## INTRODUCTION AUX DÉFINITIONS de document XML, AUX DTD et aux XSD

Avant les vacances, nous avons commencé à explorer ce qu'était le XML.

Le XML est rarement utilisé seul et on préfère souvent l'utiliser en parallèle d'autres technologies permettant d'étendre ses possibilités de la technologie. Nous allons aborder les outils qui permettent de définir une structure stricte aux documents XML : les **fichiers de définition**.

Il existe plusieurs méthodes de création de fichiers de configuration, nous verrons principalement les **DTD** et les fichiers **XSD** (XML Schema Definition).

### Qu'est-ce que la définition d'un document XML ?

Une **définition d'un document XML** est un ensemble de règles que l'on impose au document. Ces règles permettent de décrire la façon dont le document XML doit être construit. Elles peuvent être de natures différentes. Par exemple, ces règles peuvent imposer la présence d'un attribut ou d'une balise, imposer l'ordre d'apparition des balises dans le document ou encore, imposer le type d'une donnée (nombre entier, chaîne de caractères, etc.).

##### Un document valide

Dans la partie précédente, nous avons vu ce qu'était un document **bien formé**. Cette seconde partie est l'occasion d'aller un peu plus loin et de voir le concept de document **valide**.

**Un document valide est un document bien formé conforme à une définition.** Cela signifie que le document XML respecte toutes les règles qui lui sont imposées dans les fameuses définitions.

#### Pourquoi écrire des définitions ?

Associer une définition à un document oblige à une certaine rigueur dans l'écriture des données XML. C'est d'autant plus important lorsque plusieurs personnes travaillent sur un même document. La définition impose ainsi une écriture uniforme que tout le monde doit respecter. On évite ainsi que l'écriture d'un document soit anarchique et, par conséquent, difficilement exploitable.

*Exploitable par quel acteur et pour quels usages ?*

Le plus souvent, par un programme informatique.

Le cas typique est celui d' un **programme** informatique qui traite les **données** contenues dans un **document** **XML** respectant une **définition** donnée. Imposer une définition aux documents que votre programme exploite permet d'assurer un automatisme et un gain de temps précieux :

* Le document n'est pas valide : le programme s'arrête et ne cherche pas à exploiter le fichier.
* Le document est valide : le programme peut "parser" et exploiter le fichier.

## Quelques notions sur les DTD

### Définition d'une DTD

#### Une définition rapide

Une **Document Type Definition** ou en français une **Définition de Type de Document**, souvent abrégé **DTD**, est la principale technologie pour écrire les définitions de nos documents XML.

L'objectif est d'écrire une définition des documents XML, c'est-à-dire, de construire un ensemble de règles qui vont régir la construction du document XML.

Grâce à l'ensemble de ces règles, nous allons ainsi définir l'architecture de notre document XML et la hiérarchie qui existe entre les balises de celui-ci. Ainsi, on pourra préciser l’enchaînement et le contenu des balises et des attributs contenus dans le document XML.

Finalement, sachez qu'avec les DTD, vous ne pourrez pas toujours tout faire, la technologie commençant en effet à vieillir. Mais comme elle est encore beaucoup utilisée, il est indispensable qu'elle soit étudiée dans ce tutoriel.

#### Où écrire les DTD ?

Tout comme les fichiers XML, les DTD s'écrivent dans des fichiers.

Nous reviendrons sur ce point un peu plus tard, mais sachez dès à présent qu'il existe 2 types de DTD : les **DTD externes** et les **DTD internes**.

Les règles des **DTD internes** s'écrivent directement dans le fichier XMLqu'elles définissent tandis que les règles des **DTD externes** sont écrites dans un fichier séparé portant l’extension **.dtd** .

### Les éléments

#### La syntaxe

Pour définir les règles portant sur les **balises**, on utilise le mot clef ELEMENT.

<!ELEMENT balise (contenu)>

Une règle peut donc se découper en 3 mots clefs : ELEMENT, balise et contenu.

Une règle peut donc se découper en 3 mots clefs : ELEMENT, balise et contenu.

#### Retour sur la balise

Le mot-clef balise est à remplacer par le nom de la balise à laquelle vous souhaitez appliquer la règle.  
Pour exemple, prenons la balise suivante:

<price>400</price>

On écrira alors :

<!ELEMENT price (contenu)>

#### Retour sur le contenu

Cet emplacement a pour vocation de décrire ce que doit contenir la balise : est-ce une autre balise ou est-ce une valeur ?

