ACORD™

Global Insurance Standards

Review of Schema Extension Methodologies

DRAFT

4/17/2003 7:39 PM

New York
One Blue Hill Plaza
P.O. Box 1529
Pearl River, NY 10965-8529
U.S.A.

London
London Underwriting Centre
Suite 2/ 5
3 Minster Court
Mincing Lane
London EC3R 7DD
United Kingdom

CONDITIONS AND TERMS OF USE

Section 1. Intellectual Property Rights. Copyright and other intellectual property laws protect the Standard. All intellectual property rights in the Standard belong to ACORD and are expressly reserved by ACORD except for your limited right to use, distribute, copy, display, and add to, but not modify or otherwise change, the fundamental data message format of the Standard granted under this Agreement. You agree to defend, indemnify, and hold ACORD, its officers, directors, employees and agents harmless from and against any claims, actions or demands, and liabilities, including, without limitation, reasonable legal and accounting fees, alleging or resulting from your use of the Standard or your breach of the terms of this Agreement. ACORD shall provide notice to you promptly of any claim, suit, or proceeding and shall assist you, at your expense, in defending any such claim, suit or proceeding.

Section 2. Attribution. As a condition of your limited rights granted in Section 1, you agree when using the Standard to reproduce (i) the following copyright notice: **© 2002 ACORD. All Rights Reserved.** and (ii) the statement "Used with permission of ACORD."

Section 3. No Warranties. to the extent permitted by law, acord disclaims all warranties on the standard, either express or implied, including, without limitation, the implied warranties of merchantability, non-infringement of third party rights, and fitness for particular purpose. the duration of any statutorily required warranty period shall be limited to thirty days from the date of your acceptance of these conditions and terms of use.

Section 4. Exclusive, Limited Remedy. The exclusive remedy for breach of this Agreement by ACORD shall be, at ACORD's option either (i) repair or replace the Standard or (ii) a refund of the price, if any, which you paid to use the Standard which shall be your sole and entire monetary relief and ACORD's total monetary liability to you.

Section 5. Limitation of Liability. (i) Special Damages. in no event shall acord or an affiliated company of acord be liable for any damages, whether direct, indirect, special, incidental, consequential, or otherwise (including lost profit or business revenue, or goodwill) arising in connection with this agreement, the standard, or the use thereof, even if acord has been advised, knew or should have known of the possibility of such damages.

(ii) ACORD Not Liable. acord will not be responsible, and shall not be liable, for any losses or damages to you arising in connection with this agreement, the standard or the use thereof, except to the extent that such losses or damages were caused solely and directly by ACORD's gross negligence or willful misconduct, in which case ACORD's liability shall, in no event, exceed the lesser of the actual damage or loss caused, or $50.00. under no circumstances shall acord be liable for any damages or losses caused in any manner by your failure to perform your obligations under this agreement, or by your breach of any provisions herein, or by modifications or enhancements made to the standard by anyone other than an authorized acord representative. acord shall not be liable to you, any party involved in a transaction initiated by you using the standard or any other third party for any claims or damages, including, without limitation, losses or damages of any and every nature, resulting from the loss or corruption of data, inability to access the internet or internal systems, or inability to transmit or receive information, caused by, or resulting from, delays, non-delivery, or service interruptions whether or not caused by the fault or negligence of acord. acord shall not be responsible for the servers not being accessible on the internet or internal systems due to circumstances not in the direct control of acord such as, without limitation, your or any other third party's equipment capabilities, limitations or internet service provider limitations. acord shall not be responsible or liable for unauthorized dissemination of any of your data, whether as a result of defeat of data security, misappropriation or misuse of passwords, or any other cause not in the direct control of acord.

(iii) Allocation of Risk. you expressly agree that the limitations and exclusions of these terms fairly represent our agreement on the allocation of risk between us as to ACORDs obligations hereunder. the payments made by you, if any, reflect this allocation of risk and the exclusion of special, incidental, consequential, indirect or punitive, or other damage and other damage limitations in this agreement notwithstanding that any exclusive remedy shall fail of its essential purpose or otherwise be unavailable

Section 6. General. Any dispute arising out of or related to these conditions and terms of use or the Standard shall be subject to binding arbitration in the District of Columbia conducted by a single arbitrator chosen pursuant to and shall be conducted in accordance with the Streamlined Rules of Arbitration of J.A.M.S./ENDISPUTE. This Agreement and performance hereunder shall be governed and construed in accordance with (i) the laws of the State of New York, without reference either to its choice of law provisions or its Arbitration statute or other dispute resolution provisions, and (ii) the U.S. Arbitration Act, 9 U.S.C. §§ 1 *et seq*. Subject to the first sentence of this Section 6, any and all proceedings relating to the subject matter hereof shall be maintained in the federal and local courts of the District of Columbia, as applicable, which courts shall have exclusive jurisdiction for such purpose, and you hereby consent to the personal jurisdiction of such courts. In the event that any term of this Agreement conflicts with the law under which this Agreement is to be construed, or if any term is held invalid by any court of competent jurisdiction, such term shall be deemed to be restated to reflect, as nearly as possible, the original intention of both of the parties in accordance with applicable law, and the remainder of this Agreement shall remain in full force and effect. In the event that such invalid provision relates to matters which constitute the essence of the Agreement including, but not limited to, matters relating to the ownership of the Standard, and such invalid provision cannot be restated to reflect the original intention of both of the parties, the entire Agreement shall be deemed null and void, in which case, no licenses or other authorizations hereunder shall be deemed effective. No waiver of any term of this Agreement shall be deemed a further or continuing waiver of such term or any other term. This Agreement constitutes the entire agreement between you and ACORD with respect to the Standard and your rights and obligations regarding your access and use which supersedes all written or oral, prior and contemporaneous representations, discussions and agreements relating to its subject matter. Any changes to this Agreement must be made in writing, signed by an authorized representative of ACORD. This Agreement shall be construed fairly in accordance with the plain meaning of its terms, regardless of which party was primarily responsible for preparation and drafting of it. No statement, representation, warranty, covenant or agreement of any kind not expressly set forth in this Agreement shall affect, or be used to interpret, change or restrict, the express terms and provisions of this Agreement.

