tag has not yet been inserted into the XML) then this nested instance shall be written as a truncated tag.
— If the message allows the nested entity instance to be written and a copy of the entity instance is not currently being written then it shall be written according to the rules defined in Section 2.4.5 with the current attribute as the controlling attribute.

2.4.4.5 Data Structures

The position property shall be used to disambiguate the location of a nested instance or primitive value in an attribute with an aggregate or Select data structure. The data structure property is called ceb:pos and it shall be represented as an XML attribute of the value tag.

The description of the position property begins with a left square bracket "[" and ends with a right square bracket "]". The content of the description is made up of a sequence of positional identifiers separated by commas ",". For each value in the attribute, a descent shall be made from the data structure of the attribute to the value. For each aggregate or Select encountered in the descent a coordinate shall be added to the description of the position in left to right order. The process shall terminate for this value if the aggregate or select is null.

When an aggregate is encountered in the descent, an integer index shall be generated that corresponds to the index that would be used to access that location in an EXPRESS expression.

When a SELECT is encountered a positional identifier with a three part name shall be generated.

— A string chosen to represent EXPRESS SELECT definition.
— The character "-"
— An integer that corresponds to the position of this case in the SELECT definition in the EXPRESS . The first position shall be "1".

Note – This procedure allows the positional identifier for a SELECT to be defined using a short name.

For each EXPRESS SELECT definition an element shall be added to the end of the XML. This element shall be tagged with the name “Spath-“ and the unique name generated for the SELECT by the procedure above. The content of the tag shall be a list of strings separated by space characters. The first name shall be the EXPRESS name of the SELECT. The following names shall be the EXPRESS name of each of the cases in the SELECT in the order given in the EXPRESS.

The rightmost positional index that is a Select index shall be deleted if, after the deletion, all the paths allowed by the Select can be distinguished using the entity-tags and label-tags of the values and the remaining positional indexes. If the rightmost positional Select index is deleted then this procedure shall be repeated for the new rightmost positional Select index. The position property itself shall be deleted if this procedure results in the deletion of all the positional indexes for this property. The Spath- element shall be deleted if no positional Select index refers to this path after this procedure has been applied.

The following error message shall be used if the client sends the server a message containing a value tag with an invalid position.

Error:

Code:	10	Message:	Invalid position

2.4.5 Entity-Tags

2.4.5.1 Introduction

An entity-tag is written by the server into a STEP-RESULT message to describe information about an entity instance in the server.

2.4.5.2 Complex Instances

The following procedure shall be used to create an entity definition for AND/OR instances.

If the AND/OR instance is nested then the type of the attribute containing the nested instance shall control which attributes are included in the entity instance. This attribute shall be called the controlling attribute.

— Create a list of all the entity types in the complex.
— If a controlling attribute exists, then eliminate the entity types that are not subtypes of the entity type of the controlling attribute.
— Eliminate entity types that are super-types of other entity types in the list.
— Generate names for the entity types as described in Section 2.4.1
— Sort the names in alphabetic order.

The synthetic entity shall have the types in the sorted list as its super-types in the order of the sorting. A name shall be created for the synthetic entity as follows:

— Concatenate the names of the types and separate each with the "-" character.

The complex instance shall be treated as an ordinary entity instance with the synthetic entity as its definition and the entity types given in the sorted order as its super-types.

2.4.5.3 Entity-Tag Properties

The XML element representing an entity-tag shall have an instance identifier property if the entity-tag represents an instance belonging to the server.

The XML element representing an entity-tag may have copy, references and partial properties.

If an entity-tag is both an instance-tag and a value-tag then it may have a position property.

> Note - The convention followed in the examples of this document is to given each entity instance an identifier containing the entity-ID number of a corresponding example in 10303-21 format.

2.4.5.4 Entity-Tag Attributes

The attributes of the entity instance shall be written as XML elements belonging to the entity-tag representing the instance.

The attributes of super-types of an entity instance shall be included as attributes of the entity instance using the following algorithm:

— All inherited attributes shall appear sequentially prior to the attributes of the entity instance.

— The attributes of a supertype entity shall be inherited in the order they appear in the supertype entity itself.

— If the supertype entity is itself a subtype of another entity, then the attributes of the higher supertype entity shall be inherited first.

— When multiple supertype entities are specified, the attributes of supertype entities shall be processed in the order specified in the SUBTYPE OF expression.

This procedure may result in a supertype entity being referenced more than once. In this case all references after the first one shall be ignored.

After this procedure has been calculated then the attributes shall be sorted into the categories: explicit attributes, inverse attributes, derived attributes, unique rules, where rules and defined type rules. The order determined by the procedure shall be maintained within each category.

This procedure may result in an attribute being redefined. In this case the new definition shall be applied to the original definition of the attribute.

This procedure shall be applied to derived and inverse attributes, unique and where rules, and defined type rules if the message requires their values to be computed and written.

If a Derived attribute computes a new entity instance value then it shall not have an identifier property.

A labeled Where or Unique rule shall be written as an attribute with the type Logical and the name of the label shall be the name of the tag. Unlabeled rules shall not be written.

If the message requires defined type rules of the underlying values in an entity instance to be written then they shall be computed and listed in the same order as the values that contain the rules. Unlabeled defined type rules shall not be written.

If two attributes have the same name and one attribute is not defined in a sub-type of the controlling attribute, then the name of that attribute shall be qualified by preceding it with the name of the last entity to redefine this attribute and a period symbol ".". If two attributes have that same name and both are defined by sub-types of the controlling attribute, then the name of each shall be qualified by preceding it with the name of the last entity to redefine this attribute and a period symbol ".".