##### **Cas n°1 : Cas d'une balise en contenant une autre**

Par exemple, regardons la règle suivante :

<!ELEMENT book (price)>

<!-- suite de la DTD -->

Cette règle signifie que la balise <book /> contient la balise <price />.

Le document XML respectant cette règle ressemble donc à cela :

<book>

    <price>400</price>

</book>

Nous n'avons défini aucune règle pour la balise <price/>. Le document n'est, par conséquent, pas valide. **En effet, dans une DTD, il est impératif de décrire toutle document sans exception.** Des balises qui n'apparaissent pas dans la DTD ne peuvent pas être utilisées dans le document XML.

##### **Cas n°2 : Cas d'une balise contenant une valeur**

Dans le cas où notre balise contient une **valeur simple**, on utilisera le mot clef #PCDATA  
Une valeur simple désigne par exemple une chaîne de caractères, un entier, un nombre décimal, un caractère, etc.

En se basant sur l'exemple exemple précédent :

<book>

    <price>400</price>

</book>

nous avions déjà défini une règle pour la balise <book/> :

<!ELEMENT book (price)>

Nous pouvons maintenant compléter notre DTD en ajoutant une règle pour la balise <price/>. Par exemple, si l'on souhaite que cette balise contienne une valeur simple, on écrira :

<!ELEMENT price (#PCDATA)>

Au final, la DTD de notre document XML est donc la suivante :

<!ELEMENT book (price)>

<!ELEMENT price (#PCDATA)>

##### Cas d'une balise vide

Il est également possible d'indiquer qu'une balise ne contient rien grâce au mot-clef EMPTY.  
Prenons les règles suivantes :

<!ELEMENT book (price)>

<!ELEMENT price EMPTY>

Le document XML répondant à la définition DTD précédente est le suivant :

<book>

    <price />

</book>

À noter : lors de l'utilisation du mot clef EMPTY, l'usage des parenthèses n'est pas obligatoire !

##### Cas d'une balise pouvant tout contenir

Il nous reste un cas à voir : celui d'une balise qui peut tout contenir, c'est à dire, une autre balise, une valeur simple ou tout simplement être vide. Dans ce cas, on utilise le mot-clef ANY.

Prenons la règle suivante :

<!ELEMENT book (price)>

<!ELEMENT price ANY>

Les documents XML suivants sont bien valides :

<!-- valeur simple -->

<book>

    <price>400</price>

</book>

<!-- vide -->

<book>

    <price />

</book>

Bien que le mot-clef ANY existe, il est souvent déconseillé de l'utiliser afin de restreindre le plus possible la liberté de rédaction du document XML.

Comme pour le mot-clef EMPTY, l'usage des parenthèses n'est pas obligatoire pour le mot-clef ANY.

### Autres éléments

Il existe bien d'autres éléments de DTD pour définir un document XML.

## DTD : LES ATTRIBUTS ET LES ENTITÉS

Dans le chapitre précédent, nous avons vu comment décrire les **balises** de nos documents XML, mais ce n'est pas suffisant pour pouvoir décrire l'intégralité d'un **document XML**. En effet, une balise peut contenir ce qu'on appelle des **attributs**. Il convient donc de décrire les règles relatives à ces attributs.

### Les attributs

Dans le chapitre précédent, nous avons découvert la syntaxe permettant de définir des règles sur les **balises de nos documents XML**. Vous allez voir que le principe est le même pour définir des **règles à nos attributs**.

#### La syntaxe

Pour indiquer que notre règle porte sur un **attribut**, on utilise le mot clef ATTLIST. On utilise alors la syntaxe suivante :

<!ATTLIST balise attribut type mode>

Une règle peut donc se diviser en 5 mots clefs : ATTLIST, balise, attribut, type et mode.

Ci dessous en anglais les différentes possibilités:

**Declaring Attributes**

In the DTD, XML element attributes are declared with an ATTLIST declaration. An attribute declaration has the following syntax:

|  |
| --- |
| <!ATTLIST element-name attribute-name attribute-type default-value> |

element\_name correspond au nom de la balise.

