
# By Harish Kamath

# Table of Contents

# The Road Ahead

In the first part of this article, I spent a fair amount of time explaining the difference between simple and complex types in an XML schema, demonstrating, with examples, how to go about building both. I also showed you how complex types can be extended to create new sub−types, and how this extensibility allows you to add OO−like capabilities to your XML schema. Finally, I wrapped things up with a quick look at how you can make your various schema definitions more maintainable by organizing them into separate files.

In this article, I'll be continuing the discussion of type extension, demonstrating how to derive new types by restricting (rather than extending) existing ones, create abstract definitions and redefine existing types. Note that you'll need to be up to speed on the material covered in the first part of this article in order to understand the concepts discussed in this one − so if you're coming at this from scratch, take some time out to get the basics down, and then flip the page so we can get started.

# Feeling The Force

In "The Phantom Menace", when Qui−Gon Jinn first meets Anakin Skywalker on an unscheduled stop at the Outer Rim world of Tatooine, he immediately realizes that there is something special about the boy: he is destined to be a Jedi.

Or is he?

"The Phantom Menace" and subsequent episodes certainly resolve that question conclusively − but what does that have to do with XML schemas?

Quite a lot, as you'll see shortly. First, though, let's quickly revisit a segment from the first part of this article, in which I defined a complex data type named "starWarsEntity",

```
<?xml version="1.0" encoding="UTF−8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="starWarsEntity">
<xsd:sequence>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="species" type="xsd:string"/>
<xsd:element name="language" type="xsd:string"/>
<xsd:element name="home" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

<!−− more definitions −−>

</xsd:schema>
```

and then extended it to create a new "Human" sub−type.

```
<?xml version="1.0" encoding="UTF−8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="Human">
<xsd:complexContent>
<xsd:extension base="starWarsEntity">
<xsd:sequence>
```

```
<xsd:element name="gender"
type="xsd:string"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<!-- more definitions -->

</xsd:schema>
```

Let's now extend things a little further, and derive one more datatype from the "Human" complex type − we'll call this one "Jedi", and make sure that it inherits all the characteristics of the "Human" type, together with one additional attribute. Here's the definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="Jedi">
<xsd:complexContent>
<xsd:extension base="Human">
<xsd:sequence>
<xsd:element
name="midichlorian-count" type="xsd:integer"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<!-- more definitions -->

</xsd:schema>
```

In case you're wondering − midi−chlorians are microscopic symbiotic creatures that swim around in the bloodstream of creatures in the Star Wars universe. In Star Wars lore, every Jedi possesses an unusually large number of these creatures, which are the reason they are so receptive to the Force; in "The Phantom Menace", Qui−Gon Jinn uses this midi−chlorian count to determine Anakin's eligibility for Jedi−hood.

Here's an XML file built around this schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<gallery
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<character xsi:type="Jedi">
<name>Luke Skywalker</name>
<species>Human</species>
<language>Basic</language>
<home>Tatooine</home>
<gender>Male</gender>
<midichlorian-count>9000</midichlorian-count>
</character>
</gallery>
```

What I just showed you was an example of deriving, by extension, one new type from another existing type, which is itself derived from a base type. This type of extensibility is what makes XML Schemas so powerful...and so easy to maintain.

# The Next Level

Now, deriving a new datatype by extending the characteristics of an existing one is just one way of getting the job done. The XML Schema specification also supports one other way of deriving new datatypes: by restricting, or constraining, the characteristics of existing types.

In order to understand this, let's go back to the analogy on the previous page, and consider using the "Jedi" datatype as the base for another datatype: "JediMaster" (according to http://www.starwars.com/databank/organization/thejediorder/index.html, Jedi Masters are "those who have shown exceptional devotion and skill in the Force.")

In other words, a "JediMaster" possesses all the characteristics of a "Jedi"...with one additional constraint: a very high midi−chlorian count. Therefore, it is possible to define a "JediMaster" datatype simply by adding a restriction to the definition of the "Jedi" datatype − as demonstrated below:

---

```
<?xml version="1.0" encoding="UTF−8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!−− other definitions −−>

<xsd:complexType name="JediMaster">
<xsd:complexContent>
<xsd:restriction base="Jedi">
<xsd:sequence>
<xsd:element name="name"
type="xsd:string"/>
<xsd:element name="species"
type="xsd:string"/>
<xsd:element name="language"
type="xsd:string"/>
<xsd:element name="home"
type="xsd:string"/>
<xsd:element name="gender"
type="xsd:string"/>
<xsd:element
name="midichlorian−count" >
<xsd:simpleType>
<xsd:restriction
base="xsd:integer">

<xsd:minInclusive value="10000" />

</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
```

```
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>

</xsd:schema>
```