> NOTE – The name of an attribute will only be qualified if 1) two sub-types of the root type contain attributes with the same name; 2) the root type does not define an attribute with this name; 3) both sub-types are the super-type of a named sub-type of the root.

2.4.6 Truncated Tags

Truncated tags shall be written by the client or server. A truncated tag shall contain instance identifier and status properties only.

2.4.7 Updated Tags

Update tags shall be written by the client. A tag is an update tag if it has the form of an entity-tag and a status value of "update". An update tag must contain at least one attribute with a type property value of "update".

Each attribute-tag in the update tag with a type property value of "update" must correspond to an explicit attribute of the corresponding entity-tag.

After evaluation of any binding tags, the EXPRESS representation of the value(s) in the attribute tag must be type compatible with EXPRESS type of the attribute.

If the attribute contains multiple values then only value tags with a status value of “new” or “delete” shall be processed.

The following error message shall be used if the client sends the server a message containing an update tag with no updated attributes.

Error:
Code: 11 Message: No new values in update-tag

The following error message shall be used if the client sends the server a message containing an update tag for an entity instance that is not in the server.

Error:
Code: 12 Message: Unknown instance in update-tag

The following error message shall be used if the client sends the server a message containing an updated attribute that does not belong to the entity instance.

Error:
Code: 13 Message: Unknown attribute in update-tag

The following error message shall be used if the client sends the server a message containing an updated attribute value that is not type compatible with the type of the attribute.

Error:
Code: 14 Message: Unknown type in update-tag

2.4.8 New Tags

New tags shall be written by the client. A tag is a new tag if it has the form of an entity-tag and a status value of “new”.

Each attribute-tag in the new tag must correspond to an explicit attribute of the corresponding entity-tag.

After evaluation of any binding tags, the EXPRESS representation of the value(s) in the attribute tag must be type compatible with EXPRESS type of the attribute.

The following error message shall be used if the client sends the server a message containing a new tag for an entity instance that is already in the server.

Error:
Code: 15 Message: New-tag instance already exists

The following error message shall be used if the client sends the server a message containing a new tag for an entity instance with an unknown type.

Error:
Code: 16 Message: New-tag has unknown type

The following error message shall be used if the client sends the server a message containing an attribute that does not belong to the entity instance.

Error:
Code: 17 Message: Unknown attribute in new-tag

The following error message shall be used if the client sends the server a message containing an updated attribute value that is not type compatible with the type of the attribute.

Error:
Code: 18 Message: Unknown type in new-tag

2.4.9 Delete Tags

Delete tags shall be written by the client. A tag is a delete tag if it has the form of an entity-tag and a status value of “delete”.

A delete tag must have an instance identifier that identifies an instance in the server.

The following error message shall be used if the client sends the server a message containing a delete tag for an entity instance that is not in the server.

Error:
Code: 19 Message: Unknown instance in delete-tag

2.4.10 Rule-Tags

Rule tags shall be written by the server.

2.4.11 Binding-Tags

An binding-tag is written by the client to select information belonging to the server by value.

A name for each binding argument shall be generated as follows:

— The argument for the first parameter in the EXPRESS-X binding shall be any name followed by the discriminator “-1”.
— The argument for the second parameter in the EXPRESS-X binding shall be any name followed by the discriminator “-2”
— And so on.

The arguments do not need to be given in order. If no argument is given for an EXPRESS-X parameter then this value shall be unbound in the binding call.

> Note – The CEB describes how to bind values to EXPRESS-X Views only. EXPRESS-X Maps define entity instances that can be bound using the rules defined for entity instance.

The following error message shall be used if the client sends the server a message containing a binding tag for an unknown binding.

Error:

Code:	20	Message:	Unknown binding-tag

The following error message shall be used if the client sends the server a message containing an argument tag with an unknown discriminator.

Error:

Code:	21	Message:	Unknown argument-tag

The following error message shall be sent if an execution error occurs during the evaluation of a binding argument.

Error:

Code:	22	Message:	Error evaluating binding argument

2.4.12 View-Tags

A view-tag is written by the server to describe the result of an EXPRESS-X SELECT view.

A view tag shall be encoded by computing an entity instance value that is equivalent to the view value using the rules defined in Appendix A of 10303-14. The view value shall then be written to the XML using the rules defined in this specification for entity instances with the following additions

> Note – Instances computed as the result of Return views shall be encoded using the rules defined for the value computed by that view. Return views can compute values that are primitives, entity instances or view instances.

2.4.13 Configure-Tags

An configure-tag is written by the client to control the formatting of information for a STEP-RESULT message.

A configure tag without an instance identifier is applied to every entity, view or rule tag with the same name as the configure tag.

A configure tag with an instance identifier is applied to the instance identified by that identifier.

The configure tag with the name STEP-Header is applied to every entity, view and rule tag.

A configure tag must be a root element.

A configure tag contains one or more keyword tags. The following keywords have been assigned meanings. The default values for these parameters shall be decided by the server.