As you can see from the syntax above, the ATTLIST declaration defines the element which can have the attribute, the name of the attribute, the type of the attribute, and the default attribute value.

The **attribute-type** can have the following values:

|  |  |
| --- | --- |
| **Value** | **Explanation** |
| CDATA | The value is character data |
| (eval|eval|..) | The value must be an enumerated value |
| ID | The value is an unique id |
| IDREF | The value is the id of another element |
| IDREFS | The value is a list of other ids |
| NMTOKEN | The value is a valid XML name |
| NMTOKENS | The value is a list of valid XML names |
| ENTITY | The value is an entity |
| ENTITIES | The value is a list of entities |
| NOTATION | The value is a name of a notation |
| xml: | The value is predefined |

The **attribute-default-value** can have the following values:

|  |  |
| --- | --- |
| **Value** | **Explanation** |
| #DEFAULT value | The attribute has a default value |
| #REQUIRED | The attribute value must be included in the element |
| #IMPLIED | The attribute does not have to be included |
| #FIXED value | The attribute value is fixed |

**Attribute declaration example**

|  |
| --- |
| DTD example:  <!ELEMENT square EMPTY>  <!ATTLIST square width CDATA "0">  XML example:  <square width="100"></square> |

In the above example the element square is defined to be an empty element with the attributes width of  type CDATA. The width attribute has a default value of 0.

**Default attribute value**

|  |
| --- |
| Syntax:  <!ATTLIST element-name attribute-name CDATA "default-value">  DTD example:  <!ATTLIST payment type CDATA "check">  XML example:  <payment type="check"> |

Specifying a default value for an attribute, assures that the attribute will get a value even if the author of the XML document didn't include it.

**Implied attribute**

|  |
| --- |
| Syntax:  <!ATTLIST element-name attribute-name attribute-type #IMPLIED>  DTD example:  <!ATTLIST contact fax CDATA #IMPLIED>  XML example:  <contact fax="555-667788"> |

Use an implied attribute if you don't want to force the author to include an attribute and you don't have an option for a default value either.

**Required attribute**

|  |
| --- |
| Syntax:  <!ATTLIST element-name attribute\_name attribute-type #REQUIRED>  DTD example:  <!ATTLIST person number CDATA #REQUIRED>  XML example:  <person number="5677"> |

Use a required attribute if you don't have an option for a default value, but still want to force the attribute to be present.

**Fixed attribute value**

|  |
| --- |
| Syntax:  <!ATTLIST element-name attribute-name attribute-type #FIXED "value">  DTD example:  <!ATTLIST sender company CDATA #FIXED "Microsoft">  XML example:  <sender company="Microsoft"> |

Use a fixed attribute value when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

**Enumerated attribute values**

|  |
| --- |
| Syntax:  <!ATTLIST element-name attribute-name (eval|eval|..) default-value>  DTD example:  <!ATTLIST payment type (check|cash) "cash">  XML example:  <payment type="check">  or  <payment type="cash"> |

Use enumerated attribute values when you want the attribute values to be one of a fixed set of legal values.

## Quelques notions sur les XSD

### Sources

En français, se reporter par exemple à :

<http://jean-luc.massat.perso.luminy.univ-amu.fr/ens/xml/07-xsd.html>

ou bien : <http://blog.paumard.org/cours/xml/chap03-description-xml-schema.html>

### Définition d'une XSD

La norme XML Schema est encore une norme W3C, écrite dans le but de remplacer les DTD. XML Schema possède trois avantages sur DTD :

* c’est une façon plus complète et plus précise de spécifier des documents XML ;
* on peut étendre un schéma par un autre schéma, à l’image des classes ;
* le schéma lui-même est écrit en XML, répondant à un méta-schéma.

Cette norme est actuellement de plus en plus utilisée. Cela dit, pour des raisons de compatibilités ascendantes, les DTD restent encore très présentes.

### Modèle XML d'exemple

Fichier XML marin.xml

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

<!-- un premier marin -->

**<marin** id="12"**>**

**<nom>**Surcouf**</nom>**

**<prenom>**Robert**</prenom>**

**<remarque** lang="FR"**>**Capitaine corsaire**</remarque>**

**</marin>**

### Exemple de schéma

Écrivons un schéma pour spécifier nos marins.

