ETSI ES 201 873-9 V4.6.1 (2015-06)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 9: Using XML schema with TTCN-3

**ETSI Standard**

Reference

RES/MTS-201873-9 T3ed461

Keywords

language, testing, TTCN-3, XML

***ETSI***

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C

Association à but non lucratif enregistrée à la

Sous-Préfecture de Grasse (06) N° 7803/88

***Important notice***

The present document can be downloaded from:  
<http://www.etsi.org/standards-search>

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <http://portal.etsi.org/tb/status/status.asp>

If you find errors in the present document, please send your comment to one of the following services:  
<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

***Copyright Notification***

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.  
The content of the PDF version shall not be modified without the written authorization of ETSI.  
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2015.

All rights reserved.

**DECT**TM, **PLUGTESTS**TM, **UMTS**TM and the ETSI logo are Trade Marks of ETSI registered for the benefit of its Members.  
**3GPP**TM and **LTE**™ are Trade Marks of ETSI registered for the benefit of its Members and  
of the 3GPP Organizational Partners.  
**GSM**® and the GSM logo are Trade Marks registered and owned by the GSM Association.

## 8.2 Type substitution

This clause is invoked if the XSD built-in type, *simpleType* or *complexType* is referenced by the *base* attribute of the *restriction* or *extension* element information item(s) of one or more global XSD type definition(s) (i.e. the type is a parent type of one or more global derived types) AND the parent type occurs as the type of at least one XSD *element* declaration and the user has requested to generate TTCN-3 code allowing using type substitution (see clause 8). These types are called substitutable parent types (as opposed to parent types that cannot be substituted because e.g. referenced only in *attribute* declarations). Please note that when the type of an element is substituted in an instance document, XSD requires that the actual type is identified by an *xsi:type* XML attribute.

NOTE 1: This definition also includes the case when the type of an element is a built-in XSD data type and one or more user-defined types are derived from this built-in type.

In addition to the TTCN-3 types generated according to clause 7 of the present document, for each substitutable parent type a TTCN-3 **union** type shall be generated. The name of the **union** type shall be the result of applying clause 5.2.2 to the name composed of the substitutable parent type's name, or in case the parent type is a built-in XSD type the names defined in clause 6 of the present documant, postfixed by "\_derivations".

One alternative shall be added for the substitutable parent type itself and one for each type derived from it in one or more derivation steps. The first alternative (field) of the **union** type shall correspond to the substitutable parent type. The alternatives corresponding to the derived types shall be added in an ordered manner, first alphabetically ordering the types according to their target namespaces (types with no target namespace first) and subsequently alphabetically ordering the types with the same namespace based on their names. For each alternative, the field name shall be the name applying clause 5.2.2 to the name of the XSD type corresponding to the given alternative. The type of the alternative shall be:

* the TTCN-3 type generated by applying clause 7 to the substitutable parent type for the first field (corresponding to the substitutable parent type);
* the TTCN-3 type resulted by the translation of the derived type for the subsequent fields.

Finally the "useType" encoding instruction shall be attached to the TTCN-3 **union** type.

NOTE 2: Please note that the first alternative of the union is encoded without an xsi:type attribute. The user, if he wants to force xsi:type for the first alternative, needs to add the "useType" encoding instruction to the first field manually.

When translating XSD references to the substitutable parent type to TTCN-3, the TTCN-3 **union** type generated according to this clause shall be used.

EXAMPLE 1: Built-in type substitution

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

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

targetNamespace="http://www.example.org/builtinTypeSubstitution"

xmlns:this="http://www.example.org/builtinTypeSubstitution">

<xsd:element name="elem" type="xsd:integer"/>

<!-- derived type -->

<xsd:simpleType name="integer\_deriv">

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

<xsd:minInclusive value="5"/>

</xsd:restriction>

</xsd:simpleType>

*Will be translated to TTCN-3 e.g. as:*

**module** www\_example\_org\_builtinTypeSubstitution **{**

**import** **from** XSD **all**;

**type** Integer\_derivations Elem

**with {**

**variant** "name as uncapitalized";

**variant** "element";

}**;**

<!-- derived type -->

**type** XSD.Integer Integer\_deriv (5 .. infinity)

**with {**

variant "name as uncapitalized";