Want to verify that I'm speaking the truth? Take the XML sample on the previous page, alter it to use the new "JediMaster" datatype, and pass it through an XML validator.

```
<?xml version="1.0" encoding="UTF-8"?>
<gallery
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<character xsi:type="JediMaster">
<name>Luke Skywalker</name>
<species>Human</species>
<language>Basic</language>
<home>Tatooine</home>
<gender>Male</gender>
<midichlorian-count>9000</midichlorian-count>
</character>
</gallery>
```

You should see an error, since the value of the <midiclorian−count> element is less than the defined minimum value in the "JediMaster" datatype.

Now, alter the value of the <midiclorian−count> element in your XML sample, and validate it again.

```
<?xml version="1.0" encoding="UTF-8"?>
<gallery
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<character xsi:type="JediMaster">
<name>Luke Skywalker</name>
<species>Human</species>
<language>Basic</language>
<home>Tatooine</home>
<gender>Male</gender>
<midichlorian-count>18000</midichlorian-count>
</character>
</gallery>
```

This time, you should see no errors − indicating that the restriction you've imposed while deriving the new datatype is in effect.

Note also that when deriving by restriction, it is necessary to repeat the definition of all elements in the derived complex type from my original type. As a result, all elements of type "JediMaster" will also be acceptable as elements of type "Jedi".

# Big Brother Is Watching...

It's also possible to control whether or not schema designers can derive new datatypes from existing one, simply by specifying the level of extensibility allowed in the base definition. This is accomplished via the "final" attribute, which can accept any one of three values: "restriction", "extension" and "#all". Consider the following example, which illustrates:

```
<xsd:complexType name="Human" final="#all">
<xsd:complexContent>
<xsd:extension base="starWarsEntity">
<xsd:sequence>
<xsd:element name="gender"
type="xsd:string"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

In this case, I've specified that new types may *not* be derived from this type by extending or restricting it, via the "final" attribute. In order to verify that this works, try deriving a new type by extension and using that derived type in a document instance − your XML validator should display an error.

This ability to control the extent to which a base type can be used for further type derivations is very important when you're creating layered schema definitions; it can provide schema authors with an easy, efficient way to restrict misuse or erroneous use of a type definition, especially in the case of schemas which are widely used by many different applications.

You can specify a default "final" value for *all* the datatypes in a schema via the special "finalDefault" attribute, which must be included within the outermost <xsd:schema> element. In order to illustrate, consider the following code snippet, which allows derivation of new types from all existing definitions by restriction only.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
finalDefault="extension">

<!−− definitions −−>

</xsd:schema>
```

# Speaking In The Abstract

While on the subject of controlling the manner in which type definitions can be used, it's instructive to also look at abstract type definitions. If a base type spawns several new sub−types, as in the example below,

```
<?xml version="1.0" encoding="UTF−8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="gallery">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="character"
type="starWarsEntity" maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<!−− base definition −−>
<xsd:complexType name="starWarsEntity">
<xsd:sequence>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="species" type="xsd:string"/>
<xsd:element name="language" type="xsd:string"/>
<xsd:element name="home" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

<!−− derived definition −−>
<xsd:complexType name="Ewok">
<xsd:complexContent>
<xsd:extension base="starWarsEntity">
<xsd:sequence>
<xsd:element name="vehicle"
type="xsd:string"/>
<xsd:element name="society"
type="xsd:string"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Human">
<xsd:complexContent>
<xsd:extension base="starWarsEntity">
<xsd:sequence>
<xsd:element name="gender"
type="xsd:string"/>
```

```
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<!-- and so on -->

</xsd:schema>
```

schema authors can force document authors to be more precise in their usage of these types by declaring the base type as abstract.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- base definition -->
<xsd:complexType name="starWarsEntity" abstract="true">
<xsd:sequence>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="species" type="xsd:string"/>
<xsd:element name="language" type="xsd:string"/>
<xsd:element name="home" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