Table Of Contents

1 BACKGROUND

ACORD needs a standard way to extend schemas via a standard schema mechanism. The ACORD P&C membership has SPX (Service Provider eXtensions) as a methodology to document in a processable form the following information:

- Extensions and restrictions to the ACORD specification for company unique data and codes
- Edits - Business rules and data relationships
- GUI management – field names, labels, default values, display values, error, help, and prompt text

This information is captured in an XML document form that is specified in the ACORD standard. We have never provided guidance or set any requirements on how this information should be interpreted. We have also never indicated what the result of processing this information should result in. The first bullet is primary interest in what should the result be. It has been proposed that some sort of hybrid DTD or schema could be one result. Currently there are no tools that accept this information.

SPX and its extension and restriction mechanism were designed in a DTD based environment. DTDs had no built-in means to change the design of the original specification. We did document a way that parameter entities could be used, and had the ACORD DTD contained the proper hooks to support this that might have been one way to approach the problem in a semi-standard way. Instead we defined a documentation format that could be processed (because it is XML) that told where the changes should be made. Implementation of these changes has been left up to the user.

Since then, ACORD and its members have been developing a schema to complement the DTD. Several members have already played with some ideas on how to extend the schema based upon methods defined in the W3C schema specification. This document will try to document the methods discussed, there pros and cons and a final recommendation on how to do schema based extensions.

2 DESIGN GOALS

The following are the general requirements or guidelines that the various methods will be rated against:

1. We want a controlled way of adding or removing specific elements and codes from the original standard.
2. We want a single content model for a given element or type. An address element should always allow the same elements in any use, there should not be a variety of content models.
3. We want to be able to identify the source/ownership/definition of the extensions.
4. We want an easily maintainable method that doesn't require the editing or direct modification of the ACORD schema.
5. We want the modifications once made to be easily migrated to the next version of the standard.
6. We want a method that doesn't require duplication of the existing schema to define the extensions and restrictions. This is primarily and issue for restriction and enumerated lists.
7. For the near future, we need a schema method that can be mimicked in a DTD. We can change the DTD to support the design, but the result of using an extension feature should not result in something we couldn't model in a DTD. To model a feature in the DTD, we might have to introduce some restrictions in the schema. For instance, something that requires multiple namespaces to appear in the data stream would be difficult to implement and would require limiting the number and predefining the prefixes.
8. We want a method that members and outside organizations can easily implement and is properly supported by the tools available. Some features in schema may not be consistently implemented and as such would be less appealing than another solution.

Here is another way of looking at these requirements:

- Business Issues List
  - o Handle company-unique content (tags, code lists, code list values) in a standard way that XML parsers will recognize
  - o Leverage the power of schema validators to define and check data in business documents

- Make it as easy as possible for users of ACORD XML to define extensions in a processable, “in-band” way
- Be backward compatible with V1.x for documents without company-unique code extensions
- Avoid forcing changes in existing instance documents

- Technical Issues List
  - Works with more than one parser, covering both MS and Java platforms
  - Minimum impact on instance document
  - Name collision protection
  - Tight validation
  - Allow users to re-use ACORD code list enumerations
  - Minimize schema code needed in user schemas

3 TECHNICAL OPTIONS

SPX defines a set of extension/restriction capabilities that form the minimum requirements for the functionality that we want to support. Beyond what was thought of in the SPX design, when DTDs were the target, there are additional features that we need to consider to truly support schema extensions. These features are:

- SPX Related
  - Creation of new elements. Associating a new element with an ACORD data type (and optional length), an ACORD entity, any ACORD or SPX defined code list, and a dictionary description
  - Creation of new code lists or replacements for ACORD code lists. Defining their name, content, associated question for the GUI, optional name of the ACORD code list being replaced
  - Creation or new code values. Defining the code, description, code display value (for GUI) and a default indicator
  - Specify a shorter length on appropriate data types.
  - Defining content (using existing ACORD or SPX defined elements) for of an element, its optionality, repeatability and logic flag (OR, XOR, or none) for:
    - SPX defined elements
    - ACORD elements
    - No mechanism provided to support the extension of entities directly
  - Additional properties
    - No mechanism to remove (restrict) an element from an ACORD element definition, extension is the only SPX method.
    - New elements are added to the end of the ACORD listed elements. ACORD only adds optional elements during point releases. Added elements can be defined as Required.
    - Changes in SPX are restricted to message content, so no new messages or changes to the framework are allowed via SPX.
- Schema Related
  - Restriction of element content
  - Creation of new attributes
  - Restriction of attributes
  - Creation of new entities (groups in schema)
  - Creation of new data types other than code lists and the existing ACORD types
  - Use of elements, data types, etc. from other standards

Note: SPX has defined the SPXPrefix (com.foo_) as the mechanism to identify SPX extensions. Schema modifications would use a namespace and its associated prefix (foo:) to identify the extensions.

The following methods or schema features have been reviewed:

- Directly editing the ACORD schema
- Schema Wildcards - xsd:any and xsd:anyAttribute
- Derived types – xsi:type
- Substitution groups
- Abstract elements – abstract and substitution group combination
- Abstract types – xsi:type
- xsd:redfine

Many of these features can be used in the original schema and are not necessarily just methods for extending an existing schema. Many of these require setup in the original schema to make use of the feature, some create additional requirements for the resulting data stream. Of these methods, xsd:redefine is the only feature clearly for the purpose of changing an existing schema without modifying the original file. The other features can be looked at as methods of extension (given the required setup) and you had some method for implementing them The W3C presents xsd:redfine in what has been termed an adapter schema by some folks. This is the only way xsd:redefine can be used, but the general approach can be used as a method to implement the other features.

Note: This review is based upon ACORD's current situation of having an existing specification and implementations. So issues about changing or supporting a feature with a DTD and minimizing impact on the data stream are important until we can get to a point where we have no constraints. If were working with a brand new development many of these issues would not be important.

We also review issues related to:

- Adapter Schema Implementation
- Special issues related to enumerated types in elements and attributes

Warning: Many of the schema features reviewed here increase the complexity of the schema and make it harder to maintain by hand (with minimal development tool support). Pick the features you want to use carefully and limit your use. Don't try to use all these features in a complex schema as the interplay between them and then later extension may be impossible to understand or predict.

3.1 Directly Editing the ACORD Schema

Your first response to this requirement might be to go in and edit the ACORD schema directly and add you new elements and codes or delete things as you want. This might be the "easiest way" to accomplish that immediate task but in the long run it has the following problems:

- It will be difficult to move your changes to the next release of the schema. You will essentially have to redo all the work each time a schema is released. In redoing the work you may introduce errors because you place something out of order or miss the change.
- There is a semantic meaning to the ACORD standard/schema and its structures. The ACORD specification establishes a vocabulary and its definitions. If you just change the ACORD schema and don't flag your changes by changing the namespace, your trading partners will not know that you have made changes. Then a data stream that validates with the modified ACORD schema most likely will not validate with official schema.