<Nesting/>	Nested instance shall be encoded.
<No-Nesting/>	Nested instances shall be truncated.
<Duplicates/>	Second and subsequent instances shall be encoded.
<No-Duplicates/>	Second and subsequent instances shall be truncated.
<Copies/>	The copies property shall be written.
<No-Copies/>	The copies property shall be omitted.
<References/>	The references property shall be written.
<No-References/>	The references property shall be omitted.
<Partial/>	The partial property shall be written.
<No-Partial/>	The partial property shall be omitted.
<No-Derived/>	Derived attributes shall not be evaluated and encoded.
<Derived/>	Derived attributes shall be evaluated and encoded.
<No-Inverse/>	Inverse attributes shall not be evaluated and encoded.
<Inverse/>	Inverse attributes shall not be evaluated and encoded.
<No-Where-rule/>	Where rules shall not be evaluated and encoded.
<Where-rule/>	Where rules shall be evaluated and encoded.
<No-Unique-rule/>	Unique rules shall not be evaluated and encoded.
<Unique-rule/>	Unique rules shall not be evaluated and encoded.
<No-Type-rule/>	Defined type rules shall not be evaluated and encoded.
<Type-rule/>	Defined type rules shall be evaluated and encoded.
<No-Global-rule/>	This global rule shall not be evaluated and encoded.
<Global-rule/>	This global rules shall be evaluated and encoded.
<All-root/>	Every type in a root instance shall be encoded.
<Exact-root/>	Only the types in the root instance that match the pick tag shall be encoded.

The following error message shall be used if the client sends the server a message containing a configure tag that is not a root element.

Error:
Code: 23 Message: Configure-tag must be a root element

The following error message shall be used if the client sends the server a message containing a configure tag that does not identify an entity or view instance.

Error:
Code: 24 Message: Configure-tag not for an entity or view

The following error message shall be used if the client sends the server a message containing a configure tag with an unknown keyword.

Error:
Code: 25 Message: Unknown keyword in configure-tag.

2.4.14 Pick-Tags

A pick-tag is written by the client to identify entity instances in the server. A pick tag shall identify an instance in the server using the instance identifier property, or it shall identify a type of entity. If a pick-tag has the name no-value then it must have an instance identifier and it shall identify the corresponding entity.

If a pick-tag identifies a type of entity then it may contain the following keywords.

<Exact/>	The entity must match this type only.
<AND-OR/>	At least one type in the complex instance match this type.
<Self/>	The match must be exact.
<IS-A/>	The match must be a sub-type of the pick type.

The following error message shall be used if the client sends the server a message containing a pick tag that is not a root element.

Error:
Code: 26 Message: Pick-tag must be a root element

The following error message shall be used if the client sends the server a message containing a pick tag that does not identify an entity instance.

Error:
Code: 27 Message: Pick-tag not for an entity

The following error message shall be used if the client sends the server a message containing a pick tag with an unknown keyword.

Error:
Code: 28 Message: Unknown keyword in pick-tag.

2.5 Message Interpretation

2.5.1 Query Message

The following actions shall be performed by the server.

If a pick-tag has an instance identifier then a CEB XML encoding of the corresponding entity shall be added to the Result Message.

If a pick-tag has no instance identifier then a CEB XML encoding of every instance identified by the pick-tag shall be added to the Result Message.

The binding described by a binding-tag shall be evaluated by the server and a CEB XML encoding of each result shall be added to the Result Message.

A configure tag is applied to all subsequent instances of the identified tag.

Elements shall be added to the Result message in the order requested by the Export Message. This procedure may result in an entity instance being encoded more than once.

If an entity instance is required to be included in the Result message because it describes a nested value of another instance, but that instance cannot include as a nested instance because of a no-nesting configuration, then it shall be added to a list of discarded instances. After all the instances requested by the Export message have been encoded, a discarded instance that has not yet been encoded shall be picked at random and encoded into XML. This procedure shall be repeated until all the discarded instances have been encoded. The full value of the discarded instances shall be encoded.

If an Export message contains no entity or binding tags, then an entity instance shall be picked at random and encoded into XML. This procedure shall be repeated until all the instances in the server have been encoded. For this procedure an entity instance shall not be considered encoded if has been partially encoded to meet the requirements of a controlling attribute in a nested instance. The full value of an instance shall be encoded when it is picked at random.

2.5.2 Dictionary Message

The server shall interpret the message as a STEP-QUERY message to the SDAI dictionary that defines its data.

2.5.3 Update Message

The server shall search the message for new, delete and update tags.

The new, delete and update tags shall be interpreted in depth first order.

Entity instances corresponding to the new tags shall be added to the server.

Entity instances corresponding to delete tags shall be marked for deletion. After all the other instructions in the message have been interpreted, each marked entity shall be visited and it shall be deleted if it will not be referenced by any other entity after all the deletions have been completed.

If an updated attribute in an update-tag has a singleton data type then the value in the server shall be replaced with the value given in the XML.

If an updated attribute in an update tag has a multi-value data type then

— If the value is defined by a new-tag and the tag has a position property, then this value shall be inserted into this location of the attribute and replace the old value at that location.

— If the value is defined by a new-tag and the tag does not have a position property and the aggregate is not an array, then the following locations shall be tried for this item in the given order

  — if the value immediately before this item was inserted then immediately after that item in the same aggregate.
  — if the value immediately after this item was inserted then immediately before this item in the same aggregate.
  — if no item has been inserted and this is the first item and the appropriate aggregate for this type of value is empty then as the first item in a new aggregate.

— If the value is defined by a delete-tag and the tag has a position property and this property identifies a location then the value at the identified location shall be deleted and if the aggregate is an array that slot of the array shall be set to <no-value>.

— If the value is defined by a delete tag and the value has an identity and no position property is given then every occurrence of the value in the data structure shall be deleted and for each occurrence in an array the corresponding slot shall be set to <no-value>.

