



IXRetail Technical Report:

Extending Schemas V2.0

July 4, 2005

Authors:

Richard Halter	MIC
----------------	-----

Copyright © National Retail Federation 2005. All rights reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the NRF, ARTS, or its committees, except as needed for the purpose of developing ARTS standards using procedures approved by the NRF, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the National Retail Federation or its successors or assigns.

TABLE OF CONTENTS

1	INTRODUCTION	3
1.1	OVERVIEW	3
1.2	EXTENSIBILITY DESIGN GOALS.....	4
1.3	IMPLEMENTATION & CONFORMANCE CONSIDERATIONS.....	4
2	EXTENDING ENUMERATIONS	5
2.1	ENUMERATION EXTENSION CONSIDERATIONS	5
2.2	ENUMERATION EXTENSION METHODOLOGY.....	5
2.3	ENUMERATION EXTENSION EXAMPLE - SAMPLE IXRETAIL SCHEMA AND INSTANCE DOCUMENTS	6
3	EXTENDING CONTENT WITH USER-DEFINED ELEMENTS	8
3.1	BEST PRACTICES FOR CONTENT EXTENSIBILITY	8
3.2	<XS:ANY> FOR ELEMENT EXTENSION	8
3.3	ELEMENT EXTENSION EXAMPLE	9
4	EXTENDING CONTENT WITH USER-DEFINED ATTRIBUTES.....	11
4.1	SPECIAL CONSIDERATIONS FOR ATTRIBUTE EXTENSIONS.....	11
4.2	ATTRIBUTE EXTENSION EXAMPLE	11
5	REFERENCED DOCUMENTS.....	15
6	DOCUMENT HISTORY	16
7	GLOSSARY	17

1 INTRODUCTION

1.1 Overview

IXRetail produces a set of XML schemas that are designed for interoperability between applications within the retail enterprise. The content of the schemas is based on the ARTS Data Model and an analysis of the common integration scenarios to be addressed by the recommendation. The standardization of business information always brings the issue of how to deal with necessary differentiation of business rules and practices without forsaking the goals of standardization. The system implementer has two options:

1. Request extensions to the IXRetail recommendation to support the necessary functionality and wait for the approval and release. This is often not reasonable in the fast-paced business environment.
2. Create unique Schemas based on the IXRetail recommendations. This option may be appropriate in a specialized point-to-point integration, but it is not practical in integration scenarios with multiple recipients of the affected message – all recipients of the non-standard XML would be affected, not just those requiring the unique information. This could create an adverse effect on interoperability in the common publish/subscribe integration scenarios.

If the implementation-specific extensions are pure extensions of the referenced schema, XML instance documents which are schema-valid against the IXRetail recommendation will be valid relative to the extended schema. However, XML documents with the extended content would not validate against the published IXRetail schemas unless the extensions are managed through mechanisms explicitly defined in the standard schema.

The goal of this Technical Report is to define a common method for establishing extension mechanisms within IXRetail published schemas that will allow extensions while preserving the ability to validate and process the instance documents using only the IXRetail published schemas.

The solution is for IXRetail to define a standard methodology for extending the IXRetail published schemas so that instance documents that are validated against the extensions are also valid against the IXRetail published schemas.

The use of user-defined XML Schemas to validate the extension content is possible with the recommended practices, but is not required and is beyond the scope of IXRetail Best Practices.

1.2 Extensibility Design Goals

The design goals for the IXRetail guidelines for implementation-specific extensions are:

1. User-defined extensions should neither require nor depend on modifications to any IXRetail schema specification. To ensure interoperability of products from multiple vendors, XML documents created with extensions must be schema-valid to the IXRetail standard schema.
2. The extension methodology must ensure that there is a clear separation between IXRetail XML data definitions and vendor-defined extensions. This separation will aid in conformance testing of vendor systems.
3. The development of these extensions must be easy for the XML Schema developer to make, and for the application developer to implement.
4. The extension mechanism must be clear and simple so that all appropriate extensions can be accomplished in a consistent manner.