3.1.1 Pros

1. Gets the job done.
2. Can use any of the methods to define the extensions.

3.1.2 Cons

1. The changes are not portable. The changes cannot be easily applied to the next version of the schema.
2. Would not encourage user to define their own (new) elements in their own schema and namespace.
3. Depending upon methods used in editing the schema, there may be issues for the data stream.

3.2 Schema Wildcards – xsd:any and xsd:anyAttribute

The W3C Schema standard has two wildcard elements that can be used when defining the content model. These wildcards can be used in the initial schema design as well as in extension definitions. This is perhaps the easiest way to allow extension when you are authoring a schema, or when you are trying to create extensions to an existing schema.

Schema Syntax – xsd:any

```
<any
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  namespace = ((##any | ##other) | List of (anyURI | (##targetNamespace | ##local))
)  : ##any
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</any>
```

Schema Syntax – xsd:anyAttribute

```
<anyAttribute
  id = ID
  namespace = ((##any | ##other) | List of (anyURI | (##targetNamespace | ##local))
)  : ##any
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</anyAttribute>
```

Sample Usage:

```
<xsd:complexType name="Communications_Type">
   <xsd:sequence>
      <xsd:element ref="PhoneInfo" minOccurs="0"
            maxOccurs="unbounded"/>
      <xsd:element ref="EmailInfo" minOccurs="0"
            maxOccurs="unbounded"/>
      <xsd:element ref="WebsiteInfo" minOccurs="0"
            maxOccurs="unbounded"/>
      <xsd:any maxOccurs=’unbounded’ namespace=’##other’ processContents=’lax’/>
   </xsd:sequence>
   <xsd:attribute name="id" type="ID"/>
  <xsd:anyAttribute namespace=’##other’ processContents=’lax’/>
</xsd:complexType>
```

The above example shows how this might be used in our schema. To use these, you would have to determine which elements or types you would want to extend. To agree with our SPX requirements, xsd:anyAttribute would never be used and xsd:any would be placed at the end of all the aggregates and messages.

There are two types of control that can be defined with these wildcards. First you can control where these elements or attributes can come from with the namespace attribute and these values:

- ##any – this would allow elements or attributes from any namespace/schema (default)

- ##other – any namespace other than the targetNamepsace
- Specific URIs/namespaces – anyURI – A list of specific namespaces that can be used.
- ##targetNamespace – must belong to the targetNamepace
- ##local – Any unqualified element or attribute from the local model

The second control is how the elements or attributes that are substituted for the wildcards should be processed or validated. The processContents attribute has the following values:

- skip – skip over any elements or attributes that have been substituted – no processing or validation, but must be well-formed
- lax – Validate where you can, but don't worry if you can't
- strict – provide strict validation based upon the schema these elements or attributes comes from (default)

3.2.1 Pros

1. Easily configured.
2. Any element or attribute can be used wherever these wildcards are placed in the original schema.

3.2.2 Cons

1. There is no control of what can be used. Anything can be used as a replacement. As such you might get completely unexpected or unrelated data at a location you don't want it.
2. If an aggregate appears more than once in a data stream, its content can be completely different.
3. This only supports extensions to element content or attribute use.
4. Namespace of new elements must appear in data stream

3.3 Derived Types – xsi:type

In a schema, you will typically have some core types that you will extend and modify. These types are derived from the original source. Any type that is used in a derivation can automatically be used in place of the original type. For instance, if I have a data type A from which I derive types B, C, and D. Element <Foo> is based upon type A. In a data stream<Foo> can be mapped to one of the derived types. The schema parser already knows that A, B, C, and D are related because of the derivation. It also knows that <Foo> is based upon the original A. In your data stream you can use a special schema attribute, xsi:type, to indicate that you want a particular instance of <Foo> to be use the C definition. By default, this <Foo> uses the A data type. But <Foo xsi:type='C'> uses the C data type.

Here is an example from the W3C Schema specification primer. This is the part of the schema showing how the USAddress, and UKAddress are derived by extension from Address. It is this simple reuse or derivation used in the schema that allows this to work.

```
<complexType name="Address">
  <sequence>
   <element name="name"   type="string"/>
   <element name="street" type="string"/>
   <element name="city"   type="string"/>
  </sequence>
 </complexType>

 <complexType name="USAddress">
  <complexContent>
   <extension base="Address">
    <sequence>
     <element name="state" type="USState"/>
     <element name="zip"   type="positiveInteger"/>
    </sequence>
   </extension>
  </complexContent>
```

```
    </complexType>

    <complexType name="UKAddress">
     <complexContent>
      <extension base="Address">
       <sequence>
        <element name="postcode" type="UKPostcode"/>
       </sequence>
       <attribute name="exportCode" type="positiveInteger" fixed="1"/>
      </extension>
     </complexContent>
    </complexType>
```

Elsewhere in this schema is this use of these types:

```
    <complexType name="PurchaseOrderType">
     <sequence>
      <element name="shipTo"     type="Address"/>
      <element name="billTo"     type="Address"/>
      <element ref="comment"     minOccurs="0"/>
      <element name="items"      type="Items"/>
     </sequence>
     <attribute name="orderDate" type="date"/>
    </complexType>
```

So we have the elements <shipTo> and <billTo> defined to be of type Address in the schema. Note that the above schema is referenced with the namespace prefix ‘ipo:’. Now in the data stream we can use the following:

```
       <shipTo exportCode="1" xsi:type="ipo:UKAddress">
           <name>Helen Zoe</name>
           <street>47 Eden Street</street>
           <city>Cambridge</city>
           <postcode>CB1 1JR</postcode>
       </shipTo>

       <billTo xsi:type="ipo:USAddress">
           <name>Robert Smith</name>
           <street>8 Oak Avenue</street>
           <city>Old Town</city>
           <state>PA</state>
           <zip>95819</zip>
       </billTo>
```

In the data stream we were able to choose if we wanted to use the basic Address type definition, USAddress or UKAddress by adding the xsi:type attribute and selecting the data type we wanted to use. The schema parser is able to make switch between these types and validate their content appropriately.