If the interpretation of any new, delete or update tags requires a value described by an binding-tag then that binding shall be computed by the server.

A SOAP-Substitutions (Informative)

A.1 Introduction

The following text substitutions can be used to make a message more closely conform to the conventions of the SOAP 1.0 specification. A client may request these conventions using the value “SOAP 1.0” in a configure tag.

A.2 SOAP id

The following XML attributes in an entity-tag XML element

```
id=”value”
```

shall be equivalent to the following XML attributes in the same XML element.

```
ceb:id=”value” ceb:copies=”1”
```

A.3 SOAP href

The following XML attributes in an entity-tag or label-tag XML element

```
<entity-tag props href=”value”>
```

shall be equivalent to the following XML attributes in the same XML element.

```
<entity-tag props ceb:id=”value” ceb:status=”truncated”>
```

The following XML attributes in an attribute-tag element

```
<attribute-tag entity-props props href=”value”>
```

shall be equivalent to the following nested XML element where name is the name of the XML element referenced by href and entity-props is any XML attribute with the name ceb:references, ceb:copies, ceb:is or ceb:partial.

```
<attribute-tag props>
     <name ceb:id=”value” ceb:status=”truncated”
          entity-props/>
```

A.4 SOAP Attributes with Fixed type

If the XML element for an EXPRESS attribute can contain a single nested entity instance of one type only and the entity-tag for that nested instance is not the first nested tag inside the XML element for the attribute then the following sequence

```
<attribute-tag props entity-props>
     <next-tag>
```

```
          </last-tag>
     </attribute-tag>
```

shall be equivalent to the following tag sequence where attribute-tag is the XML element for an EXPRESS attribute containing an entity of the type entity-tag and entity-prop is any XML attribute with the name id, ceb:references, ceb:copies, ceb:is or ceb:partial.

```
     <attribute-tag props>
          <entity-tag entity-props>
               <next-tag>

               </last-tag>
          </entity-tag>
     </attribute-tag>
```

A.5 SOAP Attributes with xsi:type

If the XML element for an EXPRESS attribute contains an "xsi:type" attribute then that attribute shall be assumed to define an entity-tag or label-tag for the underlying value and the following

```
     <attribute-tag xsi:type="value" props
                              entity_or_label-props>
          <next-tag>

          </last-tag>
     </attribute-tag>
```

shall be equivalent to the following tag sequence where entity_or_label-prop is any XML attribute with the name id, ceb:references, ceb:copies, ceb:is or ceb:partial.

```
     <attribute-tag props>
          <value entity_or_label-props>
               <next-tag>

               </last-tag>
          </value>
     </attribute-tag>
```

A.6 SOAP-ENC:pos

The following XML attributes in an entity-tag or label-tag XML element

```
     SOAP-ENC:pos=["value"]
```

shall be equivalent to the following XML attributes in the same XML element.

```
     ceb:pos=["value"]
```

A.7 SOAP href=”null”

The following XML attribute value in an attribute-tag

```
<attribute-tag props href=“null”/>
```

shall be equivalent to the following XML attributes in the same XML element.

```
<attribute-tag props>
     <no-value>
</attribute-tag>
```

B Table of Error Messages (Informative)

Code:	1	Message:	Undefined name space
Code:	2	Message:	Tag not allowed in STEP-QUERY
Code:	3	Message:	Unknown instance tag property
Code:	4	Message:	Unknown status value
Code:	5	Message:	Unknown instance identifier
Code:	6	Message:	Unknown discriminator tag property
Code:	7	Message:	Unknown type value
Code:	8	Message:	Unknown value tag property
Code:	9	Message:	Illegal syntax for basic value
Code:	10	Message:	Invalid position
Code:	11	Message:	No new values in update-tag
Code:	12	Message:	Unknown instance in update-tag
Code:	13	Message:	Unknown attribute in update-tag
Code:	14	Message:	Unknown type in update-tag
Code:	15	Message:	New-tag instance already exists
Code:	16	Message:	New-tag has unknown type
Code:	17	Message:	Unknown attribute in new-tag
Code:	18	Message:	Unknown type in new-tag
Code:	19	Message:	Unknown instance in delete-tag
Code:	20	Message:	Unknown binding-tag
Code:	21	Message:	Unknown argument-tag
Code:	22	Message:	Error evaluating binding argument
Code:	23	Message:	Configure-tag must be a root element
Code:	24	Message:	Configure-tag not for an entity or view
Code:	25	Message:	Unknown keyword in configure-tag.
Code:	26	Message:	Pick-tag must be a root element
Code:	27	Message:	Pick-tag not for an entity
Code:	28	Message:	Unknown keyword in pick-tag.

C Examples (Informative)

EXAMPLE 1 - The following is an example of how of EXPRESS instances are written to XML for an Export message.

```
ENTITY parent;
first : child;
second : child;
END_ENTITY;

ENTITY child;
name : STRING;
END_ENTITY;
```

The Part 21 data is:

```
#10 = PARENT (#20, #20);
#20 = CHILD (‘Daphne’);
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=” urn:iso10303-28:ceb” >
      <Parent/>
</ceb:STEP-QUERY>
```

The Result message is:

```
<ceb:STEP-RESULT xmlns:ceb=” urn:iso10303-28:ceb” >
   <Parent ceb:id=” I-10” >
      <first>
            <Child ceb:id=” I-20”  ceb:copies=” 2” >
                  <name>Daphne<name/>
            </Child>
      </first>
      <second>
            <Child ceb:id=” I-20”  ceb:copies=” 2” >
                  <name>Daphne<name/>
            </Child>
      </second>
   </Parent>
</ceb:STEP-RESULT>
```

EXAMPLE 2 - The following is an example of how EXPRESS instances are written to XML for a Export message configured to never write duplicates. The data in this example is same as that of Example 1.