**Exemple. Un premier exemple de XML schéma pour marin.xml**

<?xml version="1.0"?>

**<xsd:schema** xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:galilee="http://www.galilee.org"

targetNamespace="http://www.galilee.org"

**>**

**<xsd:element** name="marin"**>**

**<xsd:complexType>**

**<xsd:sequence>**

**<xsd:element** name="nom" type="xsd:string"**/>**

**<xsd:element** name="prenom" type="xsd:string"**/>**

**</xsd:sequence>**

**</xsd:complexType>**

**</xsd:element>**

**</xsd:schema>**

La première remarque que l’on peut faire, est qu’un schéma XML est lui-même un document XML. Le nœud racine est le nœud schema, appartenant à l’espace de noms http://www.w3.org/2001/XMLSchema, préfixé parxsd. L’attribut targetNamespace indique que les éléments spécifiés dans ce schéma appartiennent à l’espace de noms http://www.galilee.org.

### Types simples

Les éléments d’un XML schema peuvent être de type simple ou complexe. Un élément simple est un élément qui ne possède pas de sous-éléments. Un élément complexe en possède. De plus, un type simple ne peut pas posséder d’attribut. Un élément, même s’il ne possède aucun sous-élément, ou aucun contenu, est de type complexe s’il possède au moins un attribut.

Un élément de type simple se définit simplement par la déclaration suivante.

**Exemple. Déclaration d'un élément de type simple**

**<xsd:element** name="nom" type="xsd:string"**/>**

Un élément simple possède juste un nom et un type. Notons que ce type est une référence à un élément XML, se trouvant dans l’espace de noms http://www.w3.org/2001/XMLSchema, désigné par le préfixe xsd en l’occurrence. Il s’agit effectivement d’un type standard et unique pour tous les schémas XML que l’on peut imaginer, et correspondant à une chaîne de caractères.

Le schéma XML de base déclare ainsi un certain nombre de types standard, dont voici les plus courants.

**Exemple. Principaux types standard XML Schema**

xsd:string

xsd:decimal

xsd:integer

xsd:boolean

xsd:date

xsd:time

De même que pour les DTD, il est possible de fixer une valeur par défaut pour les types simples, ainsi qu’une valeur fixée.

**Exemple. Valeurs par défaut et valeur fixée pour un type simple**

**<xsd:element** name="nom" type="xsd:string" default="Surcouf"**/>**

**<xsd:element** name="prenom" type="xsd:string" fixed="Robert"**/>**

Notons tout de suite qu’il est possible de référencer un type en provenance de n’importe quel schéma lors de la définition d’un type simple, y compris d’un type défini dans le schéma courant. Nous verrons un exemple dans la suite.

### Attribut

Un attribut lui-même est défini à partir d’un type simple dans le cas d’un XML schema. La syntaxe de définition d’un attribut ressemble à celle d’un élément simple.

**Exemple. Définition d’un attribut**

**<xsd:attribute** name="age" type="xsd:int"**/>**

De la même façon que pour les types simples, on peut déclarer une valeur par défaut pour un attribut, ou une valeur fixée, en ajoutant un attribut default ou fixed à l’élément xsd:attribute.

### Restriction sur une déclaration

Un des points intéressants de la déclaration sous forme de XML Schema, est que l’on peut contraindre les valeurs des éléments et attributs que l’on définit.

Montrons comment il est possible de déclarer que l'âge de nos marins doit être un entier positif, qui ne peut pas dépasser 150.

**Exemple. Déclarations de restrictions sur l’âge**

**<xsd:attribute** name="age"**>**

**<xsd:simpleType>**

**<xsd:restriction** base="xsd:integer"**>**

**<xsd:minInclusive** value="0"**/>**

**<xsd:maxInclusive** value="150"**/>**

**</xsd:restriction>**

**</xsd:simpleType>**

**</xsd:attribute>**

Notons que l'on ne peut déclarer des restrictions que sur des éléments de type simpleType. Pour déclarer nos restrictions sur le type age, il faut utiliser le sous-élément restriction de simpleType. Cet élément prend lui-même d'autres sous-éléments dont minInclusive et maxInclusive.

Dans notre cas, les restrictions sont énoncées sur le type standard integer. Tous les types peuvent être utilisés ici, pas seulement les types standard.