1.3 Implementation & Conformance Considerations

1. This extension mechanism shall not be used to avoid use of the IXRetail approved standard schema data elements. Part of conformance testing will evaluate any extensions included in the candidate system to ensure the extensions are not duplicating functionality that is covered by the IXRetail standard schema.
2. The “Content and Syntax” conformance verification will ensure that an instance document is schema-valid relative to the relevant IXRetail Technical Specification. In addition, “Test Data Verification” testing may be performed to validate the content. This means that predefined information put into one application will yield comparable predefined results when extracted from another application. Therefore, instance documents that fail to populate the standardized data elements will not be deemed conformant.
3. Extensions that may have general applicability should be submitted to IXRetail for possible inclusion in future releases.

2 EXTENDING ENUMERATIONS

2.1 Enumeration Extension Considerations

IXRetail schemas often include attributes whose values can be enumerated, but which may, to varying degrees, be found with use to be incomplete. This could be handled in various ways:

1. Ignore the enumerations – address code values as an integration issue.
2. Document the enumeration values outside of the XML Schemas – this precludes meaningful schema validation of instance documents.
3. Define the enumeration values in the schema and define an extension mechanism

The IXRetail schema extension methodology defines a generalized flexible extension pattern. The basic approach is to define a common set of enumerations in one simpleType. In second simpleType define the standard pattern for extension values. Then define a third simpleType as the union of the enumerated simpleType and the standard pattern simpleType. The resultant vocabulary element has the advantage of making the extensions easy to recognize and parse. The result appears similar to a namespace prefix on an element name; while there is some conceptual similarity, these should not be confused with actual namespace prefixes.

2.2 Enumeration Extension Methodology

The following code pattern shall be followed when defining every enumeration list in an IXRetail schema that has any possibility of incompleteness or change:

1. Define, in common data, the standard extension pattern type:

```
<xs:simpleType name="IXREnumerationExtension">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9A-Za-z][0-9A-Za-z]*:[A-Z][0-9A-Za-z]*/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TypeCodeExtension">
  <xs:restriction base="IXREnumerationExtension"/>
</xs:simpleType>
```
2. Define the enumerated simpleType based on IXRetail Best Practices recommendations.
3. Define a simpleType as the union of the two preceding simpleTypes.
4. Define the related attribute as being of the type defined in step 3.

2.3 Enumeration Extension Example - Sample IXRetail Schema and Instance Documents

2.3.1 The IXRetail Schema Definition

This section shows how to create an enumerated list which allows the defined pattern.

Step 1: IXRetail will use the IXRetail defined pattern –

```
<xs:simpleType name="IXREnumerationExtension">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9A-Za-z][0-9A-Za-z]*:[A-Z][0-9A-Za-z]*"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TypeCodeExtension">
  <xs:restriction base="IXREnumerationExtension"/>
</xs:simpleType>
```

Step 2: IXRetail will enumerate the simpleType with a standard list of values:

```
<xs:simpleType name="TransactionLinkReasonEnumeration">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="Resume"/>
    <xs:enumeration value="DeferredBill"/>
    <xs:enumeration value="Return"/>
    <xs:enumeration value="LayAway"/>
    <xs:enumeration value="Voided"/>
    <xs:enumeration value="ReceiptReprint"/>
    <xs:enumeration value="Reservation"/>
    <xs:enumeration value="SuggestedItem"/>
    <xs:enumeration value="RetrospectiveLoyalty"/>
  </xs:restriction>
</xs:simpleType>
```

Step 3: IXRetail will create the union of the two; the enumerated simpleType and the extension pattern:

```
<xs:simpleType name="TransactionLinkReasonTypeCode">
  <xs:union memberTypes="TransactionLinkReasonEnumeration TypeCodeExtension"/>
</xs:simpleType>
```

Step 4: IXRetail will define an attribute in an IXRetail schema using the above union created in Step 3:

NOTE: IXRetail best practices define enumerated values will be instantiated as attributes.

```
<xs:complexType name="TransactionLinkCommonData">
  <xs:sequence>
```

```
<xs:element name="RetailStoreID" type="xs:string"/>
<xs:element name="WorkstationID" type="WorkstationIDType" minOccurs="0"/>
<xs:element name="SequenceNumber" type="SequenceNumberType"/>
<xs:element name="LineItemSequenceNumber" type="SequenceNumberType" minOccurs="0"/>
<xs:element name="BusinessDayDate" type="BusinessDayDateType" minOccurs="0"/>
<xs:element name="BeginDateTime" type="xs:dateTime" minOccurs="0"/>
<xs:element name="EndDateTime" type="xs:dateTime" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="ReasonCode" type="TransactionLinkReasonTypeCode" use="optional" default="Return"/>
<xs:attribute name="EntryMethod" type="EntryMethodTypeCode" use="optional" default="Scanned"/>
</xs:complexType>
```