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=” urn:iso10303-28:ceb” >
   <STEP-HEADER>
      <No-Duplicates/>
   </STEP-HEADER>
   <Parent/>
</ceb:STEP-QUERY>
```

The Result message is:

```
<ceb:STEP-RESULT xmlns:ceb=” urn:iso10303-28:ceb” >
   <Parent ceb:id=” I-10” >
      <first>
            <Child ceb:id=” I-20” >
                  <name>Daphne<name/>
            </Child>
      </first>
      <second>
            <Child ceb:id=" I-20"  ceb:status=" truncated" />
      </second>
   </Parent>
</ceb:STEP-RESULT>
```

EXAMPLE 3 - The following is an example of how of EXPRESS instances are written to XML for an Export message configured to never nest entity instances. The data in this example is the same as that of Example 1. The example uses SOAP-Substitutions as described in Section 2.9.

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=" urn:iso10303-28:ceb" >
   <STEP-Header>
      <No-Nesting/>
      <SOAP-Substitutions/>
   </STEP-Header>
   <Parent/>
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT xmlns:ceb=" urn:iso10303-28:ceb" >
   <Parent id=" I-10" >
      <first href=" I-20" />
      <second href=" I-20" />
   </Parent>
   <Child id=" I-20" >
      <name>Daphne<name/>
   </Child>
</ceb:STEP-RESULT>
```

EXAMPLE 4 – In this example a teacher is also a student and is written out in two entity tags. The first tag describes the student role. The second tag describes the teacher role. The ceb:partial attribute is set in the two tags to show that only part of the data is being written in each tag. The ceb:copies attribute is set to 2 because the name of the person is repeated in the two tags.

```
ENTITY person;
name : STRING;
END_ENTITY;

ENTITY student SUBTYPE OF (person);
GPA : REAL;
END_ENTITY;

ENTITY teacher SUBTYPE OF (person);
teaches : SET OF student;
END_ENTITY;
```

The Part 21 data is:

```
#10 = TEACHER ('David', (#20, #30));
#20 = STUDENT ('Joe', 3.0);
#30 = ((PERSON ('Ann'),
      STUDENT (4.0),
      TEACHER((#20)));
#40 = STUDENT ('Arthur', 3.0);
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=" urn:iso10303-28:ceb" >
```

```
      <Teacher/>
</ceb:STEP-QUERY>
```

The Result message is:

```
<ceb:STEP-RESULT xmlns:ceb=" urn:iso10303-28:ceb" >
   <Teacher ceb:id=" I-10" >
      <name>David</name>
      <teaches>
            <Student ceb:id=" I-20"  ceb:pos=" [0]" >
                  <name>Joe</name>
                  <GPA>3.0</GPA>
            </Student>
            <Student ceb:id=" I-30" ceb:pos=" [1]"
                        ceb:partial=" true"  ceb:copies=" 2" >
                  <name>Ann</name>
                  <GPA>4.0</GPA>
            </Student>
      </teaches>
   </Teacher>
   <Teacher ceb:id=" I-30"  ceb:partial=" true"  ceb:copies=" 2" >
      <name>Ann</name>
      <teaches>
            <Student ceb:id=" I-20"  ceb:pos=" [0]" >
                  <name>Joe</name>
                  <GPA>3.0</GPA>
            </Student>
      </teaches>
   </Teacher>
</ceb:STEP-RESULT>
```

EXAMPLE 5 – The following example contains STEP data defined by ISO 10303-203:1994. Product_definition_formation_with_specified_source has the value of ceb:references set indicating that this entity instance is referenced by instances other than the product_definition instance shown in the XML.

The relevant fragment of the Part 21 data is:

```
#421420 = PRODUCT_DEFINITION ('DESIGN', '', #670750, #670720);
#670700 = APPLICATION_CONTEXT ('CONFIGURATION CONTROLLED 3D
DESIGNS OF MECHANICAL PARTS AND ASSEMBLIES');
#670720 = PRODUCT_DEFINITION_CONTEXT ('', #670700, 'DESIGN');
#670730 = MECHANICAL_CONTEXT ('', #670700, 'MECHANICAL');
#670740 = PRODUCT ('TIRE', 'TIRE', 'NOT SPECIFIED', (#670730));
#670750 = PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE
('3', 'LAST_VERSION', #670740, .MADE.);
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=" urn:iso10303-28:ceb" >
      <Product_definition ceb:id=" I-421420" />
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT xmlns:ceb=" urn:iso10303-28:ceb" >
<Product_definition ceb:id=" I-421420>
   <id>design</id>
   <description></description>
   <formation>
      <Product_definition_formation_with_specified_source
         ceb:id =”I-670750” ceb:references=”true”>
         <id>3</id>
         <description>LAST_VERSION</description>
```

```
          <make_or_buy>made</make_or_buy>
          <of_product>
             <Product ceb:id=" I-670740">
                <id>TIRE</id>
                <name>TIRE</name>
                <description>NOT SPECIFIED</description>
                <frame_of_reference>
                   <Mechanical_context ceb:id="I-670730">
                      <name></name>
                      <discipline_type>MECHANICAL</discipline_type>
                      <frame_of_reference>
                         <Application_context ceb:id=" I-607000"
                         ceb:copies=" 2" ceb:references=" true"
                               <text>CONFIGURATION CONTROLLED 3D
                         DESIGNS OF MECHANICAL PARTS AND
                         ASSEMBLIES</text>
                         </Application_context>
                      </frame_of_reference>
                   </Mechanical_context>
                </frame_of_reference>
             </Product>
          </of_product>
       </Product_definition_formation_with_specified_source>
    </formation>
    <frame_of_reference>
       <Product_definition_context ceb:id="I-670720"
          <name></name>
          <life_cycle_stage>DESIGN</life_cycle_stage>
          <frame_of_reference>
             <Application_context ceb:id=" I-607000"
             ceb:copies=" 2" ceb:references=" true"
                <application>CONFIGURATION CONTROLLED 3D DESIGNS OF
                MECHANICAL PARTS AND ASSEMBLIES</application>
             </Application_context>
          </frame_of_reference>
       </Product_definition_context>
    </frame_of_reference>
<Product_definition>
</ceb:STEP-RESULT>
```

