software AG

Tamino

Tamino XML Schema Reference Guide

Version 10.1

April 2018

WEBMETHODS

This document applies to Tamino Version 10.1 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: INS-TSL-REF-101-20180413

Table of Contents

Preface

The purpose of this document is to describe the elements of the *Tamino Schema Definition* language as implemented in the current version of Tamino. A user guide is also available: *Tamino XML Schema Language (TSD) User Guide*.

This document is intended for database administrators familiar with methods of data modeling and data management tasks (for example, indexing, normalization). Their task will be to create schemas and define them to the Tamino *Data Map*.

Application programmers intending to write programs that address Tamino schemas should also be familiar with the principles of schema definition.

This document contains reference documentation that describes the *TSD* elements that are similar to *XML Schema*'s elements and also TSD's extensions to the XML Schema 1.0 standard of the ***W3C***. An overview is given at:

TSD Logical: Definitions (xs namespace)

Tamino Extensions to XML Schema: List of Elements and Attributes (tsd namespace, containing mainly TSD Physical)

Features of the W3C XML Schema that are Not Supported by Tamino

I TSD Logical: Definitions

1 TSD Logical: Definitions

This part of the document describes the logical part of TSD that represents a subset of the ***XML Schema*** functionality defined by the ***W3C*** containing all functionality needed to define schemas for an XML database.

The following elements are provided by TSD and described in this document:

`xs:all`
`xs:annotation`
`xs:any`
`xs:anyAttribute`
`xs:appinfo`
`xs:attribute`
`xs:attributeGroup`
`xs:choice`
`xs:complexContent`
`xs:complexType`
`xs:documentation`
`xs:element`
`xs:enumeration`
`xs:extension`
`xs:field`
`xs:fractionDigits`
`xs:group`
`xs:import`
`xs:include`
`xs:key`
`xs:keyref`

xs:length
xs:list
xs:maxExclusive
xs:maxInclusive
xs:maxLength
xs:minExclusive
xs:minInclusive
xs:minLength
xs:notation
xs:pattern
xs:restriction
xs:schema
xs:selector
xs:sequence
xs:simpleContent
xs:simpleType
xs:totalDigits
xs:union
xs:unique
xs:whiteSpace

2 xs:all

Purpose	This element is used when defining elements using the `xs:all` particle in a complex type definition, meaning that the child elements can appear in arbitrary order, and that each child element can occur at most once (if the value of the `minOccurs` attribute is 0) or must occur exactly once (if the value of the `minOccurs` attribute is 1).
Parent element	`xs:complexType`, `xs:extension`, `xs:group`
Child elements	`xs:element`
Attributes	`minOccurs`, `maxOccurs`

Name	Type	Description
`minOccurs`	`xs:nonNegativeInteger`	Together with the `maxOccurs` attribute described below, this attribute expresses multiplicity in TSD (and ***XML Schema***) in `xs:all` elements within a content model. The value of a `minOccurs` attribute declaration determines the minimum number of occurrences of `xs:all`. The value of `minOccurs` must be 0 or 1. **Note:** A similar `minOccurs` parameter is also available for the `xs:element` element.
`maxOccurs`	`xs:nonNegativeInteger`	Together with the `minOccurs` attribute described above, this attribute expresses multiplicity in TSD (and **XML Schema**) in `xs:all` elements within a content model. The value of a `maxOccurs` attribute determines the maximum number of occurrences of an `xs:all` element. The value of `maxOccurs` may only be 1.

Attributes

Example 1

The following example indicates that the `cl_firstname` and the `cl_lastname` elements can appear in either order but each element must occur exactly once.

```
<xs:element name="client">
  <xs:complexType>
    <xs:all>
      <xs:element name="cl_firstname" type="xs:string"/>
      <xs:element name="cl_lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Example 2

The following example indicates that the `customer_firstname` and the `customer_lastname` elements can appear in either order and each child element is optional.

```
<xs:element name="customer">
  <xs:complexType>
    <xs:all minOccurs="0">
      <xs:element name="customer_firstname" type="xs:string"/>
      <xs:element name="customer_lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

3 xs:annotation

Purpose	Allows additional information to be associated with an object, e.g. comments (`xs:documentation`) or application-related data (`xs:appinfo`).
Parent element	Essentially any element defined by the ***W3C XML Schema*** except `xs:annotation`, `xs:appinfo` and `xs:documentation`. See also `tsd:schemaInfo`, `tsd:elementInfo` and `tsd:attributeInfo`.
Child elements	`xs:appinfo`, `xs:documentation`
Attributes	None

Example

The following example shows an `xs:annotation` element containing a `tsd:schemaInfo` element

```
<xs:schema ...>
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name = "NEW_schema">
        <tsd:doctype name = "Customer">
          <tsd:logical>
            <tsd:content>open</tsd:content>
          </tsd:logical>
        </tsd:doctype>
        <tsd:doctype name = "Order">
          <tsd:logical>
            <tsd:content>open</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  .
  .
```

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

4 xs:any

Purpose	This element allows a wildcard to be defined in TSD. It allows the occurrence of arbitrary elements (for instance the inclusion of any piece of well-formed XML code) together with the current element in the XML instance to be validated against the schema. The `processContents` attribute specifies how this fragment of XML code is validated. The origin of the XML code can be specified using the `namespace` attribute.
Parent element	`xs:choice`, `xs:sequence`
Child elements	None
Attributes	`minOccurs`, `maxOccurs`, `processContents`, `namespace`

Name	Type	Description
`minOccurs`	`xs:nonNegativeInteger`	Together with the `maxOccurs` attribute described below, this attribute expresses multiplicity in TSD (and ***XML Schema***) in `xs:any` elements within element definitions. The value of a `minOccurs` attribute in an `xs:any` element determines the minimum number of occurrences of `any` elements. The value of `minOccurs` must be a non-negative integer. The default value for `minOccurs` is 1.
`maxOccurs`	`xs:nonNegativeInteger` or “unbounded”	Together with the `minOccurs` attribute described above, this attribute expresses multiplicity in TSD (and **XML Schema**) in an `xs:any` element within a content model. The value of a `maxOccurs` attribute in an `xs:any` declaration determines the maximum number of occurrences of `any` elements. The value of `maxOccurs` must either be a non-negative integer or the value "unbounded". The default value for `maxOccurs` is 1.
`namespace`	■ `xs:anyURI` ■ `##any`	The `namespace` attribute can only have the following values:

Name	Type	Description
	■ ##other ■ ##local ■ ##targetNamespace	■ ##any Elements belonging to any namespace match xs:any. This is the default value. ■ ##other Elements belonging to any non-absent namespace except the current schema's targetNamespace match. ■ **Item list** A whitespace-separated list of items, each matching one of the following: ■ **An arbitrary URI** The item must be a valid URI with respect to the predefined type xs:anyURI. Only elements belonging to the respective namespace are accepted during validation. ■ ##local Only elements without a namespace are allowed. ■ ##targetNamespace Only elements belonging to the current schema's targetNamespace match. Only elements that are valid with respect to at least one of the items are accepted.
processContents	xs:NMTOKEN	This attribute governs whether a strict or a simplified validation process is performed. It can have the values: ■ "strict" ■ "lax" ■ "skip" The default value is "strict". These validation methods differ in their behavior if they encounter elements without a corresponding definition of a global element. If "lax" is specified, the XML processor only validates the elements for which it can obtain schema information, but for elements where no corresponding definition of a global element is available, no errors are signalled. If "strict" is specified, each element must validate against a corresponding global element definition.

Name	Type	Description
		If "skip" is used, the respective elements is not validated at all.

Attributes

Example 1

This example shows a complex type definition with a choice element containing an `xs:any` element with `maxOccurs` and `processContents` attributes for processing with strict validation:

```
<xs:complexType mixed = "true">
  <xs:choice>
    <xs:any processContents = "strict" maxOccurs = "unbounded"></xs:any>
  </xs:choice>
</xs:complexType>
```

Example 2

To allow an arbitrary number of XML elements being defined as global elements, proceed as shown here:

```
<xs:element name="Comment">
  <xs:complexType>
    <xs:sequence>
      .
      .
      .
      <xs:any namespace="##local" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

5 xs:anyAttribute

Purpose	This element can be used to specify attribute wildcards in complex type definitions. It allows the occurrence of arbitrary attributes with the current element in the XML instance to be validated against the schema. The details of the validation of this fragment of XML code are specified with the `processContents` attribute. The origin of the XML code can be specified using the `namespace` attribute.
Parent element	`xs:complexType`, `xs:extension`
Child elements	None
Attributes	`namespace`, `processContents`

Name	Type	Description
`namespace`	`xs:anyURI` or one of the special values `##any`, `##other`, `##local`, `##targetNamespace`	The `namespace` attribute can only have the following values: ■ `##any` Attributes belonging to any namespace match `xs:anyAttribute`. This is the default value. ■ `##other` Attributes belonging to any non-absent namespace other than the current schema's `targetNamespace` match `xs:anyAttribute`. ■ **Item list** A whitespace-separated list of items, each of which matches one of the following: ■ **An arbitrary URI** The item must be a valid URI with respect to the predefined type `xs:anyURI`. Only attributes belonging to the corresponding namespace are accepted during validation.

Name	Type	Description
		■ ##local Only attributes without a namespace are allowed. ■ ##targetNamespace Only attributes belonging to the current schema's targetNamespace match. Only attributes that are valid with respect to at least one of the items are accepted.
processContents	xs:NMTOKEN	This attribute specifies whether strict or simplified validation is performed. It can have the values: ■ "strict" ■ "lax" ■ "skip" The default value is "strict". These validation methods differ in their behavior if they encounter attributes without a corresponding definition of a global attribute. If "lax" is specified, the XML processor only validates the attributes for which it can obtain schema information, but for attributes where no corresponding definition of a global attribute is available, no errors are signalled. If "strict" is specified, each attribute must validate against a corresponding global attribute definition. If "skip" is chosen, the respective attributes are not validated at all.

Attributes

6 xs:appinfo

Purpose	In the *W3C XML Schema*, this element specifies information for applications, i.e. for associating application-related data with a logical schema. Tamino additionally implements a special use: In TSD, this element is required for linking elements from the *tsd* namespace. Tamino extensions to the **W3C XML Schema** standard are added here.
Parent element	`xs:annotation`
Child elements	`tsd:schemaInfo`, `tsd:elementInfo`, `tsd:attributeInfo`
Attributes	`source`

Name	Type	Description
`source`	`xs:anyURI`	An optional link to additional application information.

Attributes

Examples

See `xs:annotation`, `tsd:schemaInfo`, `tsd:elementInfo` or `tsd:attributeInfo`.

7 xs:attribute

Purpose	This element defines an attribute. The `name` attribute and the `ref` attribute are mutually exclusive. If the `xs:attribute` element is the child element of an `xs:schema` element, it defines a global attribute and the `name` attribute is required. If the `xs:attribute` element is the descendant element of an `xs:complexType` element, its `name` attribute defines local attributes.
Parent element	`xs:attributeGroup`, `xs:complexType`, `xs:extension` or `xs:schema`
Child elements	`xs:annotation`, `xs:simpleType`
Attributes	`default`, `fixed`, `form`, `name`, `ref`, `type`, `use`

Name	Type	Description
`default`	`xs:string`	This attribute enables you to specify a default value for an attribute definition. The default value is used if the attribute is not explicitly specified in the XML instance. **Note:** 1. The `default` attribute may not be used together with the `fixed` attribute. 2. Specifying a default value for an attribute only makes sense if the attribute itself is optional.
`fixed`	`xs:string`	This attribute specifies a fixed value for the attribute. That value cannot be changed later on (i.e. it is a constant value). This has the following consequences for instances of that schema:

Name	Type	Description
		■ **Attribute not present in instance:** Tamino inserts the attribute with the value specified by the `fixed` attribute. ■ **Attribute present in instance:** The value of the attribute must match the value specified by the `fixed` attribute. **Note:** The `fixed` attribute may not be used together with the `default` attribute.
`form`	`xs:NMTOKEN`	This attribute specifies whether the appearance of locally-declared attributes in an instance document must be qualified with a namespace prefix. It overrides the default that is set using the `attributeFormDefault` attribute of the `xs:schema` element. It can have either the value "qualified" or "unqualified" (default value).
`name`	`xs:NCName`	This attribute specifies a name for the attribute to be defined using the `xs:attribute` element.
`ref`	`xs:QName`	A global attribute can be referenced in other declarations in the same schema using the `ref` attribute of the `xs:attribute` element. The value of the `ref` attribute must refer to a global attribute. If you specify a `ref` attribute, you cannot specify a `name` attribute on that attribute declaration.
`type`	`xs:QName`	This attribute indicates the type of the attribute to be defined using the `xs:attribute` element. It may not be specified if there is a `xs:simpleType` present, as the occurrence of the `type` attribute is mutually exclusive with the occurrence of `xs:simpleType`. The type can be any of the predefined types described in **section 3 "Built-in datatypes" of *W3C Recommendation, XML Schema Part 2: Datatypes*** or any user-defined simple type.

`use`	`xs:NMTOKEN`	The `use` attribute determines the multiplicity of attributes. It may have one of the following values:
		`required`
		The attribute must appear once.
		`optional`
		The attribute may appear once.
		`prohibited`
		The attribute must not appear at all.
		The default value for the `use` attribute is "optional".
		Restriction:
		If the `default` attribute has been specified in an attribute definition, the `use` attribute may only be specified with the value "optional".

Attributes

Example

This example shows an attribute definition using restriction. It defines a simple type that can only accept the values:

instrumentalist
jazzSinger
jazzComposer

```
<xs:attribute name = "type">
  <xs:simpleType>
    <xs:restriction base = "xs:NMTOKEN">
      <xs:enumeration value = "instrumentalist"/>
      <xs:enumeration value = "jazzSinger"/>
      <xs:enumeration value = "jazzComposer"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

The next example presents a more complex attribute definition in TSD .

```
<xs:attribute name = "C_ob_id" type = "xs:integer">
  <xs:annotation>
    <xs:appinfo>
      <tsd:attributeInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:objectRef>
              <tsd:collectionRef>Customers</tsd:collectionRef>
              <tsd:accessPredicate operator = "=">
                <tsd:nodeRef>/Customer/CustomerNo</tsd:nodeRef>
              </tsd:accessPredicate>
            </tsd:objectRef>
          </tsd:native>
        </tsd:physical>
      </tsd:attributeInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

8 xs:attributeGroup

This element can be used in different contexts:

1. As an `attributeGroup` definition within `xs:schema`:

Purpose	In this context, this element defines named groups of attributes. These attributes can subsequently be referenced in a complex type definition by a common name. See section *3.6 Attribute Group Definitions* of *XML Schema - Part 1*.
Parent element	`xs:complexType`, `xs:extension`, `xs:schema`
Child elements	`xs:attribute`
Attributes	`name`

Name	Type	Description
`name`	`xs:NCName`	The value of the `name` attribute specifies the name that identifies the group of attributes.

Attributes

2. As an `attributeGroup` reference within `xs:complexType` or `xs:attributeGroup` or `xs:extension`:

Purpose	This element references a predefined group of attributes. This leads to the insertion of the whole attribute group at the current location in the schema.
Parent element	`xs:complexType`
Child elements	`xs:annotation` (optional)
Attributes	`ref`

Name	Type	Description
ref	xs:QName	A global attribute can be referenced in other declarations in the same schema using the ref attribute of the xs:attribute element. The value of the ref attribute must refer to a global element. If you specify a ref attribute, you may not specify a name attribute on that element declaration. **Note:** An attributeGroup reference can be nested within an **attributeGroup definition**. However, cyclic references are prohibited.

Attributes

9 xs:choice

Purpose	This element (the “choice” particle) expresses a choice in complex type definitions of elements. It allows one of the particles contained in its scope to be present in instances of the containing element. The number of times that one particle may be chosen from the list is bounded by the values of the `minOccurs` and `maxOccurs` attributes.
Parent element	`xs:choice`, `xs:complexType`, `xs:extension`, `xs:group`, `xs:sequence`
Child elements	`xs:any`, `xs:choice`, `xs:element`, `xs:group`, `xs:sequence`
Attributes	`minOccurs`, `maxOccurs`

Name	Type	Description
`minOccurs`	`xs:nonNegativeInteger`	Together with the `maxOccurs` attribute described below, this attribute expresses multiplicity in TSD (and ***XML Schema***) in choice elements within element definitions. The value of the `minOccurs` attribute in an `xs:choice` declaration determines the minimum number of occurrences of `xs:choice`. The value of the `minOccurs` attribute must be a non-negative integer. The default value of `minOccurs` is 1.
`maxOccurs`	`xs:nonNegativeInteger` or "unbounded"	Together with the `minOccurs` attribute described above, this attribute expresses multiplicity in TSD (and **XML Schema**) in choice elements within element definitions. The value of a `maxOccurs` attribute in an `xs:choice` declaration determines the maximum number of occurrences of `xs:choice`. The value of the `maxOccurs` attribute must be a non-negative integer or "unbounded". The default value of `maxOccurs` is 1.

Attributes

Example 1

The example below defines an element `partner` which must contain either a `customer` element or a `supplier` element or an `employee` element.