2.3.2 Sample User Implementation

Example User Instance Document with IXRetail standard enumeration value:

Notice: the IXRetail defined value from the enumerated list written in Camel Case.

```
<TransactionLink ReasonCode="Return" >
  <RetailStoreID/>
  <WorkstationID/>
  <SequenceNumber/>
  <BusinessDayDate/>
</TransactionLink>
```

Example User Instance Document with User Defined enumeration value:

Notice: the user defined value following the extension pattern and written in Camel Case.

```
<TransactionLink ReasonCode="X:ExtendedWarranty" >
  <RetailStoreID/>
  <WorkstationID/>
  <SequenceNumber/>
  <BusinessDayDate/>
</TransactionLink>
```

3 EXTENDING CONTENT WITH USER-DEFINED ELEMENTS

IXRetail reviewed alternate extension techniques and determined the following is most appropriate for our needs.

3.1 Best Practices for Content Extensibility

The IXRetail extension mechanism is similar to those defined by other standards bodies. IXRetail has decided to include the W3C extension mechanism, `<xs:any>`, at the end of substantive complexTypes. IXRetail has also decided to allow the schema designers to use their best judgment, the input of reviewers and feedback from the user community to ensure that extension points are provided where they are likely to be required.

3.2 `<xs:any>` For Element Extension

The W3C has provided the `<xs:any>` element for allowing user defined content. The use of `<xs:any>` means that any element that conforms to the given definition (follows) will be valid against the schema. The following code shall be included at the end of appropriate complexType definitions, allowing vendor specific extensions to be added at the end of the IXRetail standard content.

```
<xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
```

The following table describes the various components of the `<xs:any>` statement above. (Abstracted from the W3C schema technical specification)

namespace	- Specifies the namespaces containing the elements that can be used - ##other – Only elements that are defined in a namespace that is not the IXRetail namespace may be included in the extension.
processContents	- Specifies how the XML processor should handle validation against the elements specified by this any element - lax - the XML processor should attempt to obtain the schema for the designated namespaces and validate the extending elements against that schema. If the schema cannot be obtained, however, no validation errors will occur.
maxOccurs	- Specifies the maximum number of times the any element can occur in the parent element; “unbounded” indicates that any number of extension elements may exist.
minOccurs	- Specifies the minimum number of times the any element can occur in the parent element; “0” indicates that no extension element is required.

3.3 Element Extension Example

3.3.1 IXRetail Sample Schema Vocabulary Component

Step 1: IXRetail identifies the appropriate location for an extension and places an “<xs:any>” (as defined above) there.

The underlined element shows the approved location and appropriate attributes for the <xs:any>

```
<xs:complexType name="POSLogInventoryTransaction">
  <xs:sequence>
    <xs:choice>
      <xs:element name="InventoryCount" type="POSLogITInventoryCount"/>
      <xs:element name="ReceiveInventory" type="POSLogITDocument"/>
      <xs:element name="TransferOut" type="POSLogITDocument"/>
      <xs:element name="ReturnToVendor" type="POSLogITDocument"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="Version" type="xs:decimal" use="required" fixed="2.1"/>
</xs:complexType>
```

3.3.2 User Defined Sample Extension Schema

Step 2: The user defines their unique elements in their own schema in their own target namespace.

```
<?xml version="1.0"?>
<!-- Key elements
  targetNamespace - identifies the namespace of the extension – required to put the extension schema in the same namespace as the instance
  document -->
<xs:schema
  targetNamespace="XXXExtensions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.nrf-arts.org/IXRetail/namespace/" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!-- defines the vendor specific global element names -->
  <!-- to be used in the instance document -->
  <xs:element name="MyStuff">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="MyElement" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SomeMoreOfMyStuff">
    <xs:complexType>
```

```
        <xs:sequence>
          <xs:element name="AnotherElementForMe" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

3.3.3 Sample User XML Instance Document (comments embedded)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Key Elements
  xmlns:XXX="XXXExtension"
  - names the extensions target namespace
  xsi:schemaLocation="XXXExtensions XXXSpecificSchema.xsd"
  - Ties the extension target namespace to the schema location for validation – necessary for the parser to locate the extension schema.
-->
<POSLog
  xmlns="http://www.nrf-arts.org/IXRetail/namespace/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Step 3 : The User adds their target namespace (xmlns :) to their instance document.

xmlns:XXX="XXXExtensions"

Step 4: The User adds the schema location in order for the parser to locate and tie their schema to their target namespace.