EXAMPLE 6 – The following example defines two entity instances that containing a common nested instance with a ceb:id value of “bar1”.

The Update message is:

```
<ceb:STEP-UPDATE xmlns:ceb=" urn:iso10303-28:ceb" >
<Foo ceb:is="t1" ceb:id="_" ceb:status="new">
  <aa>
    <Bar ceb:id="bar1" ceb:status="new">
      <name>A Bar value</name>
    </Bar>
  </aa>
</Foo>
<Baz ceb:is="t1" ceb:id="_" ceb:status="new" >
  <barval>
    <Bar ceb:id="bar1" ceb:status="truncated"/>
  </barval>
</Baz>
<ceb:STEP-UPDATE>
```

EXAMPLE 7 - The following is an example of how EXPRESS inheritance is managed. For the EXPRESS declarations:

```
ENTITY x; END_ENTITY;
ENTITY y SUBTYPE OF (x);
att:INTEGER;
END_ENTITY;
ENTITY z SUBTYPE OF (x);
att:STRING;
END_ENTITY;

ENTITY a;
att : x;
END_ENTITY;

ENTITY b SUBTYPE OF (a);
SELF\a.att : y;  (* Redeclare att *)
END_ENTITY;

ENTITY c SUBTYPE OF (a);
END_ENTITY;

ENTITY d SUBTYPE OF (b, c); END_ENTITY;
```

The Part 21 data is:

```
#10=(X(),Y(30),Z('Mary'));
#20=C(#10);
#30=D(#10);
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=" urn:iso10303-28:ceb" >
      <C ceb:id=" I-20" />
      <D ceb:id=" I-30" />
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT xmlns:ceb=" urn:iso10303-28:ceb" >
<C ceb:id=" I-20" >
   <att>
      <Y-Z ceb:id=" I-10"  ceb:copies=" 2" >
            <Y.att>30</Y.att>
            <Z.att>Mary</Z.att>
      </Y-Z>
   </att>
</C>
<D ceb:id=" 30" >
   <att>
      <Y ceb:id=" I-10" ceb:partial=" true"  ceb:copies=" 2" >
            <att>30</att>
      </Y>
   </att>
</D>
</ceb:STEP-RESULT>
```

EXAMPLE 8 – The following example describes a two dimensional aggregate of a SELECT type where the positional indexes can be deleted.

The EXPRESS definition is:

```
TYPE name = STRING; END_TYPE;
TYPE a_select = SELECT (name, REAL); END_TYPE;
END_TYPE;

ENTITY crazy;
one: SET OF ARRAY[1:3] OF OPTIONAL a_select;
END_ENTITY;
```

The Part 21 data is:

```
#10=CRAZY (((“first”),$,10.0),($,$,$))
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=” urn:iso10303-28:ceb” >
      <Crazy ceb:id=” I-10” />
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT xmlns:ceb=” urn:iso10303-28:ceb” >
      <Crazy ceb:id="I-10">
            <one>
                  <name ceb:pos=”[1,1]”>first</name>
                  <no-value ceb:pos=”[1,2]”/>
                  <REAL ceb:pos=”[1,3]”>10.0</REAL>
                  <no-value ceb:pos=”[2,1]”/>
                  <no-value ceb:pos=”[2,2]”/>
                  <no-value ceb:pos=”[2,3]”/>
            </one>
      </Crazy>
</ceb:STEP-RESULT>
```

EXAMPLE 9 – The following example contains a SELECT type with positional indexes that cannot be deleted.

The EXPRESS definition is:

```
TYPE a_select = SELECT (STRING, REAL); END_TYPE;
TYPE b_select = SELECT (INTEGER, REAL); END_TYPE;
TYPE c_select = SELECT (a_select, b_select); END_TYPE;
END_TYPE;

ENTITY wild;
one: LIST OF c_select;
END_ENTITY;
```

The Part 21 data is:

```
#10=WILD ((A_SELECT(10.0),B_SELECT(20.0));
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=” urn:iso10303-28:ceb” >
      <Wild ceb:id=” I-10” />
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT xmlns:ceb=” urn:iso10303-28:ceb” >
      <Wild ceb:id="I-10">
```

```
         <one>
            <REAL ceb:pos=”[1, SP1-1]”>10.0</REAL>
            <REAL ceb:pos=”[2, SP1-2]”>20.0</REAL>
         </one>
      </Wild>
      <Spath-SP1<c_select a_select b_select</Spath-SP1>
</ceb:STEP-RESULT>
```

EXAMPLE 10 – In this example the message requests the values of derived attributes and defined type rules to be computed.