```
<xs:element name="partner">
  <xs:complexType>
    <xs:choice>
      <xs:element name="customer" type="t_customer"/>
      <xs:element name="supplier" type="t_supplier"/>
      <xs:element name="employee" type="t_employee"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Example 2

The example below defines an element named `partner` which may contain either a `customer` element or a `supplier` element or an `employee` element.

```
<xs:element name="partner">
  <xs:complexType>
    <xs:choice minOccurs="0">
      <xs:element name="customer" type="t_customer"/>
      <xs:element name="supplier" type="t_supplier"/>
      <xs:element name="employee" type="t_employee"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

10 xs:complexContent

Purpose	This element defines a complex content model.
Parent element	`xs:complexType`
Child elements	`xs:annotation`, `xs:extension`, `xs:restriction`
Attributes	`id`, `mixed`

Name	Type	Description
`id`	`xs:ID`	Assigns an identifier to the schema node.
`mixed`	`xs:boolean`	Specifies whether the complex type being defined with the `xs:complexContent` element allows mixed content (i.e. a combination of text and child elements). The default value for the `mixed` attribute is "false", indicating that `mixed` content is not allowed.

Attributes

11 xs:complexType

Purpose	This element enables you to define a content model for a complex data type. You can use three compositors for different ways of combining data: ■ xs:choice, ■ xs:sequence ■ xs:all Furthermore, it offers the possibility to define simple content models (xs:simpleContent), attributes (xs:attribute) and wildcards for attributes (xs:anyAttribute).
Parent element	xs:element or xs:schema
Child elements	xs:all, xs:anyAttribute, xs:attribute, xs:attributeGroup, xs:choice, xs:complexContent, xs:group, xs:sequence, xs:simpleContent
Attributes	abstract, block, final, mixed, name

Name	Type	Description
abstract	xs:boolean	If "true", the complex type definition must not be used for validation, i.e. an instance must reference, for example, a derived type via xsi:type. Default: "false".
block		This attribute, in conjunction with the **xs:schema** element's blockDefault attribute (if the block attribute is not specified, the blockDefault attribute value takes effect), is used at validation time. It specifies the types that may be used in an xsi:type attribute of an element that is validated against an element declaration that references this complex type definition. The value is a whitespace-separated list of: **extension** The type referenced by xsi:type must not be derived from the current type by extension. **restriction** The type referenced by xsi:type must not be derived from the current type by restriction.

Name	Type	Description
		Alternatively, the special value "#all" indicates that no xsi:type may be used.
final		This attribute, in conjunction with the **xs:schema** element's finalDefault attribute (if the final attribute is not specified, the finalDefault attribute value takes effect), is used when performing schema checks. It specifies the methods that may be used to derive new types from the current complex type. The value is a whitespace-separated list of: **extension** No type may be derived from this complex type by extension. **restriction** No type may be derived from this complex type by restriction. Alternatively, the special value "#all" prevents both methods of deriving a new type from the current complex type definition.
mixed	xs:boolean	Specifies whether the complex type being defined with the xs:complexType element allows mixed content (i.e. a combination of text and child elements). The default value for the mixed attribute is false, indicating that mixed content is not allowed.
name	xs:NCName	For purposes of reuse, it is possible to specify a name attribute for a complex type definition if xs:complexType is used in the context of xs:schema, and to reference it later by this name. This is done with the name attribute of the xs:complexType element. This attribute is mandatory for the definition of named complex types. For anonymous complex types it must be omitted. **Note:** Versions of Tamino up to Tamino 3.1 only supported anonymous complex type definitions.

Attributes

Examples

This example of a complex type definition is taken from the “Jazz” schema example as introduced in the *Advanced Concepts* documentation. It uses xs:all to define a structure containing the two elements location and time:

```
<xs:complexType>
  <xs:all>
    <xs:element name = "location"
                type = "xs:normalizedString"/>
    <xs:element name = "time" type = "xs:dateTime"/>
  </xs:all>
</xs:complexType>
```

The next example is based on the previous one and adds a sequence to it:

```
<xs:complexType>
  <xs:choice>
    <xs:sequence>
      <xs:element name = "from" type = "xs:date"/>
      <xs:element name = "to" type = "xs:date"/>
    </xs:sequence>
    <xs:all>
      <xs:element name = "location"
                  type = "xs:normalizedString"/>
      <xs:element name = "time" type = "xs:dateTime"/>
    </xs:all>
  </xs:choice>
</xs:complexType>
```

The next example shows the definition of an element named `collaborationContext` using a choice that includes `xs:sequence` and `xs:all`.

```
<xs:element name = "collaborationContext">
  <xs:complexType>
    <xs:choice>
      <xs:sequence>
        <xs:element name = "from" type = "xs:date"/>
        <xs:element name = "to" type = "xs:date"/>
      </xs:sequence>
      <xs:all>
        <xs:element name = "location"
                    type = "xs:normalizedString"/>
        <xs:element name = "time" type = "xs:dateTime"/>
      </xs:all>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

12 xs:documentation

Purpose	This element allows a descriptive text to be specified for the current context, either inline or by referencing an external description via the source attribute.
Parent element	`xs:annotation`
Child elements	None
Attributes	`source`

Name	Type	Description
`source`	`xs:anyURI`	An optional link to documentation.

13 xs:element

Purpose	This element allows you to define elements. If the `xs:element` element is the child element of an `xs:schema` element, it defines a global element and the `name` attribute is required. If the `xs:element` element is the descendant element of a `xs:complexType` element, its `name` attribute performs a definition of local elements.
Parent element	`xs:all`, `xs:choice`, `xs:group`, `xs:sequence`, `xs:schema`
Child elements	`xs:annotation`, `xs:complexType`, `xs:key`, `xs:keyref`, `xs:simpleType`, `xs:unique`
Attributes	`abstract` (global elements only), `default`, `final` (global elements only), `fixed`, `form` (only if not below `xs:schema`), `maxOccurs` (only if not below `xs:schema`), `minOccurs` (only if not below `xs:schema`), `name`, `nillable`, `ref` (only if not below `xs:schema`), `substitutionGroup` (global elements only), `type`

Name	Type	Description
`abstract`	`xs:boolean`	If "true", this element declaration must not be used for validation. Typically, another element from the substitution group will be used in an XML document.
`block`	`xs:boolean`	This attribute, in conjunction with the **schema**'s `blockDefault` attribute (if `block` is not specified, the `blockDefault` attribute value takes effect), is used at validation time. It specifies the types that can be used in an xsi:type attribute or the types of a substituting element that can be used in an element that is validated against the current element declaration. The value is a whitespace-separated list of: **restriction** The alternate type definition is not allowed to be derived by restriction from the default type defined by this element declaration.

Name	Type	Description
		extension The alternate type definition is not allowed to be derived by extension from the default type defined by this element declaration. **substitution** Substitution by a member of the substitution group is not allowed. Alternatively, the value "#all" can be specified. This prevents the substitution of an element that is validated against this element declaration.
`default`	`xs:string`	Allows you to specify for an element a default value that is inserted if the element occurs with empty content in the XML instance. **Note:** The `default` attribute may not be used together with the `fixed` attribute.
`final`		This attribute, in conjunction with the **schema**'s `finalDefault` attribute (if `final` is not specified, the `finalDefault` attribute value takes effect), is used when performing schema checks. It specifies the substitutions that are valid for an element that is validated against the current element declaration. The value is a whitespace-separated list of: **substitution** No substitution is allowed. **restriction** Substitution by an element whose datatype is derived from the current element's datatype by restriction is not allowed. **extension** Substitution by an element whose datatype is derived from the current element's datatype by extension is not allowed. Alternatively, the special value "#all" can be specified. This also means that no substitution is allowed.
`fixed`	`xs:string`	Specifies the value that the element that must have if present. This has the following consequences for instances of that schema:

Name	Type	Description
		■ **Element present in instance:** The value of the element must either match the value of the `fixed` attribute or be empty. If empty, Tamino inserts the value of the `fixed` attribute into the element. ■ **Element not present in instance:** Tamino does not insert the element or any value into the XML instance. **Note:** The `fixed` attribute may not be used together with the `default` attribute.
`form`	`xs:NMTOKEN`	Determines whether the occurrence of locally-declared elements in an instance document must be qualified by a namespace specification. It overrides the default that is set by the `elementFormDefault` attribute of the `xs:schema` element. It can have the value "qualified" or "unqualified". The default is "unqualified".
`maxOccurs`	`xs:nonNegativeInteger` or "unbounded"	Together with the `minOccurs` attribute described above, this attribute expresses multiplicity in TSD (and **XML Schema**) element definitions. The value of a `maxOccurs` attribute in an `xs:element` declaration determines the maximum number of occurrences of a specific element. The value of `maxOccurs` must be a non-negative integer or the value "unbounded". The default value for `maxOccurs` is 1.
`minOccurs`	`xs:nonNegativeInteger`	Together with the `maxOccurs` attribute described below, this attribute expresses multiplicity in TSD (and **XML Schema**) element definitions. The value of a `minOccurs` attribute in an `xs:element` declaration determines the minimum number of occurrences of a specific element. The value of `minOccurs` must be a non-negative integer. The default value for `minOccurs` is 1.
`name`	`xs:NCName`	Specifies a name for the element. **Note:** The `name` and `ref` attributes are mutually exclusive.
`nillable`	`xs:boolean`	If "true", an instance to be validated against this element may have an additional `xsi:nil` attribute. If `xsi:nil="true"`, the element must not have any content: neither text nor child nodes. The validation of attributes is not affected. (See **section 2.9 Nil Values** in the W3C documentation **XML Schema Part 0: Primer**.) Default value: "false". **Tamino-Specific Constraints**

Name	Type	Description
		An element or a descendant of an element with `nillable="true"` must not be referenced in a tsd:unique constraint.
`ref`	`xs:QName`	A global element can be referenced in other declarations in the same schema using the `ref` attribute of the `xs:element` element. The value of the `ref` attribute must refer to a global element. Only the `minOccurs` and `maxOccurs` attributes are allowed in conjunction with the `ref` attribute For more information, refer to the **subsection 2.2 *Complex Type Definitions, Element & Attribute Declarations* in W3C's document *XML Schema Part 0: Primer*** **Note:** The `name` and `ref` attributes are mutually exclusive.
`substitutionGroup`	`xs:QName`	Refers to a global element declaration with a matching QName. The referenced element declaration is called the substitution group head element. This is an element that is valid against the current element declaration (which must be a global declaration), then it is a valid replacement for any element validated against the substitution group head element. See **section 4.6 Substitution Groups** in the W3C documentation **XML Schema Part 0: Primer**.
`type`	`xs:QName`	Indicates the type of the element to be defined. The type can be any of the predefined types described in **section 3 *Built-in datatypes* of *W3C Recommendation, XML Schema Part 2: Datatypes*** or any user-defined type. The occurrence of the `type` attribute and the occurrence of the `xs:simpleType` and `xs:complexType` child elements are mutually exclusive. The type can be either a simple or a complex type. Simple types are defined with the `xs:simpleType` element; complex types are defined with the `xs:complexType` element.

Attributes

Example (using a complex type definition)

This example of an element definition using a complex type definition is taken from an **XML Schema** for a book store. It describes how an attribute `Category` is defined that can only have the values:

autobiography
non-fiction
fiction

```
      <xs:element name="Book">
        <xs:complexType>
          <xs:attribute name="Category" use="required">
            <xs:restriction>
              <xs:simpleType base="xs:string">
                <xs:enumeration value="autobiography"/>
                <xs:enumeration value="non-fiction"/>
                <xs:enumeration value="fiction"/>
              </xs:simpleType>
            </xs:restriction>
          </xs:attribute>
          <xs:attribute name="InStock" type="xs:boolean" use="optional" ↩
default="false"/>
           <xs:attribute name="Reviewer" type="xs:string" use="optional" default=""/>
        </xs:complexType>
      </xs:element>
```

14 xs:enumeration

Purpose	This element enables you to define an enumeration type by restricting an existing type (the base type). The restriction is done by constraining the value set of the base type to the values for which an `xs:enumeration` element with a matching `value` attribute is given.
Parent element	`xs:restriction`
Child elements	None
Attributes	`value`

Name	Type	Description
`value`	`xs:string`	Allows you to specify an allowed value of the value space of the enumeration type to be defined.

Attributes

Example

See the **example** for the `xs:restriction` element.

15 xs:extension

Purpose	This element enables you to define simple content by adding attributes for an element.
Parent element	`xs:complexContent`, `xs:simpleContent`
Child elements	`xs:all`, `xs:anyAttribute`, `xs:attribute`, `xs:attributeGroup`, `xs:choice`, `xs:group`, `xs:sequence`
Attributes	`base`

Name	Type	Description
`base`	`xs:QName`	When you derive a new simple type from an existing simple type (the base type) using the `xs:extension` element, you must use the `base` attribute to specify the base type to be used for the derivation. See the **example** below.

Attributes

Example

The following example shows the use of the `xs:extension` element to extend an element of complex type with two attributes, namely `form` and `brand`. In this example a base type `xs:string` (defined by **XML Schema**) is used for derivation of a new type by extension.

```
<xs:extension base = "xs:string">
  <xs:attribute name = "form" type = "xs:string" use = "required"></xs:attribute>
  <xs:attribute name = "brand" type = "xs:string"></xs:attribute>
</xs:extension>
```

16 xs:field

Purpose	This element defines a component of the identity constraint (`xs:key`, `xs:keyref`, `xs:unique`).
Parent element	`xs:key`, `xs:keyref` or `xs:unique`
Child elements	`xs:annotation`
Attributes	`id`, `xpath`

Name	Type	Description
`id`	`xs:ID`	Assigns an identifier to the schema node.
`xpath`		A list of one or more XPath expressions that select the possible nodes that contribute to the values of the identity constraint. For details of the syntax, see **section 3.11.6 Constraints on Identity-constraint Definition Schema Components** of the W3C documentation **XML Schema Part 1: Structures**. For an example, see **section 5.1 Specifying Uniqueness** of the W3C documentation **XML Schema Part 0: Primer**.

Attributes

17 xs:fractionDigits

Purpose	This element specifies the number of fraction digits of a numeric data field (in values of datatypes derived from decimal). It indicates the maximum number of digits in the fractional part. For additional details, see the section *Defining Simple Types* of the *Tamino XML Schema User Guide* and references therein.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	`xs:nonNegativeInteger`	The value of the `value` attribute specifies how many digits the fractional part of the decimal value may have.

Attributes

Example

See **Example 2** for `xs:restriction`.

18 xs:group

This element is used in two different contexts:

- As a model group definition in `xs:schema`:

Purpose	In this context, the element defines a named model group to be reused later on in the schema definition. A named model group is a group of elements belonging together that is given a common name for later reference. See the section *Model Group Definition Schema Component* in **W3C XML Schema - Part 1**.
Parent element	`xs:schema`
Child elements	`xs:all`, `xs:choice`, `xs:element`, `xs:sequence`
Attributes	`name`

Name	Type	Description
`name`	`xs:NCName`	The `name` attribute is mandatory. Its value is the name that identifies the group of elements.

Attributes

- As a model group reference in `xs:choice`, `xs:sequence` or `xs:complexType`:

Purpose	This element references a predefined model group. This leads to the insertion of the whole model group at the current location in the schema.
Parent element	`xs:choice`, `xs:complexType`, `xs:extension`, `xs:sequence`
Child elements	None
Attributes	`ref`, `minOccurs`, `maxOccurs`

Name	Type	Description
ref	xs:QName	A global group can be referenced in other declarations in the same schema using the ref attribute of the xs:group element. The value of the ref attribute must refer to a global group. If you specify a ref attribute, you may not specify a name attribute on that group declaration. This attribute is mandatory.
minOccurs	xs:nonNegativeInteger	This attribute is optional. It specifies the minimum number of occurrences of the group.
maxOccurs	xs:string	This attribute is optional. It specifies the maximum number of occurrences of the group.

Attributes

Example

```
<xs:group name="address">
  <xs:element name="firstname" type="xs:string" />
  <xs:element name="lastname" type="xs:string" />
  <xs:element name="street" type="xs:string" />
  <xs:element name="town" type="xs:string" />
  <xs:element name="zip" type="xs:string" />
</xs:group>
<xs:element name="letter">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="sender">
        <xs:complexType>
          <xs:group ref="address" />
        </xs:complexType>
      </xs:element>
      <xs:element name="addressee">
        <xs:complexType>
          <xs:group ref="address" />
        </xs:complexType>
      </xs:element>
      <xs:element name="text" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

19 xs:import

Purpose	This element is used for schema composition across multiple namespaces. Its purpose is to make definitions from a *different* namespace visible in the importing schema. The schema to be imported must already be defined in the Tamino database. For example, to import a schema named *ImportedSchema.tsd*, specify an `xs:import` element with a `schemaLocation` attribute like this: `../<collection name>/ImportedSchema.tsd` where `<collection name>` is the name of the collection containing the referenced schema. **Note:** 1. The `namespace` attribute and the `schemaLocation` attribute are not mutually exclusive. 2. Multiple `xs:import` elements are allowed under one `xs:schema` element. They may be necessary to import multiple namespaces into one schema.
Parent element	`xs:schema`
Child elements	None
Attributes	`namespace`, `schemaLocation`
Restriction	The `targetNamespace` of the importing schema and the `targetNamespace` of the imported schema must not be the same. In particular, only one of these may have the value "absent".

Name	Type	Description
namespace	xs:anyURI	This attribute specifies the URI of the namespace that is to be imported. It must match the targetNamespace attribute of the imported schema. If this attribute is not specified, the schemaLocation attribute must be specified, and it must refer to a schema without a target namespace.
schemaLocation	xs:anyURI	This attribute specifies the location of the schema that is to be imported. To locate a schema stored in the same database, the structure of the schemaLocation attribute can be any of the following: ■ *../collectionName/schemaName* ■ *./schemaName* ■ *schemaName* **Note:** *./schemaName* and *schemaName* are always equivalent.

Attributes

Example 1

This example shows the use of the xs:import element. It shows how to import a namespace with the prefix html: to be able to define a documentation paragraph in a schema document according to **XHTML** (as defined in *http://www.w3.org/1999/xhtml/*):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
            xmlns:html="http://www.w3.org/1999/xhtml"
            targetNamespace="http://www.w3.org/2001/XMLSchema">

 <xs:import namespace="http://www.w3.org/1999/xhtml"/>
 .
 .
 .
 <xs:complexType name="myType">
  <xs:sequence>
   <xs:element ref="html:p" minOccurs="0"/>
  </xs:sequence>
 .
 .
 .
 </xs:complexType>
 .
 .
 .
```

Example 2: `namespace` **and** `schemaLocation` **attributes**

The following schema illustrates the simultaneous use of the `namespace` and `schemaLocation` attributes of the `xs:import` element:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  ↩
targetNamespace="http://www.softwareag.com/tamino/doc/examples/models/jazz/encyclopedia"
  xmlns:e="http://www.softwareag.com/tamino/doc/examples/models/jazz/encyclopedia"
  xmlns:i="http://www.softwareag.com/tamino/doc/examples/models/jazz/instruments"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:import
   namespace="http://www.softwareag.com/tamino/doc/examples/models/instruments"
   schemaLocation="../commonMusic/instruments.xsd"/>
    <xs:element name="e:jazzMusician">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="e:name" type="e:tName"/>
          .
          .
          .
          <xs:element name="e:plays" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:choice>
                <xs:element ref="i:saxophone"/>
              </xs:choice>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        .
        .
        .
      </xs:complexType>
    </xs:element>
    .
    .
    .
</xs:schema>
```

This example schema defines three additional namespaces: `e`,`xs` and `tsd`. It uses a `targetNamespace` declaration in the `xs:schema` element.

Note: The schema *instruments.xsd* was previously defined in the "commonMusic" collection (which is a sibling of the current collection) and is referenced via a relative value in the `schemaLocation` attribute.

20 xs:include

Purpose	This element is used to make additional definitions from included schema for the *same* namespace visible in the including schema. In Tamino, an `xs:include` element must resolve to a schema already defined in the *same database*. The `schemaLocation` attribute specifies the location of the schema. For example, to include a schema *IncludedSchema.tsd*, specify an `xs:include` element with the following `schemaLocation` attribute value: `../<collection name>/IncludedSchema.tsd` where *<collection name>* is the name of the collection containing the referenced schema. **Note:** 1. There can be an arbitrary number of `xs:include` elements, in order to compose a set of schema documents for the same target namespace. 2. For more information about including other schemas, refer to the W3C XML Schema documentation, especially the section **Assembling a schema for a single target namespace from multiple schema definition documents**.
Parent element	`xs:schema`
Child elements	None
Attributes	`schemaLocation`
Restrictions	Tamino does not import the chameleon include (see **http://www.w3.org/2001/05/xmlschema-rec-comments.html#pfiIdConstInclude**. Chameleon include means that a schema "S2" without a target namespace is included into a schema "S1" with a target namespace, whereby the included schema "S2" inherits the target namespace of "S1".

Name	Type	Description
schemaLocation	xs:anyURI	To locate a schema stored in the same database the structure of the schemaLocation attribute can be any of the following: ■ *../collectionName/schemaName* ■ *./schemaName* ■ *schemaName* **Note:** *./schemaName* and *schemaName* are always equivalent.

Attributes

Example

The following schema, entitled "master", uses an `xs:include` element to include the schema entitled "slave".

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="master">
        <tsd:collection name="include"/>
        <tsd:doctype name="types">
          <tsd:logical>
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:include schemaLocation="slave"/>
</xs:schema>
```

The following schema is the "slave" schema, which is included by the "master" schema shown above.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="slave">
        <tsd:collection name="include"/>
      </tsd:schemaInfo>
    </xs:appinfo>
```

```
  </xs:annotation>
  <xs:element name="types">
    <xs:complexType>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element name="decimal" type="xs:decimal">
          <xs:annotation>
            <xs:appinfo>
              <tsd:elementInfo>
                <tsd:physical>
                  <tsd:native>
                    <tsd:index>
                      <tsd:standard/>
                    </tsd:index>
                  </tsd:native>
                </tsd:physical>
              </tsd:elementInfo>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

21 xs:key

Purpose	Defines an identity constraint of type key. The values of an identity constraint are extracted from an XML fragment that is validated against the parent element declaration. They must be unique, and they must be complete, i.e. each item in the tuple must be set. For details of the construction of the tuples representing the values of an identity constraint, please see **section 3.11.4 Identity-constraint Definition Validation Rules** of the W3C documentation **XML Schema Part 1: Structures**.
Parent element	`xs:element`
Child elements	`xs:annotation`, `xs:field`, `xs:selector`
Attributes	`id`, `name`

Name	Type	Description
`id`	`xs:ID`	Assigns an identifier to the schema node.
`name`	`xs:NCName`	The names of all identity constraints (**xs:key**, **xs:keyref** and **xs:unique**) must be unique within a schema document. An **xs:key** identity constraint can be referenced by an **xs:keyref** identity constraint that cites the name of the **xs:key** identity constraint in the `refer` attribute.