In this form, this is not an example of extending an existing schema, but it does illustrate a mechanism that allows some flexibility in the design of schema types. If you use some common core types as definitions for other types by extension, then you are allowing this sort of substitution to occur in the data stream.

Schema Syntax – xsd:complexType

```
   <complexType
     abstract = boolean : false
     block = (#all | List of (extension | restriction))
     final = (#all | List of (extension | restriction))
     id = ID
     mixed = boolean : false
     name = NCName
     {any attributes with non-schema namespace . . .}>
     Content: (annotation?, (simpleContent | complexContent | ((group | all |
                  choice | sequence)?, ((attribute | attributeGroup)*,
   anyAttribute?))))
   </complexType>
```

3.3.1 Pros

1. Allows for multiple content models for the same element.
2. Doesn't require any setup in the schema to use this feature.
3. The use of "an extension" can be determined at runtime when the datastream is built.

3.3.2 Cons

1. First impulse might be to edit the ACORD schema to make these changes.
2. This is a substitution feature that would allow a single element to have multiple content models. Not really an extension mechanism.
3. Difficult for a human reader to know all the possible configurations an element might contain if extension and restriction are used extensively in a schema design. This guarantees the need of tools to aide the reader of an XML data stream.
4. Requires the use of xsi:type. This requires the addition of this attribute in the corresponding DTD wherever this complexType is used in the schema.

3.3.3 Controlling Substitution by Derivation

If you do not want to allow this sort of substitution to occur, the W3C schema specification provides the block attribute. The block attribute has the following values:

- #all – block all types of substitution for this type
- extension – block substitutions based upon extension for this type
- restriction – block substitutions based upon restriction for this type
- substitution – block substitution by substitution groups
- list of the above – a whitespace separated list of the types of substitution that you want to block

The block attribute can be used on individual type definitions as a fine level of control on this feature. A higher level setting can be made with the blockDefault attribute on the <schema> element.

Schema syntax – xsd:schema

```
<schema
  attributeFormDefault = (qualified | unqualified) : unqualified
  blockDefault = (#all | List of (extension | restriction | substitution))  : ''
  elementFormDefault = (qualified | unqualified) : unqualified
  finalDefault = (#all | List of (extension | restriction))  : ''
  id = ID
  targetNamespace = anyURI
  version = token
  xml:lang = language
  {any attributes with non-schema namespace . . .}>
  Content: ((include | import | redefine | annotation)*, (((simpleType |
          complexType | group | attributeGroup) | element | attribute |
               notation), annotation*)*)
</schema>
```

Controlling substitution is something you should consider. By default this substitution is allowed. This might result in extra flexibility that you never intended. Unless otherwise designed to support substitution by derivation you might want to shut off this feature until you have a real need for it.

3.3.4 Controlling Type Derivation

In addition to controlling substitution, you can also control the type of derivation allowed. The final attribute on xsd:complexType allows you to specify this functionality for each complexType. The possible values are:

- #all – stops any further extension or derivation
- extension – stops further extension
- restriction – stops further restriction
- “” – empty string resets a finalDefault
- list – extension and restriction in any order – the same as #all

Just like substitution by derivation, there is a finalDefault attribute on the xsd:schema element that allows you to set a global derivation control, while the final attribute allows a finer grain control and allows you to decide this on an individual basis. Allowed values are:

- #all
- extension
- restriction

Note: If you have gone to the trouble to create complexTypes, there is probably not much reason to restrict how they might be modified in the future. The ultimate control here is to never create complexTypes and just create anonymous types.

3.4 Substitution Groups

Substitution groups allow elements to be substituted for each other and doesn’t require the use of xsi:type in the data stream but requires setup in the original schema. In the schema the attribute substitutionGroup is used to identify or assign the elements that belong to the group.

Here is the example from the W3C schema primer.

```
<element name="shipComment" type="string"
         substitutionGroup="comment"/>
<element name="customerComment" type="string"
         substitutionGroup="comment"/>

<element name="comment" type="string"/>

 <complexType name="Items">
  <sequence>
   <element name="item" minOccurs="0" maxOccurs="unbounded">
    <complexType>
     <sequence>
      <element name="productName" type="string"/>
      <element name="quantity">
       <simpleType>
        <restriction base="positiveInteger">
         <maxExclusive value="100"/>
        </restriction>
       </simpleType>
      </element>
      <element name="USPrice"    type="decimal"/>
      <element ref="comment" minOccurs="0"/>
      <element name="shipDate"   type="date" minOccurs="0"/>
     </sequence>
     <attribute name="partNum" type="SKU" use="required"/>
    </complexType>
   </element>
  </sequence>
```

```
    </complexType>
```

The resulting data stream could look like this:

```
   <items>
      <item partNum="833-AA">
        <productName>Lapis necklace</productName>
        <quantity>1</quantity>
        <USPrice>99.95</USPrice>
        <ipo:shipComment>
          Use gold wrap if possible
        </ipo:shipComment>
        <ipo:customerComment>
          Want this for the holidays!
        </ipo:customerComment>
        <shipDate>1999-12-05</shipDate>
      </item>
    </items>
```

The primary requirement is that the head element (the one identified in the substitutionGroup attribute) must be a globally defined element. Elements in a substitution group must have the same type as the head element, or they can have a type that has been derived from the head element's type. The elements in the substitution group are not required to be used and the head element is not precluded from being used either. This is a simple method that allows interchangeability.

Schema Syntax – xsd:element

```
   <element
     abstract = boolean : false
     block = (#all | List of (extension | restriction | substitution))
     default = string
     final = (#all | List of (extension | restriction))
     fixed = string
     form = (qualified | unqualified)
     id = ID
     maxOccurs = (nonNegativeInteger | unbounded)  : 1
     minOccurs = nonNegativeInteger : 1
     name = NCName
     nillable = boolean : false
     ref = QName
     substitutionGroup = QName
     type = QName
     {any attributes with non-schema namespace . . .}>
     Content: (annotation?, ((simpleType | complexType)?, (unique | key | keyref)*))
   </element>
```

3.4.1 Pros

1. Does not require any modification to the source schema.
2. Does not require an attribute in the data stream to identify the substitution.

3.4.2 Cons

1. The substitution group can be defined without the head element having any notation that they exist. This makes it difficult to manage these groups and it makes it more difficult for the human reader to read the XML data stream.
2. Data stream cannot be validated with a DTD implementation without modifying the DTD. The DTD could use an OR group of all the elements in the substitution group.
3. Not really an extension mechanism as it is a way to introduce different names for the same element. There is no way to extend the content of the element.