Il existe de nombreux types de restrictions disponibles, on se reportera à la spécification pour en avoir la liste exhaustive.

Dans cet exemple, on déclare le type de age par un sous-élément de l’élément attribute. Cette déclaration est légale. Toutefois, on peut aussi déclarer un tel type différemment, en donnant un nom au simpleType, et en y faisant référence dans la déclaration de notre attribut, comme dans l’exemple suivant.

**Exemple. Déclaration nommée de restriction sur l’âge**

**<xsd:simpleType** name="ageType"**>**

**<xsd:restriction** base="xsd:integer"**>**

**<xsd:minInclusive** value="0"**/>**

**<xsd:maxInclusive** value="150"**/>**

**</xsd:restriction>**

**</xsd:simpleType>**

**<xsd:attribute** name="age" type="ageType"**/>**

Dans cet exemple, on crée un type simple nommé ageType, et l’on y fait référence dans la déclaration de notre attribut. Cette deuxième façon de faire présente deux avantages :

* il est possible d’utiliser notre déclaration ageType dans autant de types simples que l’on veut ;
* cela permet de séparer la définition des types d’une part et leur utilisation, éventuellement de l’écrire dans un fichier séparé.

### Types complexes

Un type complexe peut posséder des sous-éléments, et des attributs. Dès l’instant que l’on souhaite créer des éléments possédant des sous-éléments, ou possédant des attributs, alors ils doivent être définis par des types complexes.

Il existe quatre types d’éléments complexes :

* les éléments vides ;
* les éléments ayant un contenu textuel ;
* les éléments possédant des sous-éléments ;
* les éléments ayant à la fois un contenu textuel et des sous-éléments.

Tous ces éléments peuvent en plus avoir ou non des attributs.

### Elément sans contenu

Écrivons un élément sans contenu.

**Exemple d'un élément XML sans contenu**

**<marin** id="2"**/>**

Le schéma pour un tel élément est relativement simple. Il s’agit d’un élément de type complexe.

**Exemple. Schéma XML pour un élément vide**

**<xsd:complexType** name="marinType"**>**

**<xsd:attribute** name="id" type="xsd:integer"**/>**

**</xsd:complexType>**

Cet élément ne comporte pas de contenu ni de sous-élément, il n’y a donc pas lieu de préciser de sous-élément simpleContent ou complexContent.

### Contenu textuel

Voici un exemple d'élément possédant un contenu textuel. Écrivons un schéma pour cet élément.

**Exemple. Élément avec attribut et contenu entier**

**<longueur** unite="cm"**>**10**</longueur>**

Cet élément peut prendre des valeurs entières, et possède un attribut unite. Il s’agit d’un contenu simple (pas de sous-éléments), extension du type entier. On peut donc l’écrire de la façon suivante.

**Exemple. Schéma pour un élément avec attribut et contenu entier**

**<xsd:complexType** name="typeLongueur"**>**

**<xsd:simpleContent>**

**<xsd:extension** base="xsd:integer"**>**

**<xsd:attribute** name="unite" type="xsd :string"**/>**

**</xsd:extension>**

**</xsd:simpleContent>**

**</xsd:complexType>**

### Contenu composé de sous éléments

Les sous-éléments sont spécifiés les uns après les autres, dans l’ordre dans lequel ils doivent apparaître, à la manière des DTD. On peut de plus préciser leur cardinalité.

Reprenons l’exemple de notre élément marin, qui doit obligatoirement posséder un unique nom, un unique prénom, et autant de commentaires que l’on souhaite.

Un commentaire est un élément complexe, possédant un contenu textuel et un attribut qui indique la langue dans laquelle ce commentaire est écrit. Voici donc son schéma.

**Exemple. Schéma pour l'élément commentaire**

**<xsd:complexType** name="commentaireType"**>**

**<xsd:simpleContent>**

**<xsd:extension** base="xsd:string"**>**

**<xsd:attribute** name="lang" type="xsd:string"**/>**

**</xsd:extension>**

**</xsd:simpleContent>**

**</xsd:complexType>**

Notons qu’il existe un type lang présent dans l’espace de nom standard, défini comme pouvant prendre des valeurs normalisées, désignant les langues existantes. On aurait tout à fait pu l’utiliser à la place dexsd:string.