For the EXPRESS declaration:

```
CONSTANT
Pie : REAL := 3.142;
END_CONSTANT;

ENTITY circle;
radius : positive_measure;
center : point;
DERIVE
area : Real := Pie * radius * radius;
END_ENTITY;

ENTITY point;
coordinates : LIST [2:3] OF measure;
END_ENTITY;

TYPE measure = REAL; END_TYPE;

TYPE positive_measure = measure;
WHERE valid: SELF > 0;
END_TYPE;
```

The Part 21 data is:

```
#10 = CIRCLE (10.0, #20);
#20 = POINT ((1.0, 1.0));
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=” urn:iso10303-28:ceb” >
      <STEP-Header>
            <Derive/>
            <Type-rule/>
      </STEP-Header>
      <Circle ceb:id=” I-10” />
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT xmlns:ceb=” urn:iso10303-28:ceb” >
   <Circle ceb:id=” I-10” >
      <radius>10</radius>
      <center>
            <Point ceb:id=” I-20” >
                  <coordinates>
                        <measure ceb:pos=” [1]” >1.0</measure>
                        <measure ceb:pos=” [2]” >1.0</measure>
                  </coordinates>
            </Point>
      </center>
      <area ceb:type=” derived” >314.2</area>
      <valid ceb:type=” type rule” >true</valid>
   </Circle>
```

```
</ceb:STEP-RESULT>
```

EXAMPLE 11 – In this example the following EXPRESS-X schema is applied to the Product data given in Example 6. The example assumes that the product shown in Example 6 has an owner called “Fred” in an organization called “Bloggs and Sons”:

```
SCHEMA_VIEW ap_203_views;
USE FROM configuration_controlled_design;

VIEW  find_product;
FROM  pd:product
IDENTIFIED_BY product.name
RETURN
      pd;
END_VIEW;

VIEW  product_and_owner;
FROM  pd:product;
      own:cc_design_person_and_organization_assignment;
      pao:person_and_organization;
WHERE pd IN own.items
      own.role.name = ‘owner’;
      pao :=: own.assigned_person_and_organization;
IDENTIFIED_BY (pd);
SELECT
name                 : label := pao.the_person.first_name;
organization         : label := pao.the_organization.name;
part:                : product := pd;
END_VIEW;

END_SCHEMA_VIEW;
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=” urn:iso10303-28:ceb” >
      <PRODUCT_AND_OWNER ceb:status=” binding” >
            <argument-1>
                  <FIND_PRODUCT ceb:status=” binding” >
                        <argument-1>TIRE</argument-1>
                  </FIND_PRODUCT>
            </argument-1>
      </PRODUCT_AND_OWNER>
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT xmlns:ceb=” urn:iso10303-28:ceb” >
   <PRODUCT_AND_OWNER ceb:status=” view” >
      <name>Fred</name>
      <organization>Bloggs and Sons</organization>
      <part>
         <Product ceb:id=” I-670740”>
               <id>TIRE</id>
               <name>TIRE</name>
               <description>NOT SPECIFIED</description>
               <frame_of_reference>
                  <Mechanical_context ceb:id=”I-670730”>
                     <name></name>
                     <discipline_type>MECHANICAL</discipline_type>
                     <frame_of_reference>
                        <Application_context ceb:id=” I-607020”
                               ceb:references=” true”
```

```
                                    <text>CONFIGURATION CONTROLLED 3D
                              DESIGNS OF MECHANICAL PARTS AND
                              ASSEMBLIES</text>
                              </Application_context>
                           </frame_of_reference>
                        </Mechanical_context>
                     </frame_of_reference>
                  </Product>
            </part>
         </PRODUCT_AND_OWNER>
     </ceb:STEP-RESULT>
```

EXAMPLE 12 - For the EXPRESS Declarations:

```
ENTITY person
owns : vehicle;
END_ENTITY;

ENTITY car;
make : label;
END_ENTITY;

ENTITY boat;
make : label;
END_ENTITY;

ENTITY amphibious_car SUBTYPE OF (car, boat);
END_ENTITY;

TYPE vehicle = SELECT (car, plane); END_TYPE;
TYPE plane = STRING; END_TYPE;
TYPE label = STRING; END_TYPE;
```

The Part 21 is:

```
#10 = PERSON(#40)
#20 = PERSON(#50)
#30 = PERSON('Boeing 777');
#40 = CAR ('Jeep');
#50 = AMPHIBIOUS_CAR ('Sea Rover', 'BMW');
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=" urn:iso10303-28:ceb" >
      <STEP-Header>
            <SOAP 1.0/>
      </STEP-Header>
      <Person/>
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT xmlns:ceb=" urn:iso10303-28:ceb" >
   <Person id=" I-10" >
      <owns id=" I-40" xsi:type=" Car" >
                  <make>Jeep</make>
      </owns>
   </Person>
   <Person id=" I-20" >
      <owns id=" I-50"  xsi:type=" Amphibious_car" >
                  <make>Sea Rover</make>
                  <make>BMW</make>
      </owns>
   </person>
   <Person id=" I-30" >
      <owns xsi:type=" plane" >Boeing 777</owns>
   </Person>
```

```
</ceb:STEP-RESULT>
```