4. There maybe a need for this sort of functionality, but it isn't an extension or restriction feature.

3.5 Abstract Elements and Types

Schemas provide a mechanism to declare abstract types or elements. The usage and requirements are different depending on whether you are defining an abstract element or an abstract type.

3.5.1 Abstract Element

An abstract element uses the substitutionGroup attribute to define the relationship to the head element that is now defined to be abstract. The primary result is that the head element can no longer be used in the resulting data stream.

Here is the example from the W3C schema primer for an element abstraction.

```
<element name="shipComment" type="string"
         substitutionGroup="comment"/>
<element name="customerComment" type="string"
         substitutionGroup="comment"/>

<element name="comment" type="string" abstract=”true”/>

 <complexType name="Items">
  <sequence>
   <element name="item" minOccurs="0" maxOccurs="unbounded">
    <complexType>
     <sequence>
      <element name="productName" type="string"/>
      <element name="quantity">
       <simpleType>
        <restriction base="positiveInteger">
         <maxExclusive value="100"/>
        </restriction>
       </simpleType>
      </element>
      <element name="USPrice"    type="decimal"/>
      <element ref="comment" minOccurs="0"/>
      <element name="shipDate"   type="date" minOccurs="0"/>
     </sequence>
     <attribute name="partNum" type="SKU" use="required"/>
    </complexType>
   </element>
  </sequence>
 </complexType>
```

The resulting data stream could look like this:

```
<items>
   <item partNum="833-AA">
     <productName>Lapis necklace</productName>
     <quantity>1</quantity>
     <USPrice>99.95</USPrice>
     <ipo:shipComment>
       Use gold wrap if possible
     </ipo:shipComment>
     <ipo:customerComment>
       Want this for the holidays!
     </ipo:customerComment>
     <shipDate>1999-12-05</shipDate>
   </item>
 </items>
```

With the substitution group example, the original head element <ipo:comment> could have also been used in the data stream. Now declared as an abstract it is not allowed.

3.5.1.1 Pros

1. Does not require an attribute in the data stream to identify the substitution.

3.5.1.2 Cons

1. Requires setup in the schema to potentially create dummy element to apply the abstract attribute. This dummy element then needs to be substituted for any current use of the element.
2. The head element is described with the abstract attribute and the only connection to the actual usable elements has to be tracked down by with the substitutionGroup attribute value.
3. Data stream cannot be validated with a DTD implementation without modifying the DTD. The DTD could use an OR group of all the elements in the substitution group.
4. This is not really an extension mechanism as it is a way to introduce different names for the same element. There maybe a need for this sort of functionality, but it isn't an extension or restriction This is a variation on the substitution groups that basically only removes the ability to use the head element in the data stream.

3.5.2 Abstract Types – xsi:type

To declare an abstract type you don't have to use the substitutionGroup attribute. This is an xsd:element feature, not an xsd:complexType. The tradeoff, seems to be that you have to use the xsi:type in the resulting data stream. Once again, like the abstract element, the head object, cannot appear in the data stream.

Here is an example from the W3C schema primer for a complexType abstraction:

```
<complexType name="Vehicle" abstract="true"/>

 <complexType name="Car">
  <complexContent>
   <extension base="Vehicle"/>
  </complexContent>
 </complexType>

 <complexType name="Plane">
  <complexContent>
   <extension base="Vehicle"/>
  </complexContent>
 </complexType>

 <element name="transport" type="Vehicle"/>
```

The complexType Vehicle (would have other content) has been described to be abstract. Typically you would describe the common properties of a vehicle in this location. We have also defined two other types that inherit from the Vehicle type; Car and Plane. These new types would add the properties that are unique to a Car or a Plane. The original Vehicle type is here as a management tool, and it is what the new element <transport> is based upon. Any element that was to use one of these types would be declared to be of type Vehicle if you want the content model to be flexible, or you could use Car or Plane directly.

The resulting data stream would look something like this:

```
 <transport xsi:type="ipo:Car"/>
```

The data stream at run time can be configured to conform to either the Car or Plane type and you have to indicate which of these types has been applied. Vehicle would not be allowed because of its abstract declaration.

3.5.2.1 Pros

1. Allows a single element to have multiple content models.
2. At runtime the content model can be chosen.

3.5.2.2 Cons

1. Requires the creation of a dummy type that would replace the existing types in the schema. Then a new type would have to be created so the element could have content. With that setup, any complexType could be extended or modified.
2. The head type is described with the abstract attribute but the only connection to the actual usable type has to be tracked down by tracking the base attribute values of the types.
3. The type used as the head of an abstract cannot be used in the document. This would force the creation of dummy objects in the ACORD schema to support this functionality.
4. Anything to be extended with this mechanism must be engineered into the original schema. The abstract has to be defined first before it can be extended. The safe solution would require making an abstract type for every complexType.
5. For compatible DTD processing, the xsi:type attribute would have to be added to all elements that might support this feature.

3.6 Redefinition – xsd:redefine

The W3C Schema specification has one true method for redefining by extension or restriction an existing schema. This functionality is provided by the xsd:redefine element. The xsd:redefine acts like an xsd:include. If it has no content it would result in the same outcome as using xsd:include. To use xsd:redefine you place one or more of the following objects inside the element and reference an existing object from the included schema:

- simpleType – can only be restricted
- complexType – with simpleContent or complexContent both restriction and extension are allowed
- group – see example
- attributeGroup – see example

By using xsd:redefine you are truly modifying the original definition and every element or attribute referencing one of the above objects would now be redefined. This doesn't create optional content models, it changes the original design. This feature allows you to add and modify the content of the existing schema objects, but you can't make big elaborate changes. You can only make changes to the objects listed above. If the included schema is not configured to use these as globally defined named objects, you would not be able to make any modifications.

Here is an example from the W3C primer that shows how the Address type is extended with a new local attribute:

```
<xsd:schema targetNamespace="http://www.example.com/IPO"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.example.com/IPO">

 <!-- bring in address constructs -->
 <xsd:redefine
  schemaLocation="http://www.example.com/schemas/address.xsd">

     <!-- redefinition of Address -->
     <xsd:complexType name="Address">
      <xsd:complexContent>
        <xsd:extension base="Address">
         <xsd:sequence>
          <xsd:element name="country" type="xsd:string"/>
         </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
     </xsd:complexType>

 </xsd:redefine>

 <!-- etc. -->
```

```
</xsd:schema>
```

Note: Outside of the xsd:redefine element, any such attempt to define a complex type with the same name (and in the same namespace) as the base from which it is being derived would cause an error. But in this case, there is no error, and the extended definition of Address becomes the only definition of Address. It is completely replaced.