Attributes

22 xs:keyref

Purpose	Defines an identity constraint of type xs:keyref. This provides for constraints that are similar to referential integrity. The scope of the constraint is a single XML document. Each tuple value of an xs:refkey identity constraint must match the corresponding value of the identity constraint of type **xs:key** or **xs:unique** that is referenced by the `refer` attribute. For details of the construction of the tuples representing the values of an identity constraint, please see **section 3.11.4 Identity-constraint Definition Validation Rules** of the W3C documentation **XML Schema Part 1: Structures**.
Parent element	`xs:element`
Child elements	`xs:annotation`, `xs:field`, `xs:selector`
Attributes	`id`, `name`, `refer`

Name	Type	Description
`id`	`xs:ID`	Assigns an identifier to the schema node.
`name`	`xs:NCName`	The names of all identity constraints (**xs:key**, **xs:keyref** and **xs:unique**) must be unique within a schema document.
`refer`	`xs:QName`	The QName of a referenced identity constraint of type **xs:key** or **xs:unique**.

Attributes

23 xs:length

Purpose	Specifies a length constraint for a data item. The value of the constraint is specified in the `value` attribute.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`
Restrictions	The `xs:length` element may not be specified together with the `xs:maxLength` element or the `xs:minLength` element.

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	`xs:nonNegativeInteger`	The `value` attribute specifies the length of the restricted type in number of octets of binary data (if the type is `hexBinary` or `base64Binary`) or number of characters otherwise.

Attributes

Example

The following example shows a simple type definition of a string type based on a restriction using the `xs:length` element. The defined type only accepts strings that are 8 characters long:

```
<xs:simpleType name="code">
   <xs:restriction base="xs:string">
     <xs:length value="8"/>
   </xs:restriction>
</xs:simpleType>
```

24 xs:list

Purpose	Creates a new simple type of variety list. Valid lexical values are whitespace-separated lists of items. Each item is validated against the item type. The following constraints apply: ■ Elements and attributes that have a datatype with variety list cannot be mapped: ■ Elements and attributes that have a datatype with variety list cannot be used in a tsd:unique constraint.
Parent element	`xs:simpleType`
Child elements	`xs:annotation`, `xs:simpleType`
Attributes	`id`, `itemType`

Name	Type	Description
`id`	`xs:ID`	Assigns an identifier to the schema node.
`itemType`	`xs:QName`	A reference to a global simple type definition resembling the item type. This attribute and the **xs:simpleType** child node, which can also be used to define the item type, are mutually exclusive.

Attributes

25 xs:maxExclusive

Purpose	This element is a restriction facet used to specify an exclusive upper bound as a constraint of the value space for a data item. The actual value of the constraint is given by the `value` attribute.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`
Restrictions	The following restrictions apply to the `xs:maxExclusive` element: ■ The `xs:maxExclusive` element may not be specified simultaneously with the `xs:maxInclusive` element. ■ The `xs:maxExclusive` must be valid with respect to any `xs:minExclusive` or `xs:minInclusive` facet also specified. ■ The `xs:maxExclusive` element must be chosen in such a way that all values allowed for the restricted type also belong to the value space of the base type. ■ The following update schema constraint applies to the `xs:maxExclusive` element: The value of `xs:maxExclusive` is not permitted to become more restrictive after update schema.

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	The type of the `xs:maxExclusive` element depends on the base of the restriction.	The `value` attribute specifies the maximum value of the restricted type. The exact value of the `value` attribute is excluded from the value space.

Attributes

Example

The following example shows a simple type definition based on a restriction using the `xs:maxExclusive` element:

```
<xs:simpleType name="upto50">
  <xs:restriction base="xs:integer">
    <xs:maxExclusive value="51"/>
  </xs:restriction>
</xs:simpleType>
```

The value space of the simple type “upto50” is restricted to values less than 51. This means 50 is the largest allowed element in this value space.

26 xs:maxInclusive

Purpose	This element is a restriction facet used to specify an inclusive upper bound as a constraint of the value space for a data item. The value of the constraint is given by the `value` attribute.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`
Restrictions	The following restrictions apply for the `xs:maxInclusive` element: ■ The `xs:maxInclusive` element may not be specified simultaneously with the `xs:maxExclusive` element. ■ The `xs:maxInclusive` must be valid with respect to any `xs:minInclusive` or `xs:minExclusive` facet also specified. ■ The `xs:maxInclusive` element must be chosen in such a way that all values allowed for the restricted type also belong to the value space of the base type. ■ The following update schema constraint applies to the `xs:maxInclusive` element: The value of `xs:maxInclusive` is not permitted to become more restrictive after update schema.

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	The type of the `xs:maxInclusive` element depends on the base of the restriction.	The `value` attribute specifies the maximum value of the restricted type. The exact value of the `value` attribute is included within the value space.

Attributes

Example

The following example shows a simple type definition based on a restriction using the `xs:maxInclusive` element:

```
<xs:simpleType name='upto40'>
  <xs:restriction base='xs:integer'>
    <xs:maxInclusive value='40'/>
  </xs:restriction>
</xs:simpleType>
```

The value space of the simple type “upto40” is restricted to values less than or equal to 40. This means 40 is the largest allowed element in this value space.

27 xs:maxLength

Purpose	This element is a restriction facet that specifies an inclusive maximum length constraint for a data item. The actual value of the constraint is given by the `value` attribute.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`
Restrictions	The following restrictions apply: ■ The `xs:maxLength` element may not be specified simultaneously with the `xs:length` element. ■ If an `xs:maxLength` element is specified along with the `xs:minLength` element in the same restriction, the `value` attribute of the `xs:minLength` element must be less than or equal to the `value` attribute of the `xs:maxLength` element. ■ The following update schema constraint applies to the `xs:maxLength` element: The value of `xs:maxLength` is not permitted to become more restrictive after update schema. This means that if an `xs:maxLength` element is specified for the base type, the `value` attribute of this element in the derivation must not be greater than the `value` attribute of the base type.

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	`xs:nonNegativeInteger`	The `value` attribute specifies the maximum length of the restricted type.

Name	Type	Description
		The value of the `xs:maxLength` facet is measured in different units, depending on the base type of the restriction: ■ **Binary data** (such as "hexBinary" or "base64Binary"): Octets ■ **List data** (such as "IDREFS", "ENTITIES", "NMTOKENS"): Number of list items ■ **String types:** Number of characters

Attributes

Example

The following example shows a simple type definition based on a restriction using the `xs:maxLength` element. The string length is restricted to 80 characters:

```
<xs:simpleType name='input-field'>
  <xs:restriction base='xs:string'>
    <xs:maxLength value='80'/>
  </xs:restriction>
</xs:simpleType>
```

28 xs:minExclusive

Purpose	This element is a restriction facet that specifies an exclusive lower bound as a constraint of the value space for a data item. The value of the constraint is given by the `value` attribute.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`
Restrictions	The following restrictions apply for the `xs:minExclusive` element: ■ The `xs:minExclusive` element may not be specified simultaneously with the `xs:minInclusive` element. ■ The value specified by `xs:minExclusive` must be valid with respect to the value specified by any relevant `xs:maxExclusive` or `xs:maxInclusive` facet. ■ The `xs:minExclusive` element must be chosen in such a way that all values allowed for the restricted type also belong to the value space of the base type. ■ The following update schema constraint applies to the `xs:minExclusive` element: The value of `xs:minExclusive` is not permitted to become more restrictive after update schema.

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	The type of the `xs:minExclusive` element depends on the base of the restriction.	The `value` attribute specifies the minimum value of the restricted type. The exact value of the `value` attribute is excluded from the value space.

Attributes

Example

The following example shows a simple type definition based on a restriction using the `xs:minExclusive` element:

```
<xs:simpleType name="notlessthan20">
  <xs:restriction base="integer">
    <xs:minExclusive value="19"/>
  </xs:restriction>
</xs:simpleType>
```

The value space of the simple type “notlessthan20” is restricted to integer values greater than 19. This means 20 is the smallest allowed element in this value space.

29 xs:minInclusive

Purpose	This element is a restriction facet that specifies an inclusive lower bound as a constraint of the value space for a data item. The actual value of the constraint is given by the `value` attribute.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`
Restrictions	The following restrictions apply for the `xs:minInclusive` element: ■ The `xs:minInclusive` element may not be specified simultaneously with the `xs:minExclusive` element. ■ The `xs:minInclusive` must be valid with respect to any `xs:maxInclusive` or `xs:maxExclusive` facet also specified. ■ The `xs:minInclusive` element must be chosen in such a way that all values allowed for the restricted type also belong to the value space of the base type. ■ The following update schema constraint applies to the `xs:minInclusive` element: The value of `xs:minInclusive` is not permitted to become more restrictive after update schema.

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	The type of the `xs:minInclusive` element depends on the base of the restriction.	The `value` attribute specifies the minimum value of the restricted type. The exact value of the `value` attribute is included from the value space.

Attributes

Example

The following example shows a simple type definition based on a restriction using the `xs:minInclusive` element:

```
<xs:simpleType name="notlessthan30">
  <xs:restriction base="integer">
    <xs:minInclusive value="30"/>
  </xs:restriction>
</xs:simpleType>
```

The value space of the simple type “notlessthan30” is restricted to values greater than or equal to 30. This means 30 is the smallest allowed element in this value space.

30 xs:minLength

Purpose	This element is a restriction facet used to specify an inclusive minimum length constraint for a data item. The actual value of the constraint is given by the `value` attribute.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`
Restrictions	The following restrictions apply: ■ The `xs:minLength` element may not be specified simultaneously with the `xs:length` element. ■ If an `xs:maxLength` element is specified along with the `xs:minLength` element in the same restriction, the `value` attribute of the `xs:minLength` element must be less than or equal to the `value` attribute of the `xs:maxLength` element. ■ The following update schema constraint applies to the `xs:minLength` element: The value of `xs:minLength` is not permitted to become more restrictive after update schema. This means that if an `xs:minLength` element is specified for the base type, the `value` attribute of this element in the derivation must not be less than the `value` attribute of the base type.

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	`xs:nonNegativeInteger`	The value of the `xs:minLength` facet is measured in different units depending on the base type of the restriction:

Name	Type	Description
		■ **Binary data** (such as "hexBinary" or "base64Binary"): Octets ■ **List data** (such as "IDREFS", "ENTITIES", "NMTOKENS"): Number of list items ■ **String types:** Number of characters The `value` attribute specifies the minimum length of the restricted type in number of octets of binary data (if the type is `hexBinary` or `base64Binary`) or number of characters otherwise.

Attributes

Example

The following example shows a simple type definition based on a restriction using the `xs:minLength` element:

```
<xs:simpleType name="input-field">
  <xs:restriction base="xs:string">
    <xs:minLength value="8"/>
  </xs:restriction>
</xs:simpleType>
```

This defines a string to be used as an input field with a length of at least 8 characters.

31 xs:notation

Purpose	This element is used for reference by enumeration values in a restriction of the `xs:NOTATION` type.
Parent element	`xs:schema`
Child elements	None
Attributes	`name`, `public`, `system`

Name	Type	Description
`name`	`xs:NCName`	The `name` attribute specifies the name of the notation.
`public`	`xs:token`	The `public` attribute contains the public identifier.
`system`	`xs:anyURI`	The `system` attribute is a reference to a URI containing the system identifier.

Attributes

Example

The following example is a complete schema definition showing the use of the `xs:notation` element. It shows an example schema containing a complex type definition and two notations:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
  <xs:notation name = "myNote" public = "myNotationProgramm"/>
  <xs:notation name = "gif" public = "gifviewer"/>
  <xs:element name = "anElement">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base = "xs:string">
          <xs:attribute name = "otherText" type = "xs:string"/>
          <xs:attribute name = "myNote">
            <xs:simpleType>
              <xs:restriction base = "xs:NOTATION">
```

```
                    <xs:enumeration value = "gif"/>
                    <xs:enumeration value = "myNote"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:schema>
```

32 xs:pattern

Purpose	This element is a restriction facet for pattern matching, i.e. for constraining the value space in such a way that only literals matching the specified pattern (regular expression) are allowed for the lexical space.
Parent element	`xs:restriction`
Child elements	None
Attributes	`value`
Restrictions	The following restriction applies for the `xs:pattern` element: ■ **Multiple `xs:pattern` Elements** Multiple `xs:pattern` elements are permitted under one `xs:restriction` element. However, specifying two `xs:pattern` elements within the scope of one `xs:restriction` element is not the same as specifying two `xs:pattern` elements within separate `xs:restriction` elements; a logical "OR" operation is performed with `xs:pattern` elements belonging to the same `xs:restriction` element.

Name	Type	Description
`value`	`xs:string`	The `value` attribute allows you to specify a regular expression as defined in *http://www.w3.org/TR/xmlschema-2/#dt-regex* as a pattern that all members of the value space must match.

Attributes

Example

This simple example shows the application of the `xs:pattern` element for the definition of a German zip code, which consists of five decimal digits:

```
<xs:simpleType name="german-zipcode">
  <xs:restriction base='string'>
    <xs:pattern value='[0-9]{5}'/>
  </xs:restriction>
</xs:simpleType>
```

33 xs:restriction

Purpose	This element allows restrictions to be imposed on the datatype specified by the `base` attribute. You can choose between twelve constraining facets, which are listed below in the child elements section of this table.
Parent element	`xs:complexContent`, `xs:simpleType`
Child elements	`xs:enumeration`, `xs:fractionDigits`, `xs:length`, `xs:maxExclusive`, `xs:maxInclusive`, `xs:maxLength`, `xs:minExclusive`, `xs:minInclusive`, `xs:minLength`, `xs:pattern`, `xs:totalDigits`, `xs:whiteSpace`. These child elements represent the constraining facets that can be applied to restrict the value space of the new data type to be defined.
Attributes	`base`

Name	Type	Description
`base`	`xs:QName`	When you use the `xs:restriction` element to derive a new simple type from an existing simple type (the base type), the `base` attribute specifies the type from which the derived type is derived. See the example below in which a new type is derived from `xs:NMTOKEN` by restriction.

Attributes

Example 1

This example shows a type definition of an enumeration type whose lexical space contains the three values given as value arguments for the `xs:enumeration` element:

```
<xs:restriction base = "xs:NMTOKEN">
  <xs:enumeration value = "instrumentalist"/>
  <xs:enumeration value = "jazzSinger"/>
  <xs:enumeration value = "jazzComposer"/>
</xs:restriction>
```

Example 2

This example defines a datatype for a financial application for expressing numerical values for monetary amounts ranging from -999,999,999.99 Euros to 999,999,999.99 Euros:

```
<xs:simpleType name="amount">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="11"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>
```

34 xs:schema

Purpose	This element enables you to define a whole schema in TSD.
Parent element	None; `xs:schema` is the root element of the logical schema tree.
Child elements	`xs:annotation`, `xs:attribute`, `xs:attributeGroup`, `xs:complexType`, `xs:element`, `xs:group`, `xs:import`, `xs:include`, `xs:notation`, `xs:simpleType`
Attributes	`attributeFormDefault`, `blockDefault`, `elementFormDefault`, `finalDefault`, `targetNamespace`, `version` and namespace declarations
Restrictions	The following restriction applies for the `xs:schema` element: ■ If a `targetNamespace` attribute is present in an `xs:schema` element, it must not be empty.

Name	Type	Description
`attributeFormDefault`	`xs:NMTOKEN`	This attribute affects the representation of attributes in instance documents and the checking of the qualification of locally-declared attributes during the schema validation process. It decides whether the appearance of locally-declared attributes in an instance document must be qualified by a namespace specification. It can be set to the values "qualified" or "unqualified". Setting the `attributeFormDefault` attribute to the value "unqualified" means that all local attributes are excluded from the according namespace (that was specified with the `targetNamespace` attribute). This is the default setting for the `attributeFormDefault` attribute. Otherwise, all attributes matching locally-defined attributes must be qualified by a namespace declaration. The value of the

		attributeFormDefault attribute can be overridden using the form attribute of locally-defined attributes.
blockDefault	xs:blockSet	This attribute specifies the default for the block attribute of **xs:element** and **xs:complexType** in the current schema document. Please refer to the documentation of those elements for a list of valid values.
elementFormDefault	xs:NMTOKEN	This attribute affects the representation of elements in instance documents and the checking of the qualification of locally-declared elements during the schema validation process. It decides whether the appearance of locally-declared elements in an instance document must be qualified by a namespace specification. It can be set to the values "qualified" or "unqualified". Setting the elementFormDefault attribute to the value "unqualified" means that all local elements are excluded from the namespace of the schema (that was specified with the targetNamespace attribute). This is the default setting for the elementFormDefault attribute. Otherwise, all elements matching locally-defined elements must be qualified by a namespace declaration. The value of elementFormDefault can be overridden using the form attribute of locally-defined elements.
finalDefault	xs:fullDerivationSet	This attribute specifies the default for the final attribute of **xs:element** and **xs:complexType** in the current schema document. Please refer to the documentation of those elements for a list of valid values.
targetNamespace	xs:anyURI	This attribute defines a target namespace for the schema. This is done by specifying a URI that uniquely identifies the namespace. There must be a corresponding namespace declaration using the same URI.
version	xs:token	This attribute specifies version information.

Table of attributes

For more information about target namespaces and qualification in xs:schema definitions, see ***Advanced Concepts I: Namespaces, Schemas & Qualification*** **in W3C's introductory document** ***XML Schema Part 0: Primer***, especially the section ***Target Namespaces & Unqualified Locals***.

Examples

There are three large examples in the appendix of the *Tamino XML Schema User Guide*.

35 xs:selector

Purpose	Defines the selector for an identity constraint (`xs:key`, `xs:keyref`, `xs:unique`).
Parent element	`xs:key`, `xs:keyref` or `xs:unique`
Child elements	`xs:annotation`
Attributes	`id`, `xpath`

Name	Type	Description
`id`	`xs:ID`	Assigns an identifier to the schema node.
`xpath`		A list of one or more XPath expressions that select the element nodes, each of which contributes a single tuple value to the current identity constraint. For details of the syntax, see in section 3.11.6 **Schema Component Constraint: Selector Value OK** of the W3C documentation **XML Schema Part 1: Structures**. For an example, see **section 5.1 Specifying Uniqueness** of the W3C documentation **XML Schema Part 0: Primer**.

Attributes

36 xs:sequence

Purpose	This element (the “sequence” particle) is used for expressing sequences in the content model of a complex type used for element definitions. This means that the child elements must appear in a given order.
Parent element	`xs:choice`, `xs:complexType`, `xs:extension`, `xs:group`, `xs:sequence`
Child elements	`xs:any`, `xs:choice`, `xs:element`, `xs:group`, `xs:sequence`
Attributes	`minOccurs`, `maxOccurs`
Status	Optional.

Name	Type	Description
`minOccurs`	`xs:nonNegativeInteger`	Together with the `maxOccurs` attribute described below, this attribute expresses multiplicity in TSD (and **XML Schema**) in sequence elements within a content model. The value of a `minOccurs` attribute in an `xs:sequence` declaration determines the minimum number of occurrences of that sequence. The value of `minOccurs` must be a non-negative integer. The default value for `minOccurs` is 1.
`maxOccurs`	`xs:nonNegativeInteger` or “unbounded”	Together with the `minOccurs` attribute described above, this attribute expresses multiplicity in TSD (and **XML Schema**) in sequence elements within a content model. The value of a `maxOccurs` attribute in an `xs:sequence` declaration determines the maximum number of occurrences of that sequence. The value of `maxOccurs` must be a non-negative integer or the value "unbounded". The default value for `maxOccurs` is 1.

Attributes

Example 1

In the following example, within a `p_record` element the five children from `p_firstname` to `p_country` must each appear exactly once in the order:

1. p_firstname
2. p_lastname
3. p_address
4. p_city
5. p_country

```
<xs:element name="p_record">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="p_firstname" type="xs:string"/>
      <xs:element name="p_lastname" type="xs:string"/>
      <xs:element name="p_address" type="xs:string"/>
      <xs:element name="p_city" type="xs:string"/>
      <xs:element name="p_country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example 2

This example illustrating the use of the xs:sequence element is taken from the "patient" schema:

```
  <xs:element name = "patient">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref = "name" minOccurs = "0"></xs:element>
        <xs:element ref = "sex"></xs:element>
        <xs:element ref = "born" minOccurs = "0"></xs:element>
        <xs:element ref = "address" minOccurs = "0"></xs:element>
        <xs:element ref = "occupation" minOccurs = "0"></xs:element>
        <xs:element ref = "insurance" minOccurs = "0"></xs:element>
        <xs:element ref = "nextofkin" minOccurs = "0"></xs:element>
        <xs:element ref = "submitted"></xs:element>
        <xs:element ref = "examination" minOccurs = "0"
                                        maxOccurs = "unbounded"></xs:element>
        <xs:element ref = "therapy" minOccurs = "0"></xs:element>
        <xs:element ref = "result" minOccurs = "0"></xs:element>
        <xs:element ref = "remarks" minOccurs = "0"
                                    maxOccurs = "unbounded"></xs:element>
      </xs:sequence>
      <xs:attribute name = "ID" type = "xs:string"></xs:attribute>
    </xs:complexType>
  </xs:element>
```

It shows a sequence with a list of optional elements and two required elements, i.e. sex and submitted.

37 xs:simpleContent

Purpose	This element enables you to define simple content within complex type definitions as defined in the *W3C XML Schema*. In particular, it enables you to define new complex types by extension, i.e. by adding attributes.
Parent element	`xs:complexType`
Child elements	`xs:extension`
Attributes	None

Example

This example of a definition of a complex type based on simple content is taken from a track list of the "Jazz" schema:

```
<xs:simpleContent>
  <xs:extension base = "xs:normalizedString">
    <xs:attribute name = "duration"
                  type = "xs:unsignedShort"
                  use = "required"/>
  </xs:extension>
</xs:simpleContent>
```

38 xs:simpleType

Purpose	This element enables you to define a simple data type in the Tamino Schema Language.
Parent element	`xs:attribute`, `xs:element`, `xs:list`, `xs:schema` (only in the case of named types), `xs:union`
Child elements	`xs:list`, `xs:restriction`, `xs:union`
Attributes	`final`, `name`

Name	Type	Description
`final`		This attribute, in conjunction with the **xs:schema** element's `finalDefault` attribute (if the `final` attribute is not specified, the `finalDefault` attribute value takes effect), is used when performing schema checks. It specifies the methods that may be used to derive new types from the current simple type. The value is a whitespace-separated list of: **restriction** No type may be derived from this simple type by `restriction`. **list** No type may be derived from this simple type by the creation of a list type. **union** No type may be derived from this simple type by the creation of a union type. Alternatively, the special value "#all" prevents all methods of deriving a new type from the current simple type definition.
`name`	`xs:NCName`	For purposes of reuse, it is possible to specify a name for a simple type definition to be able to reference it later by this name. This is done with the `name` attribute of the `xs:simpleType` element. This attribute is mandatory for the definition of named simple types. For anonymous simple types, however, it must be omitted.

Attributes

Example 1

This example defines a simple type based on a restriction that only the three values "instrumentalist", "jazzSinger" and "jazzComposer" are allowed for the defined type:

```
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema"
  xmlns:tsd = "http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
 .
 .
 .
 <xs:simpleType name="role">
  <xs:restriction base = "xs:NMTOKEN">
    <xs:enumeration value = "instrumentalist"/>
    <xs:enumeration value = "jazzSinger"/>
    <xs:enumeration value = "jazzComposer"/>
  </xs:restriction>
 </xs:simpleType>
 .
 .
 .
</xs:schema>
```

Example 2

Here is another example showing anonymous simple type definitions:

```
<xs:element name = "price">
 <xs:simpleType>
   <xs:restriction base='decimal'>
     <xs:totalDigits value='9'/>
     <xs:fractionDigits value='2'/>
   </xs:restriction>
 </xs:simpleType>
</xs:element>
```

39 xs:totalDigits

Purpose	This element is used for specifying information about the total number of digits of a numeric data field (in values of datatypes derived from decimal). It indicates the maximum allowed value for the number of digits. For additional details, see the section *Defining Simple Types* of the *Tamino XML Schema User Guide* and references therein.
Parent element	`xs:restriction`
Child elements	None
Attributes	`fixed`, `value`

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`value`	`xs:positiveInteger`	The value of the `value` attribute specifies the total number of digits the decimal value may have.

Attributes

Example

The following example demonstrates the use of the `xs:totalDigits` element within a simple type definition by restriction. A simple type based on the decimal type is defined which can store decimal numbers with up to 6 digits:

```
<xs:simpleType>
  <xs:restriction base='xs:decimal'>
    <xs:totalDigits value='6'/>
  </xs:restriction>
</xs:simpleType>
```

40 xs:union

Purpose	Creates a new simple type of variety union. The lexical value must validate against one (or more) of the member types. The member types form an ordered list, and are defined by: the `memberTypes` attribute; the `xs:simpleType` child node.
Parent element	`xs:simpleType`
Child elements	`xs:annotation`, `xs:simpleType`
Attributes	`id`, `memberTypes`

Name	Type	Description
`id`	`xs:ID`	Assigns an identifier to the schema node.
`memberTypes`	`xs:QName`	A whitespace-separated list of lexical QNames referencing global simple type declarations that contribute to the member types of the new simple type to be constructed.

Attributes

41 xs:unique

Purpose	Defines an identity constraint of type xs:unique. The values of an identity constraint are extracted from an XML fragment that is validated against the parent element declaration. The values of an xs:unique identity constraint must be unique. For details of the construction of the tuples representing the values of an identity constraint, please see **section 3.11.4 Identity-constraint Definition Validation Rules** of the W3C documentation **XML Schema Part 1: Structures**.
Parent element	`xs:element`
Child elements	`xs:annotation`, `xs:field`, `xs:selector`
Attributes	`id`, `name`

Name	Type	Description
`id`	`xs:ID`	Assigns an identifier to the schema node.
`name`	`xs:NCName`	The names of all identity constraints (**xs:key**, **xs:keyref** and **xs:unique**) must be unique within a schema document. An **xs:unique** identity constraint can be referenced by an **xs:keyref** identity constraint that cites the name of the **xs:unique** identity constraint in the `refer` attribute.

Attributes

42 xs:whiteSpace

Purpose	This element is used for handling whitespace normalization of `xs:string` data types. The `xs:whiteSpace` element may be specified additionally to other facets. The corresponding feature of the ***W3C XML Schema*** is described in ***http://www.w3.org/TR/xmlschema-2/#dt-whiteSpace***.
Parent element	`xs:restriction`
Child elements	`xs:annotation`
Attributes	`fixed`, `id`, `value`

Name	Type	Description
`fixed`	`xs:boolean`	If "true", the value of this facet must not be changed in a derived type. Default: "false".
`id`	`xs:ID`	Assigns an identifier to the schema node.
`value`	`xs:string`	This attribute allows you to control whitespace normalization. It has one of three predefined values: ■ **"preserve"** No normalization takes place, the value remains unchanged. ■ **"replace "** Each occurrence of the ASCII characters #x9 (tab), #xA (line feed) and #xD (carriage return) is substituted by #x20 (space). ■ **"collapse "** This means the same processing as `replace`, but additionally each sequence of one or more consecutive #x20 (space) characters is converted to a single #x20 character; furthermore, leading and trailing #x20 characters are removed. For all data types other than `xs:string`, the value is fixed to "collapse".

Attributes

Example

This simple example shows the use of the `xs:whiteSpace` element:

A Appendix

This version of Tamino supports all features of the **W3C XML Schema** except for the following:

- The element `xs:redefine`;
- The so-called chameleon include. Chameleon include means that a schema "S2" without a target namespace is included into a schema "S1" with a target namespace, whereby the included schema "S2" inherits the target namespace of "S1";
- The attributes `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation`, which belong to the **W3C XML Schema** instance namespace characterized by the namespace prefix `xsi:` and the URI *http://www.w3.org/2001/XMLSchema-instance*, are simply ignored if they occur in documents that are being stored.

II Tamino Extensions to XML Schema: List of Elements and Attributes

43 Tamino Extensions to XML Schema: List of Elements and Attributes

The following table shows, in alphabetical order, the elements of the `tsd:` namespace *http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition* that are available for Tamino-specific extensions to the W3C ***XML Schema***.

Note: Attributes are described in a separate table in the description of the element to which they belong.

`tsd:accessOptions`
`tsd:accessPredicate`
`tsd:adminInfo`
`tsd:alternate`
`tsd:attributeInfo`
`tsd:caseFirst`
`tsd:caseLevel`
`tsd:collation`
`tsd:collection`
`tsd:collectionRef`
`tsd:compress`
`tsd:computed`
`tsd:content`
`tsd:created`
`tsd:default`
`tsd:delete`
`tsd:dereference`
`tsd:doctype`

tsd:elementInfo
tsd:field
tsd:french
tsd:function
tsd:ignoreUpdate
tsd:index
tsd:insert
tsd:language
tsd:logical
tsd:map
tsd:modified
tsd:multiPath
tsd:multiple
tsd:name
tsd:native
tsd:noConversion
tsd:nodeAdabasField
tsd:nodeParameter
tsd:nodeRef
tsd:nodeSQL
tsd:nonXML
tsd:normalization
tsd:objectRef
tsd:onBinaryInsert
tsd:onDelete
tsd:onInsert
tsd:onProcess
tsd:onTextInsert
tsd:onUpdate
tsd:physical
tsd:primaryKeyColumn
tsd:pure
tsd:read
tsd:reference
tsd:refers
tsd:schema
tsd:schemaInfo
tsd:server

tsd:shadowXML

tsd:standard

tsd:storeShadowOnly

tsd:strength

tsd:structureIndex

tsd:subTreeAdabas

tsd:subTreeAdabasPE

tsd:subTreeSQL

tsd:systemGeneratedIdentity

tsd:text

tsd:trigger

tsd:unique

tsd:update

tsd:version

tsd:which

tsd:xTension

44 tsd:accessOptions

Caution: The `tsd:accessOptions` element is deprecated in Tamino 4.2.

Purpose	This element controls access to the doctype with which it is associated. You can set the access option to allow or exclude read, insert, update or delete access to this doctype, depending on whether the optional `tsd:read`, `tsd:insert`, `tsd:update` or `tsd:delete` elements are present. Also refer to the following documentation of the child elements.
	
	The default for the optional element `tsd:accessOptions` is:
	
	`<tsd:accessOptions>`
	`  <tsd:read/>`
	`  <tsd:insert/>`
	`  <tsd:update/>`
	`  <tsd:delete/>`
	`</tsd:accessOptions>`

Parent element	`tsd:logical` or `tsd:nonXML` (no difference in details depending on parent)
Child elements	`tsd:read`, `tsd:insert`, `tsd:update`, `tsd:delete`
Attributes	None

Example

The following access options specify `read`, `insert`, and `update` access to the corresponding doctype but not `delete` access.

```
<tsd:doctype name="address">
  <tsd:logical>
    <tsd:content>open</tsd:content>
    <tsd:accessOptions>
      <tsd:read />
      <tsd:insert />
      <tsd:update />
    </tsd:accessOptions>
  </tsd:logical>
</tsd:doctype>
```

See also `tsd:logical`, `tsd:objectRef` and `tsd:subTreeSQL`

45 tsd:accessPredicate

This element can be used in two different contexts:

1. In the context of `tsd:objectRef`:

Purpose	In the context of `tsd:objectRef`, `tsd:accessPredicate` specifies a simple access predicate to join another doctype via `tsd:nodeRef` with the value of the current node which can be used to define filter expressions. Exactly one `tsd:accessPredicate` element is needed per `tsd:objectRef` element.
Parent element	`tsd:objectRef`
Child elements	`tsd:nodeRef`, `tsd:nodeParameter`
Attributes	`operator`
Type	`xs:complexType`

Name	Type	Description
`operator`	`xs:string`	In the context of `tsd:objectRef`, the `operator` attribute allows the specification of an operator to apply in the context of the `tsd:accessPredicate` element. Valid values are: ■ = ■ != ■ < ■ > ■ <= ■ >=

Attributes

Example

This example shows the use of the `tsd:accessPredicate` element in the context of the `tsd:objectRef` element. An object reference is set up using `/CustomerNo` as a key:

```
<tsd:objectRef>
  <tsd:collectionRef>Customers</tsd:collectionRef>
  <tsd:accessPredicate operator = "=">
    <tsd:nodeRef>/Customer/CustomerNo</tsd:nodeRef>
  </tsd:accessPredicate>
</tsd:objectRef>
```

2. In the context of `tsd:subTreeSQL`:

Purpose	In the context of `tsd:subTreeSQL`, this element specifies parameters to be used to build a “where” clause in retrieval expressions.
Parent element	`tsd:subTreeSQL`
Child elements	`tsd:nodeRef`
Attributes	None
Type	`xs:complexType` (with mixed content).

Example

For an example of this kind of usage of `tsd:accessPredicate`, see `tsd:subTreeSQL`.

46 tsd:adminInfo

Purpose	This element contains administrative information that is associated with the schema to which `tsd:schemaInfo` applies. This includes version information, creation date and modification date and server information.
Parent element	`tsd:schemaInfo`
Child elements	`tsd:version`, `tsd:created`, `tsd:modified`, `tsd:server`
Attributes	None
Restriction	This element and all of its children are generated by Tamino. It is neither necessary to provide this element by yourself nor may you modify this information.

Example

The following example shows what a `tsd:schemaInfo` element containing a `tsd:adminInfo` element could look like:

```
<tsd:schemaInfo name="mySchema">
 <tsd:collection name="ino:collection"/>
 <tsd:adminInfo>
  <tsd:server>8.0.0.1</tsd:server>
  <tsd:version>TSD</tsd:version>
  <tsd:created>2008-08-21T01:23:45.000+02:00</tsd:created>
  <tsd:modified>2008-08-22T01:23:45.000+02:00</tsd:modified>
 </tsd:adminInfo>
</tsd:schemaInfo>
```

47 tsd:alternate

Purpose	This element specifies the handling of punctuation in sorting.
Parent element	`tsd:collation`
Child elements	None
Attributes	`value`

Name	Type	Description
`value`	`xs:NMTOKEN`	This attribute can only have one of the following values: ■ **"shifted"** The value "shifted" sorts words containing punctuation marks together (e.g. bi-weekly and biweekly), that is, punctuation is ignored for levels 1-3. ■ **"nonIgnorable"** The value "nonIgnorable" (default) distinguishes these words and sorts them separately, that is, punctuation marks are considered to be significant. **Note:** This is the default root locale, i.e. the default collation attribute value according to the Unicode Collation Algorithm. However, this may be different if a specific language/locale is specified. For more information, refer to `tsd:collation` and to the collation documentation in the *XML Schema User Guide*.

Attributes

Example

Also see under `tsd:collation`.