This requires document authors to specifically name the sub-type whenever they use it in a document instance. Failure to do so will result in XML validation errors. For example, while the following XML document instance is certainly conformant to the rules laid down for the base type "starWarsEntity",

```
<?xml version="1.0" encoding="UTF-8"?>
<gallery
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<character>
<name>Luke Skywalker</name>
<species>Human</species>
<language>Basic</language>
<home>Tatooine</home>
</character>
</gallery>
```

the XML validator will still generate errors while parsing it, as "starWarsEntity" has been defined as an abstract type. It is only when the document author specifies a type via the "type" attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<gallery
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<character xsi:type="Human">
<name>Luke Skywalker</name>
<species>Human</species>
<language>Basic</language>
<home>Tatooine</home>
<gender>Male</gender>
</character>
</gallery>
```

that validation will take place without errors.

Again, this mechanism assists in reducing the risk of errors, and in controlling the manner in which schema definitions are used by document authors. It's also possible to declare specific elements (rather than types) as abstract − all you need is a substitution group, which you can read about at http://www.w3.org/TR/xmlschema-0/#SubsGroups

# Going Local

You'll remember, from the first article in this series, that the XML Schema specification allows you to split up schema definitions across multiple files, and include one file within another using the <xsd:include> element. This not only helps in logically separating base and derived definitions, it also makes your code cleaner and easier to maintain.

In case you're using a schema published by an external party, it's quite possible that you may need to make changes to it in order to tailor it to your specific requirements. Making changes to the original schema definitions is not always the best way to accomplish this task; sometimes, it's more logical to leave the original schema definitions as is, and simply over−ride them with new definitions where needed.

The XML Schema specification allows you to do this via the <xsd:redefine> element, which makes it possible to easily redefine an existing schema definition. In order to illustrate this, let's return to the last example in the first part of this article, in which I split up my schema definitions into "base−defs.xsd", which contained the base definitions,

```
<?xml version="1.0" encoding="UTF−8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!−− base definitions −−>
<xsd:complexType name="starWarsEntity">
<xsd:sequence>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="species" type="xsd:string"/>
<xsd:element name="language" type="xsd:string"/>
<xsd:element name="home" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

<xsd:element name="gallery">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="character"
type="starWarsEntity" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:schema>
```

and "derived−defs.xsd", which referenced "base−defs.xsd" and contained the extensions.

```
<?xml version="1.0" encoding="UTF−8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<!-- include base types-->
<xsd:include schemaLocation="base-defs.xsd"></xsd:include>

<!-- derived types -->
<xsd:complexType name="Human">
<xsd:complexContent>
<xsd:extension base="starWarsEntity">
<xsd:sequence>
<xsd:element name="gender"
type="xsd:string"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Jedi">
<xsd:complexContent>
<xsd:extension base="Human">
<xsd:sequence>
<xsd:element
name="midichlorian-count" type="xsd:integer"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="JediMaster">
<xsd:complexContent>
<xsd:restriction base="Jedi">
<xsd:sequence>
<xsd:element name="name"
type="xsd:string"/>
<xsd:element name="species"
type="xsd:string"/>
<xsd:element name="language"
type="xsd:string"/>
<xsd:element name="home"
type="xsd:string"/>
<xsd:element name="gender"
type="xsd:string"/>
<xsd:element
name="midichlorian-count">
<xsd:simpleType>
<xsd:restriction
base="xsd:integer">

<xsd:minInclusive value="10000" />
```

```
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>

... and so on ...

</xsd:schema>
```

Finally, my XML document instance itself referenced the schema definitions in "derived−defs.xsd".

```
<?xml version="1.0" encoding="UTF−8"?>
<gallery xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:noNamespaceSchemaLocation="derived−defs.xsd">

...

</gallery>
```

Now, let's assume I wanted to redefine the new "JediMaster" type included in "derived−defs.xsd" to include one more attribute. I could put this (re)definition in a file called "local−defs.xsd",

```
<?xml version="1.0" encoding="UTF−8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!−− include and redefine derived types−−>
<xsd:redefine schemaLocation="derived−defs.xsd">

<xsd:complexType name="JediMaster">
<xsd:complexContent>
<xsd:extension base="JediMaster">
<xsd:sequence>
<xsd:element
name="weapon" type="xsd:string"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

</xsd:redefine>

</xsd:schema>
```

and reference this new file in my XML document instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<gallery xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="local-defs.xsd">

<character xsi:type="JediMaster">
<name>Luke Skywalker</name>
<species>Human</species>
<language>Basic</language>
<home>Tatooine</home>
<gender>Male</gender>
<midichlorian-count>38000</midichlorian-count>
<weapon>lightsaber</weapon>
</character>

... and so on ...

</gallery>
```

The XML validator will now use the new definition of the "JediMaster" class when validating this document instance, rather than the original definition in "local-defs.xsd". Other types may be redefined in a similar manner.

This ability to selectively redefine schema elements makes it possible to easily "localize" an externally-provided set of schema definitions to specific needs, without damaging or altering the original.

And that's about it for the moment. In the next part of this article, I'll be looking at uniqueness, keys and references. Lotsa good stuff ahead - so make sure you tune in!

Note: All examples in this article have been tested on Linux/i586. Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!