Schema Syntax – xsd:redefine

```
<redefine
  id = ID
  schemaLocation = anyURI
  {any attributes with non-schema namespace . . .}>
  Content: (annotation | (simpleType | complexType | group | attributeGroup))*
</redefine>
```

Note: All of this functionality seems best suited for extending the original definitions. A typical extension is illustrated above where the original type is referenced and the new object (and element in this case) is defined and added to the existing content. The only connection between the two files comprising this new schema is the reference to the type name. Now look at what it would take to restrict the content.

Here is the original definition of Communications_Type in the P&C schema:

```
   <xsd:complexType name="Communications_Type">
      <xsd:sequence>
         <xsd:element ref="PhoneInfo" minOccurs="0"
               maxOccurs="unbounded"/>
         <xsd:element ref="EmailInfo" minOccurs="0"
               maxOccurs="unbounded"/>
         <xsd:element ref="WebsiteInfo" minOccurs="0"
               maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="ID"/>
   </xsd:complexType>
```

If I want to restrict the content I have to do the following:

```
<xsd:schema targetNamespace="http://www.acord.org/xml/v1.2”
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.acord.org/xml/v1.2">

 <xsd:redefine
  schemaLocation="acord.xsd">

     <!-- Remove the EmailInfo aggregate and the id attribute -->

   <xsd:complexType name="Communications_Type">
      <xsd:sequence>
         <xsd:element ref="PhoneInfo" minOccurs="0"
               maxOccurs="unbounded"/>
         <xsd:element ref="WebsiteInfo" minOccurs="0"
               maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>

 </xsd:redefine>

</xsd:schema>
```

Note: To restrict the content, I have had to replicate the definition from the original schema minus those objects I wanted to remove. If you have a static schema this might not be too bad, but if this complexType is updated in the next release you have to remember to update you r definition here; assuming you want the new elements or attributes. This might be easy to track for a few restrictions on simple content like this, but suppose you had a complexType with 100 elements and you remove 2 of them at random locations.

Here is another example of restricting the content of a group. This is the original P&C specification:

```
<xsd:group name="ACORDREQ">
   <xsd:sequence>
      <xsd:element ref="SignonRq" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="SuppressNotificationInd" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="SendSPRequestsInd" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="BaseSvcRq" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="ExtensionsSvcRq" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="InsuranceSvcRq" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="SuretySvcRq" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="ClaimsSvcRq" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="AccountingSvcRq" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="SignoffRq" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:group>
```

Now lets redefine it by removing a couple of the elements:

```
<xsd:schema targetNamespace="http://www.acord.org/xml/v1.2”
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.acord.org/xml/v1.2">

 <xsd:redefine
  schemaLocation="acord.xsd">

     <!-- Remove the elements BaseSvcRq and ClaimsSvcRq -->
   <xsd:group name="ACORDREQ">
      <xsd:sequence>
         <xsd:element ref="SignonRq" minOccurs="1" maxOccurs="1"/>
         <xsd:element ref="SuppressNotificationInd" minOccurs="0" maxOccurs="1"/>
         <xsd:element ref="SendSPRequestsInd" minOccurs="0" maxOccurs="1"/>
         <xsd:element ref="ExtensionsSvcRq" minOccurs="0" maxOccurs="1"/>
         <xsd:element ref="InsuranceSvcRq" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="SuretySvcRq" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="AccountingSvcRq" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="SignoffRq" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:group>

 </xsd:redefine>

</xsd:schema>
```

Again we end up replicating the original definition just so we can restrict the use of a couple of elements. This situation applies to attribute groups and code lists (enumerated types). For simple changes or stable environments this might not be a big issue, but for a growing and dynamically changing environment this could become a burden to support.

3.6.1 Pros

1. Extensions and restrictions are consistently applied to every use of one of the supported objects. It redefines the types rather than making a new class that can be used in its place.
2. Requires the use of another file to manage the modifications and use xsd:redefine.

3. Modifications are portable.
4. Easy to find the redefined objects and their content, no surprises in the data stream.
5. No modifications required in the data stream to support this functionality.

3.6.2 Cons

1. Like other methods it does not uniquely identify the fact a modified schema is being used.
2. New elements could be added to the namespace without identifying an outside schema and namespace.
3. To redefine an element content it has to be based upon a type and not contain a local definition. This forces an extra complexType to be created to support this functionality.
4. Restriction requires the recreation of the content of a complexType.
5. Modification of a group or attributeGroup requires the recreation of the object.

3.7 Adapter Schema Methodology

The adapter schema methodology is a 3 (maybe 2) file approach to modifying an existing schema with customer unique elements, attributes, data types, and code lists. Below is a simple illustration of how these files are configured.

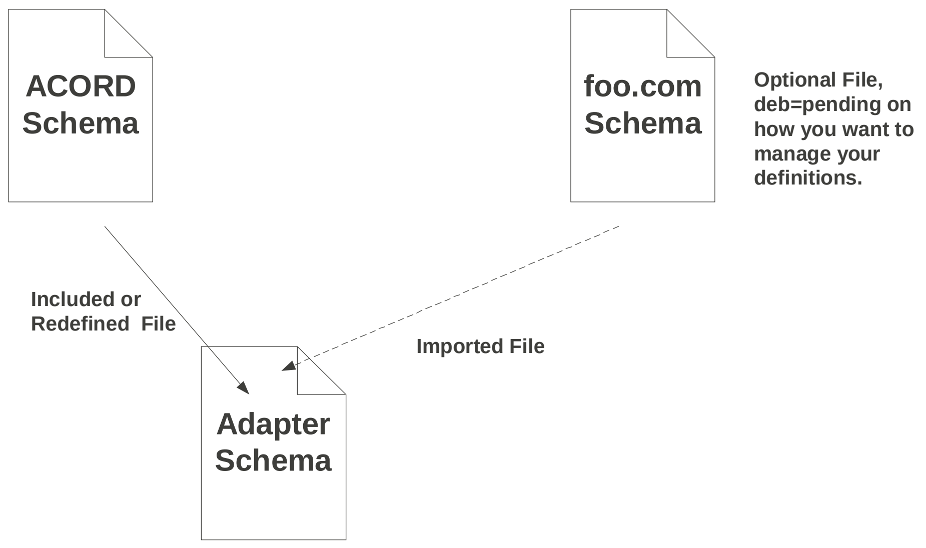
ACORD Schema
foo.com Schema
Optional File, deb=pending on how you want to manage your definitions.
Included or Redefined File
Imported File
Adapter Schema

Figure 1 Adapter Schema

In Figure 1, the ACORD schema is as it is published. No modifications would be made in this file. The foo.com schema or file would be used to manage your new elements and data types that will be added to the ACORD schema. You could manage these objects in the adapter schema, but it would make reuse of these elements a little more difficult.