48 tsd:attributeInfo

This element can be used in two different contexts:

1. In the context of `xs:attribute/xs:annotation/xs:appinfo`:

Purpose	This element contains information describing e.g. mapping, indexing or collation aspects of an attribute described by the *XML Schema* part of TSD. **Note:** A `tsd:attributeInfo` element may only be inserted under an `xs:attribute/xs:annotation/xs:appinfo` chain if the `xs:attribute` element of which `tsd:attributeInfo` is a successor has a `name` attribute and not a `ref` attribute.
Parent element	xs:attribute/xs:annotation/xs:appinfo
Child elements	`tsd:logical`, `tsd:physical`
Attributes	None

Example

This shows the `tsd:attributeInfo` element for native storage with standard indexing:

```
<xs:attribute name = "width">
  <xs:annotation>
    <xs:appinfo>
      <tsd:attributeInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:standard/>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:attributeInfo>
    </xs:appinfo>
```

```
  </xs:annotation>
.
.
.
</xs:attribute>
```

2. In the context of `tsd:schemaInfo`:

Purpose	The `tsd:attributeInfo` element contains information describing e.g. mapping, indexing or collation aspects of an attribute described by the **XML Schema** part of TSD.
Parent element	`tsd:schemaInfo`
Child elements	`tsd:physical`
Attributes	`context`

Name	Type	Description
`context`	`xs:string`	The `context` attribute is to be used for `tsd:attributeInfo` specified below a `schemaInfo` element. Based on its value, the `xs:element` or `xs:attribute` to which the `tsd:attributeInfo` belongs is determined. The meaning of the value is essentially the same as for the `tsd:which` element, i.e. a string of the following syntax can be specified: `("/" element name) * "/" (element name | "@" attribute name)`

Attributes

49 tsd:caseFirst

Purpose	This element controls the behavior of sorting with respect to case.
Parent element	`tsd:collation`
Child elements	None
Attributes	`value`
Type	`xs:complexType`

Name	Type	Description
`value`	`xs:NMTOKEN`	This attribute must have one of the following values: ■ **"upperFirst"** With a value of "upperFirst", words starting with uppercase are sorted together before words starting with lowercase. ■ **"lowerFirst"** A value of "lowerFirst" reverses this behavior. ■ **"off"** A value of "off" indicates that no distinction is made between uppercase and lowercase during sorting. The default value for this attribute is "off". **Note:** 1. If this option is set to either "upperFirst" or "lowerFirst", words starting with the same case are sorted together, either uppercase or lowercase first. Mixed case words (e.g. “AbC”, “aBc”) are therefore always sorted between uppercase and lowercase. 2. When the option is not specified, a default value is assumed. The default value indicated here applies to the default root locale that may be otherwise for a specific

Name	Type	Description
		language/locale. For this topic, refer to `tsd:collation` and to the collation documentation in the *XML Schema User Guide* for more information.

Attributes

Example

See under `tsd:collation`.

50 tsd:caseLevel

Purpose	This element specifies information on case levels (see below) via the `value` attribute.
Parent element	`tsd:collation`
Child elements	None
Attributes	`value`
Type	`xs:complexType`

Name	Type	Description
`value`	`xs:boolean`	Valid values for the `value` attribute are "true" and "false". The value "true" introduces a separate collation level for case differences, positioned between level 2 (secondary) and level 3 (tertiary). This is used primarily in Japanese to make the difference between small and large Kana more important than the other tertiary differences. The default is "false". **Note:** This is the default root locale, i.e. the default collation attribute value according to the Unicode Collation Algorithm. However, this may be different if a specific language/locale is specified. For more information, refer to `tsd:collation` and to the collation documentation in the *XML Schema User Guide*.

Attributes

Example

See under `tsd:collation`.

51 tsd:collation

Purpose	This is an element in the logical subtree of `tsd:elementInfo` or `tsd:attributeInfo`. It contains information about properties for the definition of a particular character sorting order (for example, regarding the ordering of umlauts in some languages). The process of arranging elements of a set into a particular order is called collation. This element and its child elements provide special support in TSD for collation-specific aspects. Refer to the collation documentation in the user guide for more information. **Note:** 1. All unspecified child elements of `tsd:collation` are set to the default values appropriate to the specified language/locale. 2. If no `tsd:language` child element is specified, the default root locale (in which the default Unicode Collation Element Table with no locale-specific tailoring is used in the Unicode Collation Algorithm [UCA]) is used. 3. It is also possible to specify no child elements at all, i.e. an empty collation element. This can be accomplished by specifying `<tsd:collation/>`. In this case, the default root locale is used with the default UCA collation attributes.
Parent element	`tsd:computed`, `tsd:logical`
Child elements	`tsd:language`, `tsd:strength`,`tsd:caseFirst`, `tsd:alternate`,`tsd:caseLevel`, `tsd:french`,`tsd:normalization`
Attributes	None
Type	`xs:complexType`

Example

The following example demonstrates the use of the `tsd:collation` element. It presets the child elements of the `tsd:collation` element with reasonable values:

```
<xs:element name = "address">
 <xs:annotation>
  <xs:appinfo>
   <tsd:elementInfo>
    <tsd:logical>
     <tsd:collation>
      <tsd:language value="de"/>
      <tsd:strength value="primary"/>
      <tsd:caseFirst value = "upperFirst"/>
      <tsd:alternate value = "shifted"/>
      <tsd:caseLevel value = "false"/>
      <tsd:french value = "false"/>
      <tsd:normalization value ="false"/>
     </tsd:collation>
    </tsd:logical>
    <tsd:physical>
     <tsd:native>
      <tsd:index>
       <tsd:standard/>
      </tsd:index>
     </tsd:native>
    </tsd:physical>
   </tsd:elementInfo>
  </xs:appinfo>
 </xs:annotation>
 .
 .
 .
</xs:element>
```

52 tsd:collection

Purpose	For a query on `xs:schema` no doctype per collection can occur, as doctypes can be deleted. In this case the implicit doctype name is *<collection name>/<schema name>*. For this situation, the collection name needs to be known. For define a doctype has to be specified.
Parent element	`tsd:schemaInfo`
Child elements	None
Attributes	`name`
Type	`xs:complexType`

Name	Type	Description
`name`	`xs:NMTOKEN`	Specifies the name of the associated collection. For XML doctypes, its name references a global element with the same name. For non-XML doctypes, no global element with that name is allowed to exist.

Attributes

53 tsd:collectionRef

Purpose	This element allows the specification of the collection containing the referenced doctype. If no other collection is specified, the current collection is used.
Parent element	`tsd:objectRef`
Child elements	None
Attributes	None
Type	`xs:NMTOKEN`

Example

This is an example of the specification of the collection containing the referenced doctype:

```
<xs:element name = "address">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:logical>
          <tsd:collation>
            <tsd:language value="de"/>
            <tsd:strength value="secondary"/>
            <tsd:alternate value = "shifted"/>
            <tsd:caseLevel value = "false"/>
            <tsd:french value = "false"/>
            <tsd:normalization value ="false"/>
          </tsd:collation>
        </tsd:logical>
        <tsd:physical>
          <tsd:native>
            <tsd:objectRef>
              <tsd:collectionRef>Customers</tsd:collectionRef>
              <tsd:accessPredicate operator = "=">
                <tsd:nodeRef>/Customer/CustomerNo</tsd:nodeRef>
              </tsd:accessPredicate>
```

```
            </tsd:objectRef>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  .
  .
  .
</xs:element>
```

54 tsd:compress

Purpose	This element specifies whether the corresponding doctype is stored in a compressed or uncompressed format.
Parent element	`tsd:physical`
Child elements	None
Attributes	None
Type	`xs:NMTOKEN`.
Restrictions	The `tsd:compress` element can have the following values: ■ **"smart"** Tamino checks the data to be stored and tries to find the best compromise between speed and storage space. This is the default and should lead to optimal behavior in most cases. ■ **"always"** Always compress as much as possible. This choice makes sense if you are primarily interested in reducing storage size. Especially for small documents, this increases retrieval time, but minimizes disk space usage. ■ **"none"** Compress large data records, but do not compress small data records. This setting is appropriate if you expect most of your documents to be reasonably small (< 8000 characters), and you want to optimize processing speed, sacrificing storage space. Large documents are not affected by this setting. ■ **"off"** Do no compression at all. ■ **"utf8"** In text documents, each character is replaced by its UTF-8 representation. This can result in a compression factor of up to 4, depending on platform and data.

Example

This example shows the use of the `tsd:compress` element to set compression to "always":

```
<tsd:schemaInfo name = "Telephone">
  <tsd:collection name = "CustomerData"></tsd:collection>
  <tsd:doctype name = "Telephone">
    <tsd:logical>
    .
    .
    .
    </tsd:logical>
    <tsd:physical>
      <tsd:compress>always</tsd:compress>
    </tsd:physical>
  </tsd:doctype>
</tsd:schemaInfo>
```

55 tsd:computed

Purpose	This element specifies the definition of a computed index for a doctype.
Parent element	`tsd:index`
Child elements	`tsd:collation`
Attributes	`name`, `type`, `function`
Type	`xs:complexType`

Name	Type	Description
`name`	`xs:NCName`	A name that must be unique among all computed indexes for this doctype.
`type`	`xs:QName`	A simple type that is predefined in XML Schema.
`function`	`xs:QName`	The QName of an XQuery function that is defined in an XQuery module.

Attributes

Example

This example shows ???the use of the `tsd:compress` element to set compression to "always":???

```
.
.
.
<tsd:doctype name="bib">
  <tsd:physical>
    <tsd:index>
      <tsd:standard>
        <tsd:computed name='compIndex-1'
                      function='p:getTitleLowerCase'
                      type='xs:string'
                      xmlns:p='http://example.computedIndex.org'/>
      </tsd:standard>
      <tsd:standard>
```

```
        <tsd:computed name='compIndex-2'
                      function='p:getAuthorFullName'
                      type='xs:string'
                      xmlns:p='http://example.computedIndex.org'/>
      </tsd:standard>
      <tsd:standard>
        <tsd:computed name='compIndex-3'
                      function='p:getAuthorsTotalNameLength'
                      type='xs:int'
                      xmlns:p='http://example.computedIndex.org'/>
      </tsd:standard>
    </tsd:index>
  </tsd:physical>
</tsd:doctype>
.
.
.
```

56 tsd:content

Purpose	This element specifies whether validation of instances of the corresponding doctype uses closed content mode or open content mode. Its use is optional. The default is closed content mode. **Note:** The `tsd:content` element can be applied both in the case of `tsd:logical` as a child of `tsd:doctype` and in the case of `tsd:logical` as a child of `tsd:shadowXML`.
Parent element	`tsd:logical`
Child elements	None
Attributes	None
Type	`xs:NMTOKEN`
Restrictions	Only the following values are allowed: ■ "**closed**" Apply closed content validation mode, i.e. adhere strictly to the validation rules as defined in the *XML Schema* recommendation. ■ "**open**" Apply open content validation mode, i.e. allow for arbitrary additional elements and attributes in the content model of any element defined in this doctype. For details, refer to *Open Content vs. Closed Content Validation*.

Example

This example shows how to specify that validation is checked for XML data stored using the schema. See also `tsd:logical`:

```
<tsd:schemaInfo name = "Telephone">
  <tsd:collection name = "CustomerData"></tsd:collection>
  <tsd:doctype name = "Telephone">
    <tsd:logical>
     <tsd:content>closed</tsd:content>
    </tsd:logical>
    <tsd:physical>
    .
    .
    .
    </tsd:physical>
  </tsd:doctype>
</tsd:schemaInfo>
```

57 tsd:created

Purpose	This element contains the creation date of the schema.
Parent element	`tsd:adminInfo`
Child elements	None
Attributes	None
Type	`xs:dateTime`
Restriction	This element is generated by Tamino. It is neither necessary to provide this element by yourself nor may you modify this information.

Example

See `tsd:adminInfo`.

58 tsd:default

Purpose	Using the `tsd:function` child element, `tsd:default` can specify a server extension function whose return value is used as a default value for an element or attribute.
Parent element	`tsd:logical`
Child elements	`tsd:function`
Attributes	None

Example

The following example shows how to use the `tsd:default` element:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="person">
        <tsd:collection name="person"></tsd:collection>
        <tsd:doctype name="person" />
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string">
          <xs:annotation>
            <xs:appinfo>
              <tsd:elementInfo>
                <tsd:logical>
                  <tsd:default>
                    <tsd:function xmlns:company="http://www.company.com/functions"
                                  name="company:createDefaultString"/>
```

```
                </tsd:default>
              </tsd:logical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="first" type="xs:string"></xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

59 tsd:delete

Caution: The `tsd:accessOptions` element and its child elements, including `tsd:delete`, are deprecated in Tamino 4.2.

Purpose	This empty element specifies whether the `delete` access option is valid for the doctype with which it is associated. If the `delete` access option is permitted, existing instances of the doctype may be deleted in the Tamino XML data store.
Parent element	`tsd:accessOptions`
Child elements	None
Attributes	None
Type	Empty

Example

See `tsd:accessOptions`.

60 tsd:dereference

Purpose	If present, this element specifies that dereferencing should be applied. The default is that dereferencing is not applied. If specified, for the query result the referenced document is additionally included into the current element, which becomes mixed content.
Parent element	`tsd:objectRef`
Child elements	None
Attributes	None
Type	Empty

Example

This example shows the use of the `tsd:dereference` element in the context of the `tsd:objectRef` element.

```
<tsd:objectRef>
  <tsd:collectionRef>encyclopedia</tsd:collectionRef>
  <tsd:dereference/>
  <tsd:accessPredicate operator = "=">
    <tsd:nodeRef>jazzMusician/@ID</tsd:nodeRef>
  </tsd:accessPredicate>
</tsd:objectRef>
```

61 tsd:doctype

Purpose	This element specifies the properties of a Tamino doctype. Multiple doctypes can be defined by a single schema.
Parent element	`tsd:schemaInfo`
Child elements	`tsd:logical`, `tsd:physical`, `tsd:nonXML`
Attributes	`name`

Name	Type	Description
`name`	`xs:QName`	Specifies the name of the corresponding doctype. The name of a doctype must be unique within a collection, not just within a schema. For an XML doctype, its name references a global element with same name. For a non-XML doctype, no global element with that name is allowed to exist.

Attributes

Example

This example demonstrates the use of the `tsd:doctype` element for the definition of a doctype named "Telephone" with closed content:

```
<tsd:schemaInfo name = "Telephone">
  <tsd:collection name = "Sample"></tsd:collection>
  <tsd:doctype name = "Telephone">
    <tsd:logical>
      <tsd:content>closed</tsd:content>
    </tsd:logical>
  </tsd:doctype>
</tsd:schemaInfo>
```

62 tsd:elementInfo

This element can be used in two different contexts:

1. In the context of xs:element/xs:annotation/xs:appinfo:

Purpose	This element contains information describing e.g. mapping, indexing or collation aspects of an element described by the *XML Schema* part of TSD. **Note:** A `tsd:elementInfo` element may only be inserted under an xs:element/xs:annotation/xs:appinfo chain if the `xs:element` element of which `tsd:elementInfo` is a successor has a `name` attribute and not a `ref` attribute.
Parent element	`xs:element/xs:annotation/xs:appinfo`
Child elements	`tsd:logical`, `tsd:physical`
Attributes	None

Example 1

This is a `tsd:elementInfo` element for native storage of an element using a standard index:

```
<xs:element name = "address">
  <xs:annotation>
     <xs:appinfo>
       <tsd:elementInfo>
         <tsd:logical>
           <tsd:collation>
             <tsd:language value="de"/>
             <tsd:strength value="primary"/>
             <tsd:caseFirst value = "upperFirst"/>
             <tsd:alternate value = "shifted"/>
             <tsd:caseLevel value = "false"/>
             <tsd:french value = "false"/>
             <tsd:normalization value ="false"/>
```

```
          </tsd:collation>
        </tsd:logical>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:standard/>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  .
  .
  .
</xs:element>
```

Example 2

This is a `tsd:elementInfo` element for mapping an element to an SQL table “p_table” from an ODBC data source “p_datasource” into a schema “p_schema” using the user ID “p_userid” and the password “p_password”:

```
<xs:element name = "destination">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:subTreeSQL schema = "p_schema" table = "p_table"
                            userid = "p_userid" password = "p_password"
                            datasource = "p_datasource">
            </tsd:subTreeSQL>
            <tsd:ignoreUpdate>
            </tsd:ignoreUpdate>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base = "xs:string">
        <xs:attribute ref = "xml:lang" fixed = "de"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

2. In the context of `tsd:schemaInfo`:

Purpose	The `tsd:elementInfo` element contains information describing e.g. mapping, indexing or collation aspects of an element described by the **XML Schema** part of TSD.
Parent element	`tsd:schemaInfo`
Child elements	`tsd:physical`
Attributes	`context`

Name	Type	Description
`context`	`xs:string`	This attribute contains the string with the context information for the element information. The `context` attribute is only available if the `tsd:elementInfo` is a child element of the `tsd:schemaInfo` **element**. The meaning of the value is essentially the same as for the `tsd:which` element, i.e. a string of the following syntax can be specified: `("/" element name)`

Attributes

63 tsd:field

This element specifies information about the fields of a key (in the usage scenario with `tsd:unique` as parent) or fields to be indexed within a compound index (in the usage scenario with `tsd:standard` as parent).

It can be used in these two contexts:

1. In the context of `tsd:unique`:

Purpose	In this context, the element specifies information about the fields of a key.
Parent element	`tsd:unique`
Child elements	None
Attributes	`xpath`

Name	Type	Description
`xpath`	`xs:string`	The `xpath` attribute specifies an XPath address uniquely identifying the field. The XPath expression is evaluated relative to the root element of the doctype being defined. Note that currently only paths consisting of "." or of the format `("/" element name)* "/" (element name | "@" attribute name)` are allowed, but no filters. For more information on XPath, see the ***XML Path Language (XPath) Version 1.0 specification*** published by the ***W3C***.

Attributes

Example 1

This example demonstrates the use of the `tsd:field` element as a child element of `tsd:unique`.

It contains the definition of one key using the “dot” option.