**}**;

**type union** Integer\_derivations

**{**

XSD.Integer integer\_,

Integer\_deriv integer\_deriv

**}**

**with {**

**variant** "name as uncapitalized";

**variant** "useType";

**variant** (integer\_) "name as 'integer'";

**}**;

**}**

**with {**

**encode** "XML";

**variant** "namespace as 'http://www.example.org/builtinTypeSubstitution' prefix 'this'";

**variant** "controlNamespace 'http://www.w3.org/2001/XMLSchema-instance' prefix 'xsi'";

**}**

EXAMPLE 2: Simple type substitution

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

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

targetNamespace=*"www.example.org/simpleTypeSubstitution"*

xmlns=*"www.example.org/simpleTypeSubstitution"*>

<xsd:element name=*"request"* type=*"requestType"* />

<!-- The generic type -->

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

<xsd:sequence>

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

</xsd:sequence>

</xsd:complexType>

<!-- Production specific derived type -->

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

<xsd:complexContent>

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

<xsd:sequence>

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

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!-- Programming specific derived type -->

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

<xsd:complexContent>

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

<xsd:sequence>

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

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

</xsd:schema>

*Will be translated to TTCN-3 e.g. as:*

**module** www\_example\_org\_simpleTypeSubstitution **{**

**import** **from** XSD **all**;

**type** RequestType\_derivations Request

**with** **{**

**variant** "name as uncapitalized";

**variant** "element";

**}**;

/\* The generic type \*/

**type** **record** RequestType

**{**

XSD.String commonName

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

/\* Production specific derived type \*/

**type** **record** MyProductionRequestType

**{**

XSD.String commonName,

XSD.String productionName

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

/\* Programming specific derived type \*/

**type** **record** MyProgrammingRequestType

**{**

XSD.String commonName,

XSD.String programmingName

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

**type** **union** RequestType\_derivations

**{**

RequestType requestType,

MyProductionRequestType myProductionRequestType,

MyProgrammingRequestType myProgrammingRequestType

**}**

**with** **{**

**variant** "name as uncapitalized";

**variant** "useType";

**}**;

**}** **with** **{**

**encode** "XML";

**variant** "namespace as 'www.example.org/simpleTypeSubstitution'";

**variant** "controlNamespace 'http://www.w3.org/2001/XMLSchema-instance' prefix 'xsi'";

**}**

EXAMPLE 3: Type substitution with cascaded derivations

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

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

targetNamespace=*"www.example.org/typeSubstCascaded3"*

xmlns=*"www.example.org/typeSubstCascaded3"*>

<xsd:element name=*"request"* type=*"requestType"* />

<!-- The generic base type -->

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

<xsd:sequence>

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

</xsd:sequence>

</xsd:complexType>

<!-- Production implementation -->

<xsd:element name=*"product"* type=*"myProductionRequestType"* />

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

<xsd:complexContent>

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

<xsd:sequence>

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

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!-- Derived type of myProductionRequestType -->

<xsd:complexType name=*"myProductionRequestType2"*>

<xsd:complexContent>

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

<xsd:sequence>

<xsd:element name=*"productItem"* type=*"xsd:integer"* minOccurs=*"0"* />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!-- Derived type of myProductionRequestType2 -->

<xsd:complexType name=*"myProductionRequestType3"*>

<xsd:complexContent>

<xsd:restriction base=*"myProductionRequestType2"*>

<xsd:sequence>

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

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

<xsd:element name=*"productItem"* type=*"xsd:integer"* minOccurs=*"1"* />

</xsd:sequence>

</xsd:restriction>

</xsd:complexContent>

</xsd:complexType>

</xsd:schema>

*Will be translated to TTCN-3 e.g. as:*

NOTE 3: Please note that though the XSD type myProductionRequestType2 has a type derived from it, no MyProductionRequestType2\_derivations **union** type is generated, because it is not used as the type of any XSD element.