EXAMPLE 13 - For the car_reg schema:

```
SCHEMA car_reg;

ENTITY person
name : label;
INVERSE
owns : BAG OF car FOR owner;
END_ENTITY;

ENTITY car;
make : label;
owner : OPTIONAL person;
END_ENTITY;

TYPE label = STRING; END_TYPE;
END_SCHEMA;
```

the Part 21 data is:

```
#10 = CAR (“ Land Rover” , #40);
#20 = CAR (“ Jaguar” , #40);
#30 = CAR (“ Mini” , $);
#40 = PERSON (“ joe” );
#50 = PERSON (“ sue” );
```

The Export message is:

```
<ceb:STEP-QUERY xmlns:ceb=” urn:iso10303-28:ceb” >
      <STEP-Header>
            <SOAP 1.0/>
      </STEP-Header>
      <Person/>
      <Car/>
</ceb:STEP-QUERY>
```

The Result Message is:

```
<ceb:STEP-RESULT>
      <Person id=” I-40” >
            <name>joe</name>
            <owns>
                  <Car id=” 10”  SOAP-ENC:pos=” [1]” >
                        <make>Land Rover</make>
                        <owner href=” 40” />
                  </Car>
                  <Car id=” 20”  SOAP-ENC:pos=” [2]” >
                        <make>Jaguar</make>
                        <owner href=” 40” />
                  </Car>
            </owns>
      </Person>
      <Person id=” I-50” >
            <name>sue</name>
            <owns href=” null” />
      </Person>
      <Car id=” I-30” >
            <make>Mini</make>
            <owns href=” null” />
      </Car>
</ceb:STEP-RESULT>
```

EXAMPLE 14 – The following Export message configures the XML encoding of Circle instances to include Derived attribute values.

For the EXPRESS declaration:
```
CONSTANT
Pie : REAL := 3.142;
END_CONSTANT;

ENTITY circle;
radius : positive_measure;
center : point;
DERIVE
area : Real := Pie * radius * radius;
END_ENTITY;

ENTITY point;
coordinates : LIST [2:3] OF measure;
END_ENTITY;

TYPE measure = REAL; END_TYPE;

TYPE positive_measure = measure;
WHERE valid: SELF > 0;
END_TYPE;
```

The Part 21 data is:
```
#10 = CIRCLE (10.0, #20);
#20 = POINT ((1.0, 1.0));
```

The Export message is:
```
<ceb:STEP-QUERY xmlns:ceb=" urn:iso10303-28:ceb" >
      <Circle ceb:status=" configure" >
            <Derive/>
      </Circle>
</ceb:STEP-QUERY>
```

The Result Message is:
```
<STEP-RESULT xmlns:ceb=" urn:iso10303-28:ceb" >
   <Circle ceb:id=" I-10" >
      <radius>10</radius>
      <center>
            <Point ceb:id=" I-20" >
                  <coordinates>
                        <measure ceb:pos=" [1]" >1.0</measure>
                        <measure ceb:pos=" [2]" >1.0</measure>
                  </coordinates>
            </Point>
      </center>
      <area ceb:type=" derived" >314.2</area>
   </Circle>
</STEP-RESULT>
```

EXAMPLE 15 – The following example describes a two dimensional aggregate of a SELECT type where the positional indexes can be deleted.

The EXPRESS definition is:
```
TYPE name = STRING; END_TYPE;
TYPE a_select = SELECT (name, REAL); END_TYPE;
END_TYPE;

ENTITY crazy;
one: SET OF ARRAY[1:3] OF OPTIONAL a_select;
```

```
END_ENTITY;
```

The Part 21 data is:

```
#10=CRAZY (((“first”),$,10.0),($,$,$))
```

The Update message is:

```
<ceb:STEP-UPDATE xmlns:ceb=” urn:iso10303-28:ceb” >
      <Crazy ceb:id=” I-10>
            <one ceb:type=” updated” >
                  <REAL ceb:type=”new”
                        ceb:pos=”[1,2]”>5.0</REAL>
                  <REAL ceb:type=”delete”
                        ceb:pos=”[1,3]”>10.0</REAL>
                  <name ceb:type=”new”
                        ceb:pos=”[2,3]”>second</name>
            </one>
      </Crazy>
</ceb:STEP-UPDATE>
```

The new Part 21 data is:

```
#10=CRAZY (((‘first’),5.0,$),($,$,’second’))
```

EXAMPLE 16 – The following example contains a SELECT type whose positional indexes cannot all be deleted.

The EXPRESS definition is:

```
TYPE a_select = SELECT (STRING, REAL); END_TYPE;
TYPE b_select = SELECT (INTEGER, REAL); END_TYPE;
TYPE c_select = SELECT (a_select, b_select); END_TYPE;
END_TYPE;

ENTITY wild;
one: LIST OF c_select;
END_ENTITY;
```

The Part 21 data is:

```
#10=WILD ((A_SELECT(10.0),B_SELECT(20.0));
```

The Update message is:

```
<ceb:STEP-UPDATE xmlns:ceb=” urn:iso10303-28:ceb” >
      <Wild ceb:id=” I-10” />
         <one>
            <REAL ceb:pos=”[1, SP1-2]”>10.0</REAL>
            <REAL ceb:pos=”[2, SP1-1]”>20.0</REAL>
         </one>
      <Spath-SP1<c_select a_select b_select</Spath-SP1>
</ceb:STEP-UPDATE>
```

The new Part 21 data is:

```
#10=WILD ((B_SELECT(10.0),A_SELECT(20.0));
```