```
<tsd:doctype name="A">
  .
  .
  .
  <tsd:unique name="root-key">
    <tsd:field xpath="." />
  </tsd:unique>
  .
  .
  .
</tsd:doctype>
```

Example 2

Here is another example of the `tsd:field` element as a child element of `tsd:unique`:

```
<tsd:doctype name="A">
  .
  .
  .
  <tsd:unique name="CB-key">
    <tsd:field xpath="C" />
    <tsd:field xpath="B/@b" />
  </tsd:unique>
  <tsd:unique name="D-key">
    <tsd:field xpath="D" />
  </tsd:unique>
  .
  .
  .
</tsd:doctype>
```

It contains the definition of two keys:

a. **CB-key**
   containing the fields `C` and `B/@b`

b. **D-key**
   containing the field `D`

2. In the context of `tsd:standard`:

Purpose	This element specifies information about the fields to be indexed in a compound index.
Parent element	`tsd:standard`
Child elements	None
Attributes	`xpath`
Restriction	`tsd:field` may appear as a child of `tsd:standard` only in the scope of a `tsd:elementInfo` element, because a compound index can be defined only for an element, not for an attribute.

Name	Type	Description
`xpath`	`xs:string`	The `xpath` attribute defines a component contributing to the compound index. A `tsd:field` element with a corresponding `xpath` attribute must be defined for each XPath address to be included in the compound index. **Note:** The `tsd:field` element must appear at least twice (with different `xpath` attributes) for two or more items whose XPath addresses are stored in the `xpath` attribute to define a valid compound index. Valid `xpath` values are: ■ **A sequence of element names (optionally followed by a single attribute name), separated by slashes** Any XPath expression that uses only the axes "child" and "attribute" and does not have more than one attribute. ■ A single dot "." denotes the element for which the compound index is being defined. The XPath expression is evaluated relative to the element for which the compound index is being defined. **Important:** The `xpath` attribute is required and must point to an existing element or attribute.

Attributes

Example

In this example, two compound indexes are created in addition to a “normal” text index and a “normal” standard index:

```
<xs:element ...>
  ...
  <tsd:elementInfo>
    <tsd:physical>
      [<tsd:which>...</tsd:which>]
      <tsd:native>
        <tsd:index>
          <tsd:text/>
```

```
          <tsd:standard/>
          <tsd:standard>
            <tsd:field xpath="C"/>
            <tsd:field xpath="B/@b" />
          </tsd:standard >
          <tsd:standard>
            <tsd:field xpath="D"/>
            <tsd:field xpath="B/@b" />
          </tsd:standard >
        ...
        </tsd:index>
      </tsd:native>
    </tsd:physical>
  </tsd:elementInfo>
</xs:element>
```

These are:

- an index with entries for every combination of `C` and `B/@b` below the element;
- an index with entries for every combination of `D` and `B/@b` below the element.

64 tsd:french

Purpose	This element specifies French accent sorting order information.
Parent element	`tsd:collation`
Child elements	None
Attributes	`value`

Name	Type	Description
`value`	`xs:boolean`	Valid values for the `value` attribute are "true" and "false". ■ The value "true" enforces French accent sorting. ■ The default is "false", but the function is switched on if the collation language is “FR”. **Note:** This is the default root locale, i.e. the default collation attribute value according to the Unicode Collation Algorithm. However, this may be different if a specific language/locale is specified. For more information, refer to `tsd:collation` and to the collation documentation in the *XML Schema User Guide*.

Attributes

Example

See under `tsd:collation`.

65 tsd:function

Purpose	This element allows you to specify a server extension query function whose return value is used as the default value for an element or attribute in the schema. The function call is executed whenever a default value is needed, i.e. when validation encounters an optional attribute which is not present in the document but has such a default, or the analogous case for an element. However, these default values can be overwritten by the user. The function is only possible for simple elements with or without attributes. The function takes no parameters and returns a string value.
Parent element	`tsd:default`
Child elements	None
Attributes	None

Example

The following example shows the use of the `tsd:function` element:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="person">
        <tsd:collection name="person"></tsd:collection>
        <tsd:doctype name="person" />
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="person">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element name="name" type="xs:string">
          <xs:annotation>
            <xs:appinfo>
              <tsd:elementInfo>
                <tsd:logical>
                  <tsd:default>
                    <tsd:function xmlns:company="http://www.company.com/functions"
                                  name="company:createDefaultString"/>
                  </tsd:default>
                </tsd:logical>
              </tsd:elementInfo>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="first" type="xs:string"></xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

66 tsd:ignoreUpdate

Purpose	If this element is set, the contents of the externally-mapped node cannot be updated by loading instances using Tamino.
Parent element	`tsd:map`
Child elements	None
Attributes	None
Type	Empty
Restrictions	This element is only allowed for external mappings.

Example

See example 2 of `tsd:elementInfo`.

67 tsd:index

Purpose	This element allows you to specify whether an index (standard index, text index, compound index, reference index, or – for doctypes – computed index) should be created for the current schema element. It is only possible to define indexes for native storage, not for any kind of mapping.
Parent element	`tsd:native`
Child elements	`tsd:computed`, `tsd:reference`, `tsd:standard`, `tsd:text`
Attributes	None
Type	`xs:complexType`

Example 1

This example shows the `tsd:index` element required for the definition of a standard index:

```
<tsd:attributeInfo>
  <tsd:physical>
    <tsd:native>
      <tsd:index>
        <tsd:standard/>
      </tsd:index>
    </tsd:native>
  </tsd:physical>
</tsd:attributeInfo>
```

Example 2

Another example for the definition of a standard index:

```
<xs:element name = "firstname" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:logical>
           ...
        </tsd:logical>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:standard/>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

68 tsd:insert

Caution: The `tsd:accessOptions` element and its child elements, including `tsd:insert`, are deprecated in Tamino 4.2.

Purpose	The presence or absence of this element indicates whether the `insert` access option is valid for the associated doctype. If `insert` access is permitted, new instances of the doctype may be inserted in the Tamino XML data store.
Parent element	`tsd:accessOptions`
Child elements	None
Attributes	None
Type	Empty

Example

See `tsd:accessOptions`.

69 tsd:language

Purpose	This element specifies information about the language to be used. For example, give the `value` attribute the value "de" to specify the German language. For more information on internationalization aspects of the Tamino Schema Definition Language, refer to *Collations*.
Parent element	`tsd:collation`
Child elements	None
Attributes	`value`

Name	Type	Description
`value`	`xs:string`	See description below.

Attributes

Description of `value` attribute

The value of the `value` attribute is a string containing valid language and country codes. This string must be composed as explained in the section *Collations*:

A locale in the sense of ICU (International Components for Unicode) has up to three parts:

1. **ISO 639 Language Code**
   The first argument is a lower-case two-letter language code as defined by ISO 639. You can find a full list of these codes at **http://www.loc.gov/standards/iso639-2/iso639jac.html**.
2. **ISO 3166 Country Code**
   The second argument is a country code as defined by ISO 3166. These codes appear in two different forms:
   - upper-case two-letter codes (A2 code);
   - upper-case three-letter codes (A3 code).

You can find a full list of these codes at **http://www.iso.org/iso/country_codes.htm**.

3. **Suffix**
   A suffix can be specified for various purposes, for instance EURO for specifying a table containing a Euro currency symbol.

   Note: There is no difference in the sorting order whether EURO is specified or not.

In ICU, a locale is represented by one string that consists of a mandatory ISO 639 language code plus an optional ISO 3166 country code plus an optional suffix. Tamino allows two different styles for delimiting the components:

- **ICU style**
  The separator is the underscore sign. For example, German for Germany with a Euro sign is expressed as `de_DE_EURO`.
- **RFC-1766 style**
  The separator is the minus sign, for example: `de-DE-EURO`.

Note: Locale names are case-insensitive. Usually the language part of the locale is lowercase, the country is uppercase and the variant part is uppercase.

For a list of available code strings refer to the section *Language and Country Codes* in the appendix.

Example 1

The following example illustrates the use of the `tsd:language` element within an element definition to define an element for an address that is handled as English.

```
<xs:element name = "address">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:logical>
          <tsd:collation>
            <tsd:language value="en"/>
            <tsd:strength value="primary"/>
            <tsd:caseFirst value = "upperFirst"/>
            <tsd:alternate value = "shifted"/>
            <tsd:caseLevel value = "false"/>
            <tsd:french value = "false"/>
            <tsd:normalization value ="false"/>
          </tsd:collation>
        </tsd:logical>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:standard/>
```

```
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base = "xs:string">
      <xs:attribute ref = "xml:lang" fixed = "en"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Example 2

The same for American English (country = USA). Simply change the line:

```
<tsd:language value="en">
```

in the above example to the following:

```
<tsd:language value="en_US">
```

Example 3

The same for German with phonebook-like sorting order. Simply change the line:

```
<tsd:language value="en">
```

in the above example to the following:

```
<tsd:language value="de__PHONEBOOK">
```

70 tsd:logical

This element can be used in different contexts:

1. In the context of `tsd:elementInfo` or `tsd:attributeInfo`:

Purpose	In the context of `tsd:elementInfo` or `tsd:attributeInfo`, this element is used for the following purposes: ■ To represent the logical subtree of the corresponding element; ■ To define default settings for the child elements of the `tsd:collation` element; ■ To define functions for generating default values for the child elements using the `tsd:default` element; ■ To define action triggers; for this purpose see `tsd:trigger`. This element is optional.
Parent element	`tsd:elementInfo` or `tsd:attributeInfo`
Child elements	`tsd:collation`, `tsd:default`, `tsd:trigger`
Attributes	None
Type	`xs:complexType`

2. In the context of `tsd:shadowXML`:

Purpose	In the context of `tsd:shadowXML`, this element can be used to specify: ■ **Open or closed content** For details, see the section *Open Content vs. Closed Content Validation* of the *Tamino XML Schema User Guide*.

	■ **Unique key definitions** For details, see the section *Definition of Unique Keys* of the *Tamino XML Schema User Guide*. This element is optional.
Parent element	`tsd:shadowXML`
Child elements	`tsd:content`, `tsd:unique`
Attributes	None
Type	`xs:complexType`

Example

The following example illustrates the use of the `tsd:logical` element in the context of `tsd:shadowXML`. This example defines the Server Extensions for the shadow functions and specifies that the schema should be treated as open content:

```
<tsd:shadowXML>
  <tsd:onBinaryInsert>SXSBinaryIndexer.put</tsd:onBinaryInsert>
  <tsd:onTextInsert>SXSTextIndexer.put</tsd:onTextInsert>
  <tsd:logical>
    <tsd:content>open</tsd:content>
  </tsd:logical>
</tsd:shadowXML>
```

3. In the context of `tsd:doctype`:

Caution: This usage of `tsd:logical` is deprecated in Tamino 4.2.

Purpose	In the context of `tsd:doctype`, this element can be used to specify: ■ **Access options** For details, see the section *Access options* of the *Tamino XML Schema User Guide*. ■ **Open or closed content** For details, see the section *Open Content vs. Closed Content Validation* of the *Tamino XML Schema User Guide*. ■ **Unique key definitions** For details, see the section *Definition of Unique Keys* of the *Tamino XML Schema User Guide*. This element is optional.
Parent element	`tsd:doctype`
Child elements	`tsd:content`, `tsd:accessOptions`, `tsd:unique`
Attributes	None
Type	`xs:complexType`

71 tsd:map

Purpose	This element contains mapping information, i.e. physical storage information, for use in conjunction with Tamino X-Node. The `tsd:subTreeAdabas`, `tsd:subTreeAdabasPE` and `tsd:nodeAdabasField` child elements provide support for the mapping of Adabas files and fields; the `tsd:subTreeSQL`, and `tsd:nodeSQL` child elements similarly provide support for the mapping of SQL tables and columns via the ODBC interface; and the `tsd:ignoreUpdate` child element sets read-only or read-write permissions on elements. The `tsd:pure` child elements allows you to define a pure mapping instead of the default hybrid mapping.
Parent element	`tsd:physical`
Child elements	`tsd:subTreeAdabas`, `tsd:subTreeAdabasPE`, `tsd:nodeAdabasField`, `tsd:subTreeSQL`, `tsd:nodeSQL`, `tsd:xTension`, `tsd:ignoreUpdate`, `tsd:pure`
Attributes	None
Restrictions	`tsd:subTreeAdabas`, `tsd:subTreeAdabasPE` and `tsd:subTreeSQL` are only valid if the `tsd:map` element is a descendant element of a `tsd:elementInfo` element.

Example

This example shows a `tsd:map` element for mapping data to a table “p_table” within an SQL database that is accessible via ODBC as ODBC datasource “p_datasource” with the user ID “p_userid” and the password “p_password”:

```
<tsd:map>
  <tsd:subTreeSQL schema = "p_schema"
                  table = "p_table"
                  userid = "p_userid"
                  password = "p_password"
                  datasource = "p_datasource">
  </tsd:subTreeSQL>
  <tsd:ignoreUpdate/>
</tsd:map>
```

72 tsd:modified

Purpose	This element contains the date of the last modification of the schema to which `tsd:adminInfo` belongs.
Parent element	`tsd:adminInfo`
Child elements	None
Attributes	None
Type	`xs:dateTime`
Restrictions	This element is generated automatically by Tamino.

Example

See `tsd:adminInfo`.

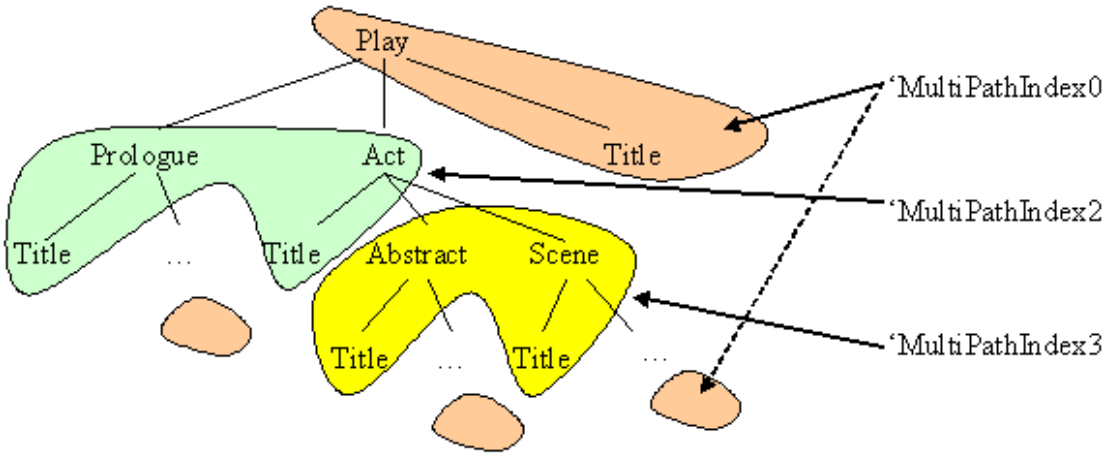
73 tsd:multiPath

Purpose	This element specifies information on multipath indexes, which enable efficient indexing even on recursive structures. Contrary to the earlier versions of Tamino, where each index reflected an absolute XPath address (meaning for queries containing wildcard expressions that all matching indexes had to be analyzed with respect to the given predicate in order to receive a complete index), Tamino can now define an index whose XPath address matches special criteria. This index is called a multipath index. The idea behind a multipath index is that indexes are identified by a unique label. All nodes (i.e. elements and attributes) having the same label in their index definition address the same index. The label is specified in the contents of the `tsd:multiPath` element (see the example below). The type of the label is `xs:NMTOKEN`. A multipath index can be defined either as a text index or as a standard index, depending on which is the parent element within the schema. It also allows indexing of recursive structures that could not be indexed in earlier versions of Tamino.
Parent element	`tsd:text`, `tsd:standard`
Child elements	None
Attributes	None
Type	`xs:NMTOKEN`
Restrictions	The multipath index can only be used when the structure index is set to "condensed" or "full". Identical datatypes and collations must be defined for the nodes that are grouped into one index.

Example

The following example shows an excerpt for the global element `Title` and two local element definitions for `/Play/Act/Abstract/Title` and `/Play/Prologue/Title`. The index definition is extended with the `<tsd:multiPath>` attribute. The definition above results in three multiPath indexes:

- an index for `/Play/*/Title`, labelled as “MultiPathIndex2”, containing data from the global title element restricted by the `<tsd:which>/Play/Act/Title` and data from the local element in `/Play/Prologue`
- an index for `/Play/Act/*/Title`, labelled as “MultiPathIndex3”, containing data from the global title element restricted by the `<tsd:which>/Play/Act/Scene/Title` and data from the local element in `/Play/Act/Abstract`
- an index for all other `Title` elements, labelled as “MultiPathIndex0”



The picture shows which of these three indexes works with which parts of the structure tree. In TSD, the index definition looks like this:

```
<!-- global Element Definition for Title -->
<xs:element name="Title" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <!-- default which for global Element Definition Title -->
          <tsd:native>
            <tsd:index>
              <tsd:text>
                <tsd:multiPath>MultiPathIndex0</tsd:multiPath>
              </tsd:text>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
        <tsd:physical>
          <tsd:which>/Play/Act/Title</tsd:which>
          <tsd:native>
            <tsd:index>
              <tsd:text>
                <tsd:multiPath>MultiPathIndex2</tsd:multiPath>
```

```
              </tsd:text>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
        <tsd:physical>
          <tsd:which>/Play/Act/Scene/Title</tsd:which>
          <tsd:native>
            <tsd:index>
              <tsd:text>
                <tsd:multiPath>MultiPathIndex3</tsd:multiPath>
              </tsd:text>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
.
.
.
<!-- local Element Definition for /Play/Prologue /Title -->
<xs:element name="Title" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:text>
                <tsd:multiPath>MultiPathIndex2</tsd:multiPath>
              </tsd:text>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
.
.
.
<!-- local Element Definition for /Play/Act/Abstract/Title -->
<xs:element name="Title" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:text>
                <tsd:multiPath>MultiPathIndex3</tsd:multiPath>
```

```
              </tsd:text>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

74 tsd:multiple

Purpose	This element must be specified if the Adabas field that is mapped to the parent element is a MU field.
Parent element	`tsd:nodeAdabasField`
Child elements	None
Attributes	None
Type	Empty

Example

The following example demonstrates the use of the `tsd:multiple` element of TSD:

```
<tsd:elementInfo>
  <tsd:physical>
    <tsd:map>
      <tsd:nodeAdabasField shortname = "mn" format = "A">
        <tsd:multiple/>
      </tsd:nodeAdabasField>
      <tsd:ignoreUpdate/>
    </tsd:map>
  </tsd:physical>
</tsd:elementInfo>
```

75 tsd:name

Purpose	This element specifies the name of a function that delivers the default value for an element or an attribute. The content of the `tsd:name` element specifies the name of the function.
Parent element	`tsd:function`
Child elements	None
Attributes	The `tsd:name` element contains an attribute belonging to a different namespace than itself, namely the `xmlns:` namespace. This attribute itself contains the namespace declaration. See the example in `tsd:function`.

Example

See the example in `tsd:function`.

76 tsd:native

Purpose	This element contains physical information on indexing and object referencing, i.e. linking of different elements.
Parent element	`tsd:physical`
Child elements	`tsd:index`; `tsd:objectRef`
Attributes	None

Example 1

This example demonstrates the use of `tsd:native` to define a standard index:

```
<tsd:attribute name="width">
  <tsd:attributeInfo>
    <tsd:physical>
      <tsd:native>
        <tsd:index>
          <tsd:standard/>
        </tsd:index>
      </tsd:native>
    </tsd:physical>
  </tsd:attributeInfo>
</tsd:attribute>
```

Example 2

This is an example of native storage using object references:

```
<xs:element name="address">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:logical>
          .
          .
          .
        </tsd:logical>
        <tsd:physical>
          <tsd:native>
            <tsd:objectRef>
              <tsd:collectionRef>Customers</tsd:collectionRef>
              <tsd:accessPredicate operator="=">
                <tsd:nodeRef>/Customer/CustomerNo</tsd:nodeRef>
              </tsd:accessPredicate>
            </tsd:objectRef>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  .
  .
  .
</xs:element>
```

77 tsd:noConversion

Purpose	If this element is specified, non-XML text data is stored "as-is", i.e. it is not converted to Unicode. The data can be retrieved from Tamino exactly as it was stored, without risk of character set conversions.
Parent element	`tsd:nonXML`
Child elements	None
Attributes	None

Normally, Tamino stores non-XML text data in Unicode. A query can specify the desired character set encoding when retrieving data; however, even if the same character set is specified when storing data and when retrieving data, information may be lost as a result of the two-fold data conversion. If this element is specified, no character set conversion is performed.