**module** www\_example\_org\_typeSubstCascaded3 **{**

**import** **from** XSD **all**;

**type** RequestType\_derivations Request

**with** **{**

**variant** "name as uncapitalized";

**variant** "element";

**}**;

/\* The generic abstract type \*/

**type** **record** RequestType

**{**

XSD.String commonName

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

/\* Production implementation \*/

**type** MyProductionRequestType\_derivations Product

**with** **{**

**variant** "name as uncapitalized";

**variant** "element";

**}**;

**type** **record** MyProductionRequestType

**{**

XSD.String commonName,

XSD.String productionName

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

/\* Derived type of myProductionRequestType \*/

**type** **record** MyProductionRequestType2

**{**

XSD.String commonName,

XSD.String productionName,

XSD.Integer productItem **optional**

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

/\* Derived type of myProductionRequestType2 \*/

**type** **record** MyProductionRequestType3

**{**

XSD.String commonName,

XSD.String productionName,

XSD.Integer productItem

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

**type** **union** RequestType\_derivations

**{**

RequestType requestType,

MyProductionRequestType myProductionRequestType,

MyProductionRequestType2 myProductionRequestType2,

MyProductionRequestType3 myProductionRequestType2,

**}**

**with** **{**

**variant** "name as uncapitalized";

**variant** "useType";

**}**;

**type** **union** MyProductionRequestType\_derivations

**{**

MyProductionRequestType myProductionRequestType,

MyProductionRequestType2 myProductionRequestType2,

MyProductionRequestType3 myProductionRequestType2,

**}**

**with** **{**

**variant** "name as uncapitalized";

**variant** "useType";

**}**;

**}**

**with** **{**

**encode** "XML";

**variant** "namespace as 'www.example.org/typeSubstCascaded3'";

**variant** "controlNamespace 'http://www.w3.org/2001/XMLSchema-instance' prefix 'xsi'";

**}**

If the value of the substitutable parent type's *abstract* attribute is "*true*", the "abstract" encoding instruction has to be attached to the field corresponding to the substitutable parent type, i.e. to the first field.

NOTE 4: If the value of a derived type's *abstract* attribute is "true", the "abstract" encoding instruction is attached to the TTCN-3 type generated for that XSD type, according to clause 7.1.9.

If the substitutable parent type's effective block value (see clause 7.1.10) is "*#all*", the "block" encoding instruction shall be attached to all fields of the **union** type except the field corresponding to the substitutable parent type (the first field).

If the substitutable parent type's effective block value (see clause 7.1.10) is "*restriction*" or "*extension*" the "block" encoding instruction shall be attached to all fields, generated for types, derived from the substitutable parent type by *restriction* or by *extension*, respectively, at any step along the derivation path.

NOTE 5: The TTCN-3 syntax allows to attach the same attribute to several fields of the same structured type in one with attribute.

EXAMPLE 4: Mapping a substitutable abstract type:

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

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

targetNamespace=*"www.example.org/typeSubstitutionAbstract"*

xmlns=*"www.example.org/typeSubstitutionAbstract"*>

<xsd:element name=*"request"* type=*"requestAbstractType"* />

<!-- The generic abstract type -->

<xsd:complexType name=*"requestAbstractType"* abstract=*"true"*>

<xsd:sequence>

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

</xsd:sequence>

</xsd:complexType>

<!-- Production implementation -->

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

<xsd:complexContent>

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

<xsd:sequence>

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

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!-- Programming implementation -->

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

<xsd:complexContent>

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

<xsd:sequence>

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

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

</xsd:schema>

Will be translated to TTCN-3 e.g. as:

**module** www\_example\_org\_typeSubstitutionAbstract **{**

**import** **from** XSD **all**;

**type** RequestAbstractType\_derivations Request

**with** **{**

**variant** "name as uncapitalized";

**variant** "element";

**}**;

/\* The generic abstract type \*/

**type** **record** RequestAbstractType

**{**

XSD.String commonName

**}**

**with** **{**

**variant** "name as uncapitalized";

**variant** "abstract";

**}**;

/\* Production implementation \*/

**type** **record** MyProductionRequestType

**{**

XSD.String commonName,

XSD.String productionName

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

/\* Programming implementation \*/

**type** **record** MyProgrammingRequestType

**{**

XSD.String commonName,

XSD.String programmingName

**}**

**with** **{**

**variant** "name as uncapitalized";

**}**;

**type** **union** RequestAbstractType\_derivations

**{**

RequestAbstractType requestAbstractType,

MyProductionRequestType myProductionRequestType,

MyProgrammingRequestType myProgrammingRequestType

**}**

**with** **{**

**variant** "name as uncapitalized";

**variant** "useType";

**variant** (requestAbstractType) "abstract";

**}**;

**}** **with** **{**

**encode** "XML";

**variant** "namespace as 'www.example.org/typeSubstitutionAbstract'";

**variant** "controlNamespace 'http://www.w3.org/2001/XMLSchema-instance' prefix 'xsi'";

**}**