Given this configuration, you can easily move or reuse your element definitions and by simply changing the ACORD schema file, you would be updated to the latest version of the specification. Also, this design is not completely dependant on the extension method you choose to use, this is just a good way to manage your extensions outside the original ACORD schema. In the review of the Redefine process you have an example of a two file approach. The second file is described in this section, with the first file being the original schema.

In addition to xsd:redefine, the W3C Schema specification provides two other mechanisms for building schemas from multiple files or sources. These are the xsd:include and xsd:import elements.

Schema Syntax – xsd:include

```
<include
  id = ID
  schemaLocation = anyURI
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</include>
```

The effect of this xsd:include element is to bring in the definitions and declarations contained in included schema, and make them available as part of the including schema target namespace. The one important caveat to using xsd:include is that the target namespace of the included components must be the same as the target namespace of the including schema. Bringing in definitions and declarations using the xsd:include mechanism effectively adds these components to the existing target namespace

Schema Syntax – xsd:import

```
<import
  id = ID
  namespace = anyURI
  schemaLocation = anyURI
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</import>
```

The import mechanism is an important mechanism that enables schema components from different target namespaces to be used together, and hence enables the schema validation of instance content defined across multiple namespaces

When schema components are imported from multiple namespaces, each namespace must be identified with a separate xsd:import element. The xsd:import elements themselves must appear as the first children of the xsd:schema element. Furthermore, each namespace must be associated with a prefix, using a standard namespace declaration, and that prefix is used to qualify references to any schema components belonging to that namespace. Finally, xsd:import elements optionally contain a schemaLocation attribute to help locate resources associated with the namespaces.

Example of type derivation using adapter files:

Here is the adapter schema file contents:

```
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:acord="http://www.ACORD.org/xml/"
         xmlns="http://www.foo.com/"
         targetNamespace="http://www.foo.com/">

    <xs:import namespace="http://www.ACORD.org/xml/"
               schemaLocation="acord.xsd"/>

    <xs:element name="EducationLevel" type="EducationLevelType"/>
    <xs:simpleType name="EducationLevelType">
        <xs:restriction base="xs:string">
              <xs:enumeration value="HighSchool"/>
              <xs:enumeration value="College"/>
              <xs:enumeration value="GraduateDegree"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="PersonInfoType">
        <xs:complexContent>
              <xs:extension base="acord:PersonInfoType">
                    <xs:sequence>
                          <xs:element ref="EducationLevel"/>
                    </xs:sequence>
              </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

```
</xs:schema>
```

In this example we are using type derivation to create a substitute element content for anything based upon the acord:PersonInfoType. We also create a new element and data type for that element which is then added (extension) to the acord:PersinInfoType. In this case we import the ACORD schema and create a new namespace for the hybrid schema that is the combination of ACORD and these extensions for the Foo company.

Note: This shows a two file adapter schema, we could have split the Foo definitions into a third file so it could be reused, but this was a short example.

The data stream that then uses this schema looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<ACORD xmlns="http://www.ACORD.org/xml/"
     xmlns:foo="http://www.foo.com/xml/"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.foo.com/xml/ fooAcord.xsd
                          http://www.ACORD.org/xml/ acord.xsd">

     ...
               <DriverInfo>
                  <PersonInfo xsi:type="foo:PersonInfoType">
                     <GenderCd>M</GenderCd>
                     <BirthDt>1972-06-24T23:21:44-05:00</BirthDt>
                     <MaritalStatusCd>M</MaritalStatusCd>
                     <foo:EducationLevel>GraduateDegree</foo:EducationLevel>
                  </PersonInfo>
               </DriverInfo>
     ...

</ACORD>
```

The data stream now references the original ACORD namespace and schema, as well as the new hybrid schema (combination of Foo and ACORD elements). Later in the stream we find the use of the ACORD PersonInfoType and it uses the xsi:type attribute to indicate that it isn’t the standard ACORD type, but is a derived type. Based upon just reading the header on this data stream there is nothing here to help you understand what sort of extension is being used. The appearance of multiple namespaces is one clue that something different is happening but you don’t know what.

Example of substitution groups using adapter files:

Here is the content in the ACORD schema:

```
<xsd:complexType name="PersonInfoType">
     <xsd:sequence>
          <xsd:element ref="GenderCd" minOccurs="0"/>
          <xsd:element ref="BirthDt" minOccurs="0"/>
          <xsd:element ref="OpenMaritalStatusCd" minOccurs="0"/>
          <xsd:element ref="OccupationDesc" minOccurs="0"/>
                … etc.
     </xsd:sequence>
     <xsd:attribute name="id" type="ID"/>
</xsd:complexType>

<xsd:complexType name="MaritalStatusCdType">
     <xsd:simpleContent>
          <xsd:extension base="OpenEnum"/>
     </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="ACORDMaritalStatusCdType">
     <xsd:simpleContent>
          <xsd:restriction base="MaritalStatusCdType">
```

```
                <xsd:enumeration value="D"/>
                <xsd:enumeration value="M"/>
                <xsd:enumeration value="P"/>
                <xsd:enumeration value="S"/>
                <xsd:enumeration value="U"/>
                <xsd:enumeration value="W"/>
            </xsd:restriction>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:element name="OpenMaritalStatusCd" type="MaritalStatusCdType"
    abstract="true"/>
    <xsd:element name="MaritalStatusCd" type="ACORDMaritalStatusCdType"
                                     substitutionGroup="MaritalStatus"/>
```

What is highlighted in blue in the above sample are all the pieces required to make a substitution group work. This is based upon abstract elements. First we have the definition of the MaritalStatusCdType that is the base type for the dummy element OpenMaritalStatusCd. Remember that this acts as a placeholder in the schema definition. It is the head element of the substitution group MaritalStatus. The OpenMaritalStatusCd element will never appear in the data stream and if no extensions are ever made you would only see the MaritalStatusCd element. These are the hooks required to be built into the original schema to allow someone to extend it with a substitution group.