Notes:

1. Even if the Accept-Charset header field is set in a query, it is ignored. The data is returned exactly as it was stored.
2. If a character encoding was specified in the Content-type HTTP header when the data was stored, this encoding is returned in the header of the HTTP response.

Tamino does not generate indexes for doctypes for which this element is specified. If you require indexes for these doctypes, use the Tamino Non-XML Indexer.

The `noConversion` option is set by default when non-XML data is stored via the WebDAV interface.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="UnconvertedTextSchema">
        <tsd:collection name="MyColl" />
        <tsd:doctype name="UnconvertedText">
          <tsd:nonXML>
            <tsd:noConversion />
          </tsd:nonXML>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

78 tsd:nodeAdabasField

Purpose	If the subtree is mapped to Adabas, the corresponding child nodes must contain `tsd:nodeAdabasField`. `tsd:nodeAdabasField` can also be used with `tsd:subTreeAdabasPE`, see there.
Parent element	`tsd:map`
Child elements	`tsd:multiple`
Attributes	`shortname`, `format`, `length`

Name	Type	Description
`shortname`	`xs:string`	This attribute must contain the short name of the Adabas field.
`length`	`xs:string`	This optional attribute specifies the length of the Adabas field being mapped.
`format`	`xs:string`	This attribute specifies the format of the field.

Attributes

Example

See the `tsd:subTreeAdabas` element.

79 tsd:nodeParameter

Purpose	This element specifies SQL column names to be used in SQL "where" clauses within access predicate definitions. It is of type `xs:string`. It may occur multiply.
Parent element	`tsd:accessPredicate`
Child elements	None
Attributes	None

Example

See the `tsd:subTreeSQL` element.

80 tsd:nodeRef

Purpose	This element allows you to specify a path to the referenced node within an access predicate definition. The doctype in the node reference must exist. For cyclic references, this means that one of the references must be defined via an “update schema”. Deletion of the referenced nodes leads to the deletion of the object reference.
Parent element	`tsd:accessPredicate`
Child elements	None
Attributes	None
Type	`xs:string`

Example

This fragment illustrates the use of the `tsd:nodeRef` element of TSD:

```
<xs:element name = "address">
<xs:annotation>
  <xs:appinfo>
    <tsd:elementInfo>
      <tsd:logical>
      .
      .
      .
      </tsd:logical>
      <tsd:physical>
        <tsd:native>
          <tsd:objectRef>
            <tsd:collectionRef>Customers</tsd:collectionRef>
            <tsd:accessPredicate operator = "=">
              <tsd:nodeRef>/Customer/CustomerNo</tsd:nodeRef>
            </tsd:accessPredicate>
          </tsd:objectRef>
```

```
        </tsd:native>
      </tsd:physical>
    </tsd:elementInfo
  </xs:appinfo>
</xs:annotation>
.
.
.
</xs:element>
```

81 tsd:nodeSQL

Purpose	This element carries the mapping information for a node in the XML sub-tree that is mapped to a single SQL column. To be used to specify mapping for logical child nodes of a logical node mapped via `tsd:subTreeSQL`.
Parent element	`tsd:map`
Child elements	None
Attributes	`column`

Name	Type	Description
`column`	`xs:string`	This attribute specifies the column in an SQL database to which an element or attribute in the Tamino database is mapped.

Attributes

Example

The following code fragment of a Tamino schema definition shows the code necessary to define a mapping of an element to an SQL column “Last Name” via X-Node:

```
<xs:element name = "lastname" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:logical>
          .
          .
          .
        </tsd:logical>
        <tsd:physical>
          <tsd:map>
            <tsd:nodeSQL column = "Last Name"></tsd:nodeSQL>
          </tsd:map>
```

```
        </tsd:physical>
      </tsd:elementInfo
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

82 tsd:nonXML

Purpose	This element indicates a non-XML doctype. Such a doctype may be indexed. Also, a special shadow document for increasing search speed may be created using `tsd:shadowXML`. Access options can be defined for a non-XML doctype similarly as for a native doctype using the `tsd:accessOptions` child element. See the documentation of the respective child elements for more information. Also see the section *Storing Non-XML Objects in Tamino* of the *Tamino XML Schema User Guide*.
Parent element	`tsd:doctype`
Child elements	`tsd:accessOptions`, `tsd:index`, `tsd:noConversion`, `tsd:shadowXML` **Note:** The `tsd:accessOptions` and `tsd:index` child elements of `tsd:nonXML` are deprecated.
Attributes	None

Example

The following `tsd:schemaInfo` element defines a doctype `BasicNonXml` as a non-XML doctype using the `tsd:nonXML` element:

```
<tsd:schemaInfo name="BasicNonXml">
  <tsd:collection tsd:name="myBasictestCollection">
    <tsd:doctype tsd:name="BasicNonXml">
      <tsd:nonXML/>
    </tsd:doctype>
  </tsd:collection>
</tsd:schemaInfo>
```

83 tsd:normalization

Purpose	This element specifies whether text is to be treated as normalized.
Parent element	`tsd:collation`
Child elements	None
Attributes	`value`

Name	Type	Description
`value`	`xs:boolean`	Valid values for the `value` attribute are "true" and "false". The value "true" produces results as if text was normalized. The default is "false" (no normalization) for most languages. **Note:** This is the default root locale, i.e. the default collation attribute value according to the Unicode Collation Algorithm. However, this may be different if a specific language/locale is specified. For more information on this topic, refer to `tsd:collation` and to the collation documentation in the *XML Schema User Guide*.

Attributes

Example

See under `tsd:collation`.

84 tsd:objectRef

Status

Purpose	This element enables the creation of object references that are used for joining doctypes (similarly to a "JOIN" operation in SQL).
Parent element	`tsd:native`
Child elements	`tsd:collectionRef`, `tsd:dereference`, `tsd:accessPredicate`
Attributes	None

Example

This example illustrates the use of the `tsd:objectRef` element:

```
<xs:element name = "address">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:objectRef>
              <tsd:collectionRef>Customers</tsd:collectionRef>
              <tsd:accessPredicate operator = "=">
                <tsd:nodeRef>/Customer/CustomerNo</tsd:nodeRef>
              </tsd:accessPredicate>
            </tsd:objectRef>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  .
  .
```

```
   .
</xs:element>
```

The `tsd:objectRef` element is only interpreted by the *Tamino X-Query* processor. In *Tamino XQuery* the user is able to specify the join within the query by himself.

85 tsd:onBinaryInsert

Purpose	This element specifies information about shadow functions for binary non-XML data. It contains the name of the *Server Extension* that caters for binary data when creating the content of the shadow document. For more information, see the section *Using Shadow Functions* of the *Tamino XML Schema User Guide*.
Parent element	`tsd:shadowXML`
Child elements	None
Attributes	None

Example

See `tsd:shadowXML`.

86 tsd:onDelete

This element can be used in different contexts:

1. In the context of `tsd:xTension`

Purpose	As a child of `tsd:xTension`, this element specifies for an XML sub-tree mapped to a Tamino Server Extension the function or method that is to be invoked in the SXS if a node is to be deleted.
Parent element	`tsd:xTension`
Child elements	None
Attributes	None
Type	`xs:string`

2. In the context of `tsd:trigger`

Purpose	As a child of `tsd:trigger`, the element can be used to specify additional information for the definition of action triggers.
Parent element	`tsd:trigger`
Child elements	`tsd:parameters`
Attributes	`name`, `type`
Type	`xs:string`

Name	Type	Description
`type`	`xs:NMTOKEN`	The `type` attribute of the `tsd:onDelete` element can only have the value "action".
`name`	`xs:NCName`	Can be used, instead of the text value inside the `tsd:onDelete` element, to specify the name of the server extension function.

Attributes

Examples

See `tsd:xTension` and `tsd:trigger`.

87 tsd:onInsert

Purpose	This element specifies information on action triggers.
Parent element	`tsd:trigger`
Child elements	`tsd:parameters`
Attributes	`name`, `type`

Name	Type	Description
`type`	`xs:NMTOKEN`	The `type` attribute of the `tsd:onInsert` element can only have the value "action".
`name`	`xs:NCName`	Can be used, instead of the text value inside the `tsd:onInsert` element, to specify the name of the server extension function.

Attributes

Example

See `tsd:trigger`.

88 tsd:onProcess

Purpose	This element specifies for an XML sub-tree mapped to a Tamino Server Extension the function or method that is to be invoked in the SXS if a node is to be processed.
Parent element	`tsd:xTension`
Child elements	None
Attributes	None
Type	`xs:string`

Example

See `tsd:xTension`

89 tsd:onTextInsert

Purpose	This element specifies information on shadow functions for non-XML text data. It contains the name of the Tamino*Server Extension* that caters for text data when creating the content of the shadow document.
Parent element	`tsd:shadowXML`
Child elements	None
Attributes	None
Type	`xs:string`

Example

See `tsd:shadowXML`.

90 tsd:onUpdate

This element can be used in different contexts:

1. In the context of `tsd:xTension`

Purpose	As a child of `tsd:xTension`, this element specifies information on keys.
Parent element	`tsd:xTension`
Child elements	None
Attributes	`value`
Type	`xs:string`

Attributes

Name	Type	Description
`value`	`xs:boolean`	Either "TRUE" or "FALSE".

2. In the context of `tsd:trigger`

Purpose	As a child of `tsd:trigger`, the element can be used to specify additional information for the definition of action triggers.
Parent element	`tsd:trigger`
Child elements	`tsd:parameters`
Attributes	`name`, `type`
Type	`xs:string`

Name	Type	Description
type	xs:NMTOKEN	The type attribute of the tsd:onUpdate element can only have the value "action".
name	xs:NCName	Can be used, instead of the text value inside the tsd:onUpdate element, to specify the name of the server extension function.

Attributes

Example

See tsd:trigger.

91 tsd:parameter

Purpose	Specifies a single additional parameter for an action trigger.
Parent element	`tsd:parameters`
Child elements	None
Attributes	`name`, `type`
Type	Determined by the value of the `type` attribute.

Name	Type	Description
`type`	`xs:QName`	The simple type of the additional parameter to be passed to the trigger server extension function.
`name`	`xs:NCName`	

Attributes

Example

See `tsd:parameters`.

92 tsd:parameters

Purpose	Specifies additional parameters for an action trigger.
Parent element	`tsd:onDelete`, `tsd:onInsert`, `tsd:onUpdate`
Child elements	`tsd:parameter`
Attributes	None

Example

```
.
.
.
<tsd:trigger>
  <tsd:onInsert type="action" name="myXtension.onInsert">
    <tsd:parameters>
      <tsd:parameter name="param1" type="xs:string">
        MyClassName
      </tsd:parameter>
      <tsd:parameter name="param2" type="xs:int">
        4711
      </tsd:parameter>
    </tsd:parameters>
  </tsd:onInsert>
</tsd:trigger>
.
.
.
```

93 tsd:physical

This element can be used in different contexts:

1. In the context of `tsd:elementInfo` or `tsd:attributeInfo`:

Purpose	In the context of `tsd:elementInfo` or `tsd:attributeInfo`, this element can be used to specify information on physical data storage, i.e. mainly indexing and mapping.
Parent element	`tsd:elementInfo` or `tsd:attributeInfo`
Child elements	`tsd:which`, `tsd:native`, `tsd:map`
Attributes	None

Example

The following example shows the use of the `tsd:physical` element in TSD:

```
<tsd:physical>
  <tsd:native>
    <tsd:objectRef>
      <tsd:collectionRef>Customers</tsd:collectionRef>
      <tsd:accessPredicate operator = "=">
        <tsd:nodeRef>/Customer/CustomerNo</tsd:nodeRef>
      </tsd:accessPredicate>
    </tsd:objectRef>
  </tsd:native>
</tsd:physical>
```

2. In the context of `tsd:shadowXML`:

Purpose	This element controls the physical storage of the corresponding doctype.
Parent element	`tsd:shadowXML`
Child elements	`tsd:map`, `tsd:structureIndex`, `tsd:compress`
Attributes	None

3. In the context of `tsd:doctype`:

Purpose	This element controls the storage of the corresponding doctype. In this usage scenario it must not be specified when `tsd:nonXML` is used, because these elements are mutually exclusive.
Parent element	`tsd:doctype`
Child elements	`tsd:structureIndex`, `tsd:compress`
Attributes	None

94 tsd:primaryKeyColumn

Purpose	This element specifies the SQL columns that make up the primary key, if the sub-tree to which it belongs is mapped to SQL.
Parent element	`tsd:subTreeSQL`
Child elements	None
Attributes	None
Type	`xs:string`

Example

See `tsd:subTreeSQL`.

95 tsd:pure

Purpose	This element specifies information for pure Adabas or SQL mapping. The basic idea is that data exist only in the Adabas file or SQL table, and Tamino only acts as an XML wrapper. In particular, when such documents are stored through Tamino, the contents are mapped to Adabas or SQL, and then Tamino forgets the document. In other words, hybrid mapping is not possible where parts of a document are mapped to (i.e. stored in) Tamino and parts are mapped to X-Node. The effect of this option is as follows: ■ No data is stored in Tamino for that doctype. ■ In particular, inserting new documents does not assign an ino:id, and any markup (comments, processing instructions, insignificant whitespace) is lost. ■ Even though a document was inserted through Tamino, it cannot be updated using _process, and it cannot be deleted using _delete (both operations are only possible through _xquery update). ■ `_xquery` update does not store anything in Tamino, any modifications to the markup (e.g. comments) are lost. ■ It is particularly important to note that in such a doctype only one Adabas or SQL file can be mapped. ■ Any real content in the document must be mapped to Adabas or SQL. ■ You cannot use a full structure index in conjunction with pure mapping to Adabas or SQL.
Parent element	`tsd:map`
Child elements	None
Attributes	None

Example

96 tsd:read

Caution: The `tsd:accessOptions` element and its child elements, including `tsd:read`, are deprecated in Tamino 4.2.

Purpose	The presence or absence of this element indicates whether read access is permitted for the associated doctype. If read access is permitted, instances of the doctype may be read from the Tamino XML data store.
Parent element	`tsd:accessOptions`
Child elements	None
Attributes	None

Example

See `tsd:accessOptions`.

97 tsd:reference

Purpose	This element specifies information on reference nodes. See the section *The Reference Index - Indexing with Respect to Sub-Trees of a Document* of the *Tamino XML Schema User Guide*.
Parent element	`tsd:index`
Child elements	`tsd:refers`
Attributes	None
Restrictions	The following rules apply for this element: ■ The `tsd:reference` element may not be specified below the `tsd:index` element of the root element of a doctype. `tsd:reference` can only be used for elements, not for attributes. It must not be applied for recursive elements.

98 tsd:refers

This element can be used in the following different contexts:

1. For reference definition, the `tsd:refers` element is used in the context of `tsd:standard` or `tsd:text` as described in the following table:

Purpose	This element defines references for use in connection with reference indexes. The content of the `tsd:refers` element specifies the node that is to be used as a base for a reference. For more information on this topic, see also `tsd:reference` and `tsd:which`.
Parent element	`tsd:text`, `tsd:standard`
Child elements	None
Attributes	None
Restrictions	The following rules apply for this element: ■ The element that describes the index (in the contents of the accompanying `tsd:which` element) must have a parent that matches the XPath expression of the `<tsd:refers>` value. Otherwise an error message is issued. In the example below, an additional `<tsd:which>/Doc/A/B/X/C</tsd:which>` would also match, but a `<tsd:which>/Doc/A/X/C</tsd:which>` would cause an error at define time. ■ Only absolute path expressions that do not contain any wildcards are supported in this context. ■ The option for the `tsd:structureIndex` element may not be "none". As long as these conditions are fulfilled, recursive structures can also be indexed.
Type	`xs:string`

Example

This example shows how to use the `tsd:refers` element for index definition: The index refers to */Doc/A/B*.

```
<xs:element name = "C" type = "xs:string">
...
   <tsd:elementInfo>
     <tsd:physical>
       <tsd:which>/Doc/A/B/C</tsd:which>
       <tsd:native>
         <tsd:index>
           <tsd:text>
             <tsd:refers>/Doc/A/B<tsd:refers>
           </tsd:text>
         </tsd:index>
       </tsd:native>
     </tsd:physical>
...
</xs:element>
```

2. For reference index definition, the `tsd:refers` element is used in the context of `tsd:reference` as described in the following table:

Purpose	In the context of reference indexes, one reference node may refer to another reference node. The current reference node uses the `tsd:refers` element to specify the reference node. For more information on this topic, see also `tsd:reference` and `tsd:which`.
Parent element	`tsd:reference`
Child elements	None
Attributes	None
Restrictions	The following rules apply for the contents of this element: ■ The element that describes the index (in the contents of the accompanying `tsd:which` element) must have a parent that matches the XPath expression of the `<tsd:refers>` value. Otherwise an error message is issued. In the example below, an additional `<tsd:which>/Doc/A/B/X/C</tsd:which>` would also match, but a `<tsd:which>/Doc/A/X/C</tsd:which>` would cause an error at define time. ■ Only absolute path expressions that do not contain any wildcards are supported in this context.
Type	`xs:string`

Example

The following excerpt from a schema describes a reference index definition for the node B, which should refer to the ancestor `/Doc/A`:

```
<xs:element name = "B" maxOccurs = "unbounded">
  .
  .
  .
  <tsd:elementInfo>
    <tsd:physical>
      <tsd:native>
        <tsd:index>
          <tsd:reference>
            <tsd:refers>/Doc/A<tsd:refers>
          </tsd:reference>
          .
          .
          .
        </tsd:index>
      </tsd:native>
    </tsd:physical>
  </tsd:elementInfo>
  .
  .
  .
</xs:element>
```

99 tsd:schema

Purpose	This element specifies whether a schema is necessary for storing data in a collection. This is decided depending on the contents of the `use` attribute (see below).
Parent element	`tsd:collection`
Child elements	None
Attributes	`use`

Name	Type	Description
`use`	`xs:NMTOKEN`	The valid values for the `use` attribute are: ■ **"required"** A schema is required for storing data in the collection specified by the parent `tsd:collection` element. Data cannot be stored in the collection without having previously defined a schema. **Note:** This is the default value. It corresponds to the behavior of collections in Tamino versions prior to version 4.2. ■ **"optional"** A schema may be used, but is not required for storing data in the collection specified by the parent `tsd:collection` element. Both explicit and implicit creation of doctypes in the collection are permitted. ■ **"prohibited"** A schema may not be used for storing data in the collection specified by the parent `tsd:collection` element. When storing data, a suitable internal schema is created automatically.

Attributes

100 tsd:schemaInfo

Purpose	This element contains schema relevant information.
Parent element	`.../xs:annotation/xs:appinfo`
Child elements	`tsd:collection`, `tsd:doctype`, `tsd:elementInfo`, `tsd:attributeInfo`, `tsd:adminInfo`
Attributes	`name`
Restrictions	`tsd:schemaInfo` is currently required for a schema to be defined in Tamino. Thus, it is currently not possible to define an arbitrary XML schema downloaded from the Internet as a Tamino schema.

Name	Type	Description
`name`	`xs:NMTOKEN`	This value of this attribute specifies the name of the schema that is described by the whole `tsd:schemaInfo` element and its associated subtree. The name of a schema must be unique in the scope of the collection in which the schema is defined.

Attributes

Example

This example shows how to specify information for the “Telephone” schema. It is defined with closed contents, and the collection name and doctype name are assigned to the schema with this definition:

```
<xs:element name = "address">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name = "Telephone">
        <tsd:collection name = "Sample"/>
        <tsd:doctype name = "Telephone">
          <tsd:logical>
```

```
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
      <tsd:elementInfo>
        <tsd:logical>
          <tsd:collation>
            <tsd:language value="de"/>
            <tsd:strength value="primary"/>
            <tsd:caseFirst value = "upperFirst"/>
            <tsd:alternate value = "shifted"/>
            <tsd:caseLevel value = "false"/>
            <tsd:french value = "false"/>
            <tsd:normalization value ="false"/>
          </tsd:collation>
        </tsd:logical>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:standard/>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base = "xs:string">
        <xs:attribute ref = "xml:lang" fixed = "de">
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

101 tsd:server

Purpose	This element specifies the version number of the Tamino server.
Parent element	`tsd:adminInfo`
Child elements	None
Attributes	None
Restriction	This element is generated by Tamino. It is neither necessary to provide this element by yourself nor may you modify this information.