```
xsi:schemaLocation="XXXExtensions XXXSpecificSchema.xsd http://www.nrf-arts.org/IXRetail/namespace/ POSLog.xsd">
  <Transaction>
    <RetailStoreID>1</RetailStoreID>
    <WorkstationID>1</WorkstationID>
    <SequenceNumber>1</SequenceNumber>
    <BusinessDayDate>2005-05-18</BusinessDayDate>
    <OperatorID>1</OperatorID>
    <InventoryTransaction Version="2.1">
```

Step 5: The User adds their unique elements in the appropriate locations in their instance document.

```
    <!-- Vendor specific extension in the vendor extension -->
    <XXX:MyStuff>
      <XXX:MyElement>22</XXX:MyElement>
    </XXX:MyStuff>
  </InventoryTransaction>
</Transaction>
</POSLog>
```

4 EXTENDING CONTENT WITH USER-DEFINED ATTRIBUTES

4.1 Special Considerations for Attribute Extensions

Attribute usage within IXRetail has been primarily constrained to metadata and to enumerable values. Enumerable data seems likely to be a candidate for inclusion in the IXRetail specification – i.e., the desire to add a user-defined attribute may be an indication of an omission that should be corrected in the standard.

Further, attributes in XML have traditionally been of local scope to their parent element; they have not been associated with a namespace independently of that parent element.

6.2 <xs:anyAttribute> For Attribute Extensions

The W3C XML Schema specification also defines <xs:anyAttribute> as a mechanism that can be used to allow any attribute that conforms to the given criteria to be included in an XML instance document without precluding validation against the schema.

This mechanism *may* be included in IXRetail standard schemas, allowing vendor-specific attributes to be added to selected complexTypes.

```
<xs:anyAttribute namespace="##other" processContents="lax"/>
```

The meaning of the attributes (namespace and processContents) is the same as for <xs:any> (see above).

4.2 Attribute Extension Example

4.2.1 Sample IXRetail Schema

This complex type is from POSLogRetailTransaction.XSD

```
<xs:complexType name="POSLogRetailTransaction">
  <xs:complexContent>
    <xs:extension base="POSLogTransaction">
      <xs:sequence>
        <xs:element name="TillID" type="TillID" minOccurs="0"/>
        <xs:element name="ReceiptDateTime" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="LineItem" type="RetailTransactionLineItem" maxOccurs="unbounded"/>
        <xs:element name="Total" type="RetailTransactionTotal" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:element name="RestrictionValidation" type="RetailTransactionRestrictionValidation" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="SpecialOrderNumber" type="xs:string" minOccurs="0"/>
<xs:element name="ItemDelivery" type="RetailTransactionDelivery" minOccurs="0"/>
<xs:element name="ItemPickup" type="RetailTransactionPickup" minOccurs="0"/>
<xs:element name="Customer" type="RetailTransactionCustomer" minOccurs="0"/>
<xs:element name="Associate" type="RetailTransactionAssociate" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="FoodService" type="RetailTransactionFoodService" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="TransactionLink" type="POSLogTransactionLink" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="Version" type="xs:string" use="required" fixed="1.0"/>
<xs:attribute name="OutsideSalesFlag" type="xs:boolean" default="false"/>
<xs:attribute name="SuspendFlag" type="xs:boolean" default="false"/>
```

Step 1: IXRetail adds the extension point at the appropriate location in the schema

```
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

4.2.2 Sample User Extension Schema

Schema validation is possible by consuming applications unaware of a particular extension:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
```

```
A sample XSD schema file showing how to define some elements and attributes which will can be included at strategic
places in an IXRetail POSLog instance document.
Note: This XSD has to be in a different namespace
```

```
-->
<xs:schema
```

Step 2: The User defines their unique attributes in their own schema in their own target namespace.

```
targetNamespace="http://www.example.com/"
xmlns="http://www.example.com/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!--
  All global attributes are available
  -->
  <xs:attribute name="Greeting">
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="Hello" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
```

```
        <xs:enumeration value="Goodbye" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:schema>
```

4.2.3 Sample XML Instance Document

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
  A sample Instance document showing how to:
  a) Pickup definitions from more than one schema in more than one namespace
  b) Showing what sample additions would look like
-->
<POSLog
  xmlns="http://www.nrf-arts.org/IXRetail/namespace/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Step 3 : The User adds their target namespace (xmlns :) to their instance document.