On peut à présent créer notre schéma pour un élément marin, s’appuyant sur le commentaireType.

**Exemple. Schéma pour un élément comportant des sous-éléments**

**<xsd:complexType** name="marinType"**>**

**<xsd:complexContent>**

**<xsd:sequence>**

**<xsd:element** name="nom" type="xsd:string" minOccurs="1" maxOccurs="1"**/>**

**<xsd:element** name="prenom" type="xsd:string" minOccurs="1" maxOccurs="1"**/>**

**<xsd:element** name="commentaire" type="commentaireType" minOccurs="0"**/>**

**</xsd:sequence>**

**</xsd:complexContent>**

**</xsd:complexType>**

Ce contenu complexe contient une séquence. Une séquence est composée d’éléments qui doivent apparaître dans cet ordre dans le document XML final.

On remarquera deux choses sur ce schéma :

* la référence interne à un type défini ailleurs dans le schéma : commentaireType ;
* la spécification de la cardinalité, par l’utilisation des attributs minOccurs et maxOccurs de l’élément element.

Il est également possible de créer des éléments en tant que tel, à l’extérieur de séquences, et d’y faire référence ensuite dans les séquences. De la même façon que la définition de types internes, cela permet de réutiliser la définition d’un élément susceptible d’être utilisé dans plusieurs séquences différentes. Voyons cela sur le même exemple.

**Exemple. Schéma pour un élément comportant des sous-éléments nommés**

**<xsd:element** name="nom" value="xsd:string"**/>**

**<xsd:element** name="prenom" value="xsd:string"**/>**

**<xsd:complexType** name="marinType"**>**

**<xsd:complexContent>**

**<xsd:sequence>**

**<xsd:element** ref="nom" minOccurs="1" maxOccurs="1"**/>**

**<xsd:element** ref="prenom" minOccurs="1" maxOccurs="1"**/>**

**<xsd:element** name="commentaire" type="commentaireType"

minOccurs="0"**/>**

**</xsd:sequence>**

**</xsd:complexContent>**

**</xsd:complexType>**

### Indicateurs

Nous avons utilisé un élément, sequence, qui appartient à une famille appelée les indicateurs . Les indicateurs permettent de préciser la façon dont les sous-éléments s’organisent dans un document XML. Il existe trois types d’indicateurs : les indicateurs d’ordre, les indicateurs d’occurrence et les indicateurs de groupe.

Les indicateurs d’ordre sont les suivants :

* l’indicateur sequence, présentés dans nos exemples, impose que les éléments apparaissent dans l’ordre spécifié par le schema ;
* l’indicateur choice : permet à l’un des éléments d’être présent ;
* l’indicateur all : impose que tous les éléments soient présents une et une seule fois, mais dans n’importe quel ordre.

Les indicateurs d’occurrence sont au nombre de deux, ils viennent en tant qu’attributs de sequence :

* maxOccurs : indique le nombre maximal de sous-éléments de ce type (peut prendre la valeur unbounded) ;
* minOccurs : indique le nombre minimal de sous-éléments de ce type (peut prendre la valeur 0).

Enfin les indicateurs de groupe, au nombre de deux également :

* les groupes d’éléments : group ;
* les groupes d’attributs : attributeGroup.

Les groupes permettent de déclarer des blocs de sous-éléments. Un élément xsd:group doit nécessairement contenir au moins un sous-élément all, choice ou sequence. On peut ensuite insérer ce groupe dans un autre sous-élément.

Les groupes d’attributs fonctionnent de la même façon, sauf qu’ils définissent des blocs d’attributs réutilisables. On les utilise en tant que sous-éléments de complexType, afin de définir les attributs de nos éléments.

**Exemple. Définition et utilisation d’un groupe**

**<xsd:group** name="PersonneGroup"**>**

**<xsd:sequence>**

**<xsd:element** name="nom" type="xsd:string"**/>**

**<xsd:element** name="prenom" type="xsd:string"**/>**

**</xsd:sequence>**

**</xsd:group>**

**<xsd:complexType** name="MarinType"**>**

**<xsd:sequence>**

**<xsd:group** ref="PersonneGroup"**/>**

**<xsd:element** name="naissance" value="xsd:date"**/>**

**</xsd:sequence>**

**</xsd:complexType>**