Here is the adapter schema file contents:

```
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:acord="http://www.ACORD.org/xml/"
            xmlns="http://www.foo.com/"
            targetNamespace="http://www.foo.com/">

        <xs:import namespace="http://www.ACORD.org/xml/"
                schemaLocation="acord.xsd"/>

        <xsd:complexType name="MaritalStatusCdType">
            <xsd:simpleContent>
                <xsd:restriction base="acord:MaritalStatusCdType">
                    <xsd:enumeration value="L"/>
                    <xsd:enumeration value="A"/>
                </xsd:restriction>
            </xsd:simpleContent>
        </xsd:complexType>

        <xsd:element name="MaritalStatusCd“ type=“MaritalStatusCdType”
                substitutionGroup="acord:MaritalStatus"/>

    </xs:schema>
```

The data stream that then uses this schema looks like this:

```
    <?xml version="1.0" encoding="UTF-8"?>
    <ACORD xmlns="http://www.ACORD.org/xml/"
        xmlns:foo="http://www.foo.com/xml/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.foo.com/xml/ fooAcord.xsd
                            http://www.ACORD.org/xml/ acord.xsd">

        ...

        <PersonInfo>
            <GenderCd>M</GenderCd>
            <MaritalStatusCd>S</MaritalStatusCd>
            <OccupationClassCd>PRO</OccupationClassCd>
            <LengthTimeEmployed>
            <NumUnits>7</NumUnits>
            <UnitMeasurementCd>Y</UnitMeasurementCd>
            </LengthTimeEmployed>
        </PersonInfo>

        ...

        <PersonInfo>
            <GenderCd>M</GenderCd>
            <foo:MaritalStatusCd>L</foo:MaritalStatusCd>
            <OccupationClassCd>PRO</OccupationClassCd>
            <LengthTimeEmployed>
            <NumUnits>7</NumUnits>
            <UnitMeasurementCd>Y</UnitMeasurementCd>
            </LengthTimeEmployed>
        </PersonInfo>

        ...

    </ACORD>
```

Here the substation group allows me to use the either the acord:MaritalStatusCd element or the foo:MaritalStatusCd element. There is nothing other than the namespace declaration and use that has been added to the data stream, but a significant amount of work went into making the schema ready to use this approach.

3.7.1 Pros

1. Provides for the portability and maintainability of extensions using any of the methods discussed in this document.

3.7.2 Cons

1. Requires the management of additional files to create a single schema.

4 CONTROLLING EXTENSION OF SIMPLE TYPES

Previously, we discussed how substitution by type derivation could be controlled with the block and blockFinal attributes. The W3C schema spec provides a set of features for controlling how simpleTypes can be restricted. Remember a simpleType can only be restricted, there is no way to extend a simpleType.

Schema Syntax – xsd:simpleType

```
<simpleType
  final = (#all | (list | union | restriction))
  id = ID
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (restriction | list | union))
</simpleType>
```

The xsd:simpleType provides the final attribute to specify what sort of derivation you want to disallow. The final attribute has the following values:

- #all – disallow union, list and further restriction.
- union – disallow the type from being used in a union
- restriction – disallow further restriction of the facets already defined. If maxLength is set, maxLength cannot be further restricted.
- list – disallow the type from being used in a list

Note: The finalDefault attribute on the xsd:schema element is set, it will apply here as well.

5 SOME NOTES ON EXTENSION AND RESTRICTION

W3C Schema based extension allows you to add new elements or content. For elements, they are added at the end of the original definition. See the next section for issues concerning code lists or enumerated types.

W3C Schema based restriction requires that the end result of a restriction must still be valid based upon the original definition. This has the following implications:

- Optional elements can be removed
- Optional elements cannot be made required
- Default values can be provided
- Values can be fixed
- An max limit can be set to something smaller

To restrict and extend a type you must do this in two steps

Some of the features described in this document provide for an element to have different content models. We do not consider this a true extension mechanism.

6 SUMMARY

The following table is based upon the above methods being used to manage element content.

Methodology	**Portable**	**Validate**	**Human Manageable**	**Use of extended schema obvious**	**Extension or Substitution Method**	**Supports**	**Setup in Schema**	**Changes in Data Stream**
Direct editing of ACORD Schema	No	Yes depending on features used	Yes	No	Anything	Anything is possible	Yes	No
Schema Wildcards	Yes	No	Yes	No	Extension only	Addition of elements or attributes with no consistent usage	Yes	Yes
Derived Types	Yes	Yes	No	No	Extension and Restriction	Element content that varies with each use of the parent element	No	Yes
Substitution Group	Yes	Yes	No	No	Substitution	Element content that varies with each use of the parent element	No	No
Abstract elements	Yes	Yes	No	No	Substitution	Element content that varies with each use of the parent element	Yes	No
Abstract types	Yes	Yes	No	No	Extension	Element content	Yes	Yes

Methodology	Portable	Validate	Human Manageable	Use of extended schema obvious	Extension or Substitution Method	Supports	Setup in Schema	Changes in Data Stream
						that varies with each use of the parent element		
Redefine	Yes	Yes	Yes	No	Extension	Extension Restriction of all uses	No	No

Definitions:

Methodology – one of the methods described in this document. In most cases this requires the use of the adapter schema to make some columns true.

Portable – The ability to easily move to the next version of the specification

Validate – The resulting schema can be used to control the content of elements and validate that they are allowed and have the proper content.

Human Manageable – Is it easy for a human to track all the relationships in the schema and recognize where content models may change.

Use of extended schema obvious – Does the given methodology expose the fact that you are no longer referencing the ACORD standard schema and that there are extension elements involved? This is based upon looking at the first few lines of the data stream and not looking for tail tell signs like xsi:type that might not appear in a given data stream.

Extension or Substitution Method – Is this an extension capability or a way to allow an element to have various content models

Supports -

Setup in Schema – Does the method require any special considerations in the schema before it can be used? This is based upon the assumption that all elements have their content defined with a base type.

Changes in Data Stream – Does the method require new attributes in the data stream? If it does, these may or may not be backward compatible with the DTD. For instance if it requires multiple namespaces to be declared in the data stream, this is problematic or at least requires a controlled number and set prefixes.