Should it be an Element or a Type?

Table of Contents

Issue

When should an item be declared as an element versus when should it be defined as a type?

Introduction

This issue is best discussed by way of example:

Example

```
Should “Warranty” be declared as an element:
    <xsd:element name=“Warranty”>
      ...
    </xsd:element>
or as a type:
    <xsd:complexType name=“Warranty”>
            ...
    </xsd:complexType>
```

Best Practice

[1] When in doubt, make it a type. You can always create an element from the type, if needed. With a type, other elements can reuse that type.

Example. If you canít decide whether to make Warranty an element or a type, then make it a type:

```
    <xsd:complexType name=“Warranty”>
       ...
    </xsd:complexType>
```

If you decide later that you need a Warranty element, you can create one using the Warranty type:

```
<xsd:element name="Warranty" type="Warranty"/>
```

*Recall that elements and types are in different **Symbol Spaces**. Hence, you can have an element and type with the same name.*

[2] If the item is not intended to be an element in instance documents then define it as a type.

Example. If you will *never* see this in an instance document:

```
<Warranty>
 ...
</Warranty>
```

then define Warranty as a complexType.

[3] If the itemís content is to be reused by other items then define it as a type.

Example. If other items need to use Warrantyís content, then define Warranty as a type:

```
<xsd:complexType name="Warranty">
   ...
</xsd:complexType>
...
<xsd:element name="PromissoryNote" type="Warranty"/>
<xsd:element name="AutoCertificate" type="Warranty"/>
```

The example shows two elements - PromissoryNote and AutoCertificate - reusing the Warranty type.

[4] f the item is intended to be used as an element in instance documents, and itís required that sometimes it be nillable and other times not, then it must be defined it as a type.

Example. Letís first see how **not** to do it. Suppose that we create a Warranty element:

```
<xsd:element name="Warranty">
  ...
</xsd:element>
```

The Warranty element can be reused elsewhere by refíing it:

```
<xsd:element ref="Warranty"/>
```

Suppose that we also need a version of Warranty that supports a nil value. You might be tempted to do this:

```
<xsd:element ref="Warranty" nillable="true"/>
```

This is not legal. This dynamic morphing capability (i.e., reusing a Warranty element declaration while simultaneously adding nillability) cannot be achieved using elements. The reason for this is that the ref and nillable attributes are mutually exclusive - you can use ref, or you can use nillable, but not both. The only way to accomplish the dynamic morphing capability is by defining Warranty as a type:

```
<xsd:complexType name="Warranty">
  ...
</xsd:complexType>
```

and then reusing the type:

```
<xsd:element name="Warranty" nillable="true" type="Warranty"/>
...
<xsd:element name="Warranty" type="Warranty"/>
```

In the first case Warranty is nillable. In the second case itís not nillable.

[5] If the item is intended to be used as an element in instance documents and other elements are to be allowed to substitute for the element, then it must be declared as an element.

Example. Suppose that we would like to enable instance document authors to use interchangeably the vocabulary (i.e., tag name) Warranty, Guarantee, or Promise, i.e.,

```
<xsd:Warranty>
   ...
</xsd:Warranty>
   ...
<xsd:Guarantee>
   ...
</xsd:Guarantee>
   ...
<xsd:Promise>
     ...
</xsd:Promise>
```

To enable this substitutable-tag-name capability, Warranty, Guarantee, and Promise must be declared as elements, and made members of a substitutionGroup:

```
<xsd:element name="Warranty">
   ...
</xsd:element>
<xsd:element name="Guarantee" substitutionGroup="Warranty"/>
<xsd:element name="Promise" substitutionGroup="Warranty"/>
```