```
  xmlns:Temp="http://www.example.com/"
```

Step 4: The User adds the schema location in order for the parser to locate and tie their schema to their target namespace.

```
  xsi:schemaLocation="http://www.nrf-arts.org/IXRetail/namespace/ POSLogRetailTransaction.xsd
    http://www.example.com/ SampleAttributeExtension.xsd"
  <Transaction xsi:type="POSLogRetailTransaction" Version="1.0">
    <RetailStoreID>HighStreet</RetailStoreID>
    <WorkstationID>POS5</WorkstationID>
    <SequenceNumber>4294967295</SequenceNumber>
    <BusinessDayDate>2001-08-13</BusinessDayDate>
    <BeginDateTime>2001-08-13T09:03:00</BeginDateTime>
    <EndDateTime>2001-08-13T09:05:00</EndDateTime>
    <OperatorID>John</OperatorID>
    <CurrencyCode>USD</CurrencyCode>
```

Step 5: The User adds their unique attributes in the appropriate locations in their instance document.

```
  <!-- Extra Attribute -->
  <LineItem VoidFlag="false" EntryMethod="Scanned" Temp:Greeting="Hello">
    <SequenceNumber>1</SequenceNumber>
    <BeginDateTime>2001-09-16T09:04:00</BeginDateTime>
    <!--Extra Attribute (same one again) -->
    <Sale ItemType="Stock" Temp:Greeting="Goodbye">
      <POSIdentity>
```

IXRetail Extending Schemas Technical Report V2.0

```
      <POSItemID>01234567890123</POSItemID>
    </POSIdentity>
    <ItemID>CA7865</ItemID>
    <MerchandiseHierarchy Level="Department">Chocolates</MerchandiseHierarchy>
    <Description>4oz Dark Chocolate</Description>
    <UnitListPrice ForeignAmount="0.87" Currency="GBP">1.79</UnitListPrice>
    <RegularSalesUnitPrice>1.63</RegularSalesUnitPrice>
    <ActualSalesUnitPrice>1.63</ActualSalesUnitPrice>
    <ExtendedAmount>4.89</ExtendedAmount>
    <Quantity>3</Quantity>
  </Sale>
</LineItem>
</Transaction>
</POSLog>
```

5 REFERENCED DOCUMENTS

The following documents are available from the W3C: <http://www.w3.org>

1. Extensible Markup Language (XML) 1.0
2. XML Schema Part 0: Primer
3. XML Schema Part 1: Structures
4. XML Schema Part 2: Datatypes

The following organizations were reviewed with respect to their extensibility methodologies.

1. Open Application Group, Inc. < <http://www.openapplications.org/> >
2. HR-XML Consortium < <http://www.hr-xml.org> >

6 DOCUMENT HISTORY

Version History

Ver	Date	Sections	Description of Change
1	6/5/2003	All	Initial Release
2	6/13/2005	All	Clarified the various methodologies for extending elements, attributes and enumerations

Version 1:

Authors:

Richard Halter	MIC
Stuart McGrigor	ARTS
Jim Galloway	AfterBot, Inc.

7 GLOSSARY

Term	Definition
xs:any	“Any well-formed XML from any namespace (default)” XML Schema Part 0: Primer
xs:anyAttribute	“enables attributes to appear in elements” XML Schema Part 0: Primer
lax	“The lax value of the processContents attribute instructs an XML processor to validate the element content on a can-do basis: It will validate elements and attributes for which it can obtain schema information, but it will not signal errors for those it cannot obtain any schema information.” XML Schema Part 0: Primer
##other	“Any well-formed XML that is not from the target namespace of the type being defined” XML Schema Part 0: Primer
Pattern	“constraint on the <u>value space</u> of a datatype which is achieved by constraining the <u>lexical space</u> to literals which match a specific pattern. The value of pattern <u>must</u> be a <u>regular expression</u> .” XML Schema Part 2: Datatypes
Union	“datatypes are those whose <u>value space</u> s and <u>lexical space</u> s are the union of the <u>value space</u> s and <u>lexical space</u> s of one or more other datatypes” XML Schema Part 2: Datatypes