Example

102 tsd:shadowXML

Purpose	This element establishes a connection between a schema and a shadow function in order to improve the integration of non-XML data in Tamino. A shadow document contains an XML representation which is generated from the non-XML data by the shadow function. Therefore a shadow document can be accessed using X-Query or XQuery, just as XML data are accessed. This technique allows efficient indexing of non-XML data. A shadow function is just a special kind of *Server Extension*. Depending on whether it caters for text data or binary data, this shadow function can be defined using the `tsd:onBinaryInsert` or `tsd:onTextInsert` child elements. For more information on shadow functions, see: ■ the *Server Extension* documentation; ■ the *Shadow Functions* section of the XML Schema Language User Guide.
Parent element	`tsd:nonXML`
Child elements	`tsd:onBinaryInsert`, `tsd:onTextInsert`, `tsd:storeShadowOnly`
Attributes	None
Restrictions	At least one of the child elements `tsd:onBinaryInsert` or `tsd:onTextInsert` must be present.

Example

This example illustrates the definition of an index for non-XML data using a shadow document that is maintained by the Server Extensions *SXSBinaryIndexer.put* and *SXSTextIndexer.put* in TSD:

```
<?xml version="1.0" encoding="windows-1252"?>
<xs:schema elementFormDefault="qualified"
          xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="nox">
        <tsd:collection name="nox" />
        <tsd:doctype name="nox">
          <tsd:nonXML>
            <tsd:shadowXML>
              <tsd:onBinaryInsert>SXSBinaryIndexer.put</tsd:onBinaryInsert>
              <tsd:onTextInsert>SXSTextIndexer.put</tsd:onTextInsert>
            </tsd:shadowXML>
          </tsd:nonXML>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="blob">
    .
    .
    .
  </xs:element>
</xs:schema>
```

103 tsd:standard

Purpose	This element specifies that a standard index should be generated for the current schema element. Depending on whether a child element is present, an advanced index (multi-path or compound index or a combination of these) or a simple index is created.
Parent element	`tsd:index`
Child elements	`tsd:refers`, `tsd:multiPath`, `tsd:field` **Caution:** The order of the `tsd:field` elements determines the order of the values that comprise the index entry.
Attributes	None
Type	Empty

Example

This example illustrates the definition of a standard index in TSD:

```
<xs:element name = "lastname" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:standard/>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

See also `tsd:index` and `tsd:field`.

104 tsd:storeShadowOnly

Purpose	This element specifies that only the shadow document is to be stored in Tamino but not the original document that the shadow document represents. Instead, a BLOB of 0 bytes is stored in conjunction with the original content type. Queries that use plain URL addressing to attempt to retrieve the original document will receive an HTTP status response 404 (not found); XQuery queries that use the serialization method `ino:document` will also receive response 404. The original document can be stored in an external system (for example, a file system) under a name which is then used as the document name (`ino:docname`) when storing the shadow XML document in Tamino. The use of `tsd:storeShadowOnly` allows the original document to be indexed in Tamino although it is located outside Tamino. The `tsd:storeShadowOnly` element cannot be added or removed during a schema update.
Parent element	`tsd:shadowXML`
Child elements	None
Attributes	None
Type	Empty

Example

This example shows how to define that only the shadow document should be stored, and not the original document itself:

```
<tsd:doctype name="...">
  ... logical and physical properties ...
  <tsd:nonXML>
    <tsd:shadowXML>
      <tsd:storeShadowOnly/>
      <tsd:onBinaryInsert>...</tsd:onBinaryInsert>
      <tsd:onTextInsert>...</tsd:onTextInsert>
    </tsd:shadowXML>
  </tsd:nonXML>
</tsd:doctype>
```

105 tsd:strength

Purpose	This element specifies strength information, i.e. the level of comparison to apply for collation.
Parent element	`tsd:collation`
Child elements	None
Attributes	`value`

Name	Type	Description
`value`	`xs:NMTOKEN`	Valid values for the `value` attribute are: ■ **"primary"** Level 1: base character comparison, e.g., “a” < “b”. ■ **"secondary"** Level 2: accents on characters are compared, e.g. “as” < “às” < “at”. ■ **"tertiary"** Level 3 (default): uppercase and lowercase characters are compared, e.g. “ao” < “Ao” < “aò”. This is ignored if a difference is found on level 1 or level 2. ■ **"quaternary"** Level 4: distinguish word with and without punctuation, e.g. “ab” < “a-b” < “aB”. This is ignored if a difference is found on level 1-3, and should be used only if a distinction based on punctuation is required. ■ **"identical"** Level 5: Used if levels 1-4 yield identical results. The Unicode point values are compared. Note that this level of comparison can impact performance negatively. **Note:** This is the default root locale, i.e. the default collation attribute value according to the Unicode Collation Algorithm. However, this may be different if a specific language/locale is specified. For more information, refer to `tsd:collation` and to the collation documentation in the *XML Schema User Guide*.

Attributes

Example

See under `tsd:collation`.

106 tsd:structureIndex

Purpose	This option specifies whether a structure index is created for the corresponding doctype. In other words, it specifies whether and how nodes not declared by the schema are registered in the repository. It is of type `xs:NMTOKEN`. It can have the following values: ■ **none** No structural indexing is done. **Caution:** Do not choose this option if you intend to define a reference index within the doctype. ■ **condensed** The repository registers the existence of any node within any instance loaded into the doctype. ■ **full** The repository registers both the existence of undeclared nodes and the instances in which they occur. "The default value is condensed."
Parent element	`tsd:physical` (used in the context of `tsd:doctype`)
Child elements	None
Attributes	None

Example

The following example shows how to define a structure index using the "full" option for a doctype “Telephone”:

```
<tsd:doctype name="Telephone">
  <tsd:logical>
    <tsd:content>closed</tsd:content>
  </tsd:logical>
  <tsd:physical>
    <tsd:structureIndex>full</tsd:structureIndex>
  </tsd:physical>
</tsd:doctype>
```

107 tsd:subTreeAdabas

Purpose	Mapping to Adabas. The subtree is mapped to Adabas, must contain the `tsd:nodeAdabasField` below.
Parent element	`tsd:map`
Child elements	None
Attributes	`dbid`, `password`, `fnr` and `encoding`
Restrictions	`tsd:subTreeAdabas` is only valid if the containing `tsd:map` element belongs to a logical node that has child nodes, i.e. it must be a descendant element of a `tsd:elementInfo`, not a `tsd:attributeInfo`.

Name	Type	Description
`dbid`	`xs:string`	Specifies the Adabas database ID.
`password`	`xs:string`	Specifies the password required to access the Adabas database.
`fnr`	`xs:string`	Specifies the file number of the external Adabas file to access (see example below).
`encoding`	`xs:string`	Specifies the encoding for the node and its children (see example below). The default encoding is ISO-8859-1.

Attributes

The valid values for the `encoding` attribute are listed in the table below (each of these denoting a special encoding):

big-5
euc-kr
extended_unix_code_packed_format_for_japanese
gb2312
ibm037
ibm1026
ibm273
ibm277

ibm278
ibm280
ibm284
ibm285
ibm420
ibm424
ibm437
ibm500
ibm775
ibm850
ibm852
ibm855
ibm857
ibm860
ibm862
ibm863
ibm864
ibm865
ibm866
ibm868
ibm869
ibm870
ibm871
ibm918
iso-10646-ucs-2
iso-2022-jp
iso-8859-1 (default)
iso-8859-13
iso-8859-15
iso_8859-2
iso_8859-3
iso_8859-4
iso_8859-5
iso_8859-6
iso_8859-7
iso_8859-8
jis_encoding
koi8-r
ks_c_5608-1987
shift_jis
unicode-1-1
utf-16
utf-16be
utf-16le
utf-8

windows-1250
windows-1251
windows-1252
windows-1253
windows-1254
windows-1255
windows-1256
windows-1257
windows-1258
windows-31j

Example

This example maps an element with all its descendants to Adabas. It illustrates the use of the `tsd:subTreeAdabas` element in conjunction with the `tsd:nodeAdabasField` element of TSD:

```
<xs:element name = "employee">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:subTreeAdabas dbid = "3" fnr = "21"></tsd:subTreeAdabas>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "name">
        <xs:complexType>
          <xs:sequence>
            <xs:element name = "firstname" type = "xs:string">
              <xs:annotation>
                <xs:appinfo>
                  <tsd:elementInfo>
                    <tsd:physical>
                      <tsd:map>
                        <tsd:nodeAdabasField
                         shortname = "AC" format = "A" length = "20"/>
                      </tsd:map>
                    </tsd:physical>
                  </tsd:elementInfo>
                </xs:appinfo>
              </xs:annotation>
            </xs:element>
            <xs:element name = "surname" type = "xs:string">
              <xs:annotation>
```

```
                <xs:appinfo>
                  <tsd:elementInfo>
                    <tsd:physical>
                      <tsd:map>
                        <tsd:nodeAdabasField
                         shortname = "AE" format = "A" length = "20"/>
                      </tsd:map>
                    </tsd:physical>
                  </tsd:elementInfo>
                </xs:appinfo>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    <xs:sequence>
  </xs:complexType>
</xs:element>
```

108 tsd:subTreeAdabasPE

Purpose	This element is needed for mapping elements of the XML schema to Adabas periodic groups.
Parent element	`tsd:map`
Child elements	None
Attributes	`shortname`
Restrictions	`tsd:subTreeAdabasPE` is only allowed if the containing `tsd:map` element is a descendant element of a `tsd:elementInfo`, not a `tsd:attributeInfo`.

Name	Type	Description
`shortname`	`xs:string`	This value of this required attribute specifies the short name of the Adabas periodic group.

Attributes

Example

This example shows the `tsd:elementInfo` for an element to be mapped to an Adabas periodic group.

```
<xs:element name = "partner">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:logical>
          <tsd:collation>
            <tsd:language value="de"/>
            <tsd:strength value="secondary"/>
            <tsd:caseFirst value = "lowerFirst"/>
            <tsd:alternate value = "shifted"/>
            <tsd:caseLevel value = "false"/>
```

```
            <tsd:french value = "false"/>
            <tsd:normalization value ="false"/>
          </tsd:collation>
        </tsd:logical>
        <tsd:physical>
          <tsd:map>
            <tsd:subTreeAdabasPE shortname="p"/>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo
   </xs:appinfo>
 </xs:annotation>
 .
 .
 .
</xs:element>
```

109 tsd:subTreeSQL

Purpose	This element is needed to map a sub-tree of an XML document to an SQL table. The primary key of the mapped SQL table can be specified using the `tsd:primaryKeyColumn` child element of the `tsd:subTreeSQL` element. Furthermore, an access predicate can be defined using the `tsd:accessPredicate` child element and its child, `tsd:nodeParameter`, belonging to this element. The subtree must contain the `tsd:nodeSQL` element.
Parent element	`tsd:map`
Child elements	`tsd:primaryKeyColumn`, `tsd:accessPredicate`
Attributes	`table`, `userid`, `password`, `datasource`, `schema`

Name	Type	Description
`table`	`xs:string`	This attribute specifies the table for SQL storage of an XML sub-tree.
`userid`	`xs:string`	This attribute specifies the userid for the ODBC connection that is used for SQL storage of an XML sub-tree.
`password`	`xs:string`	This attribute specifies the password required to access a given Adabas database.
`datasource`	`xs:string`	This attribute specifies the ODBC data source for SQL storage of an XML sub-tree.
`schema`	`xs:string`	This attribute specifies the name of an external SQL schema. **Note:** Some SQL database systems do not have the concept of a schema. Then, the schema attribute must be omitted.

Attributes

Example

This example demonstrates the use of the `tsd:subTreeSQL` element and its subelements `tsd:nodeParameter`, `tsd:primaryKeyColumn` and `tsd:accessPredicate` including all attributes:

```
<tsd:subTreeSQL datasource="myAdabasd" schema="mySchema" table="myTable"
                userid="myUserID" password="myPassword" >
  <tsd:primarykeyColumn>myPrimaryKeyColumn1</tsd:primarykeyColumn>
  <tsd:primarykeyColumn>myPrimaryKeyColumn2</tsd:primarykeyColumn>
  <tsd:accessPredicate>
         "login"     = ↩
<tsd:nodeParameter>/SQLMapping2/SQLMappingElement</tsd:nodeParameter>
     AND "accountNo" = ↩
<tsd:nodeParameter>/SQLMapping2/SQLMappingElement2</tsd:nodeParameter>
  </tsd:accessPredicate>
</tsd:subTreeSQL>
```

110 tsd:systemGeneratedIdentity

Purpose	This element specifies at doctype level whether reuse of `ino:id`'s is permitted.
Parent element	`tsd:logical`
Child elements	None
Attributes	`reuse`

Name	Type	Description
`reuse`	`xs:boolean`	This attribute indicates whether reuse of `ino:id`'s is allowed. Default value: "true".

Attributes

Example

The example schema shown below prohibits the reuse of `ino:id`'s for doctype D.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="Reusage-Schema">
        <tsd:collection name="Reusage"/>
        <tsd:doctype name="D">
          <tsd:logical>
            <tsd:systemGeneratedIdentity reuse="false"/>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="D"/>
</xs:schema>
```

111 tsd:text

Purpose	This element specifies whether a text index should be generated for the current schema element or attribute, or for a non-XML doctype. See also `tsd:index`.
Parent element	`tsd:index`
Child elements	`tsd:refers`, `tsd:multiPath`
Attributes	None
Type	Empty

Example

This example illustrates the definition of a text index for a `tsd:attributeInfo` element in TSD:

```
<xs:attribute name="value">
  <xs:annotation>
    <xs:appinfo>
      <tsd:attributeInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:text/>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:attributeInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

112 tsd:trigger

Purpose	This element contains the information for logical nodes in the schema that define action triggers for the Tamino Server Extensions. Only one trigger can be specified per type (`tsd:onDelete`, `tsd:onInsert`, `tsd:onUpdate`).
Parent element	`tsd:logical`
Child elements	`tsd:onDelete`, `tsd:onInsert`, `tsd:onUpdate`
Attributes	None

Example

This example illustrates the definition of triggers for Tamino Server Extensions:

```
<xs:element name="firstname" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:logical>
          <tsd:trigger>
            <tsd:onDelete type=action>x1</tsd:onDelete>
            <tsd:onUpdate type=action>x2</tsd:onUpdate>
            <tsd:onInsert type="action">
              myXtension.onInsert
              <tsd:parameters>
                <tsd:parameter name="param1" type="xs:string">
                  MyClassName
                </tsd:parameter>
                <tsd:parameter name="param2" type="xs:int">
                  4711
                </tsd:parameter>
              </tsd:parameters>
            </tsd:onInsert>
          </tsd:trigger>
        </tsd:logical>
```

```
      </tsd:elementInfo
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

113 tsd:unique

Purpose	This element specifies that a field is to be a unique key, as described in the section *Definition of Unique Keys* of the *Tamino XML Schema User Guide*. If present, checks on uniqueness are performed for the single key or combination of keys that are specified in the `xpath` attribute of the `tsd:field` child element(s).
Parent element	`tsd:logical`
Child elements	`tsd:field` **Note:** A `tsd:unique` element can have an arbitrary number of `tsd:field` child elements.
Attributes	`name`
Restrictions	The `tsd:field` child element may only point to elements and attributes that have multiplicity 1 and are not of one of the following types: ■ `xs:duration`; ■ Any datatype that allows an optional timezone indicator, e.g. `xs:dateTime`.

Name	Type	Description
`name`	`xs:NCName`	This required attribute specifies the name of the uniqueness constraint under which it can subsequently be referenced. The name must be unique within the scope of a single doctype.

Attributes

Example 1

This example demonstrates the use of the `tsd:unique` element and its subelement `tsd:field` to define two unique keys. One unique key definition is intended for the combined key comprising the element `C` and the attribute `b` of element `B`. The other one is intended for a single key consisting of the element `D`.

```
<tsd:doctype name="A">
  .
  .
  .
  <tsd:unique name="CB-key">
    <tsd:field xpath="C" />
    <tsd:field xpath="B/@b" />
  </tsd:unique>
  <tsd:unique name="D-key">
    <tsd:field xpath="D" />
  </tsd:unique>
  .
  .
  .
</tsd:doctype>
```

Example 2

Here is another example for the `tsd:unique` element. It shows how to impose a uniqueness constraint on the root element of the schema.

```
<tsd:doctype name="A">
  .
  .
  .
  <tsd:unique name="root-key">
    <tsd:field xpath="." />
  </tsd:unique>
  .
  .
  .
</tsd:doctype>
```

114 tsd:update

Caution: The `tsd:accessOptions` element and its child elements, including `tsd:update`, are deprecated in Tamino 4.2.

Purpose	The presence of this element indicates whether `update` access is allowed for the associated doctype. If update access is permitted, existing instances of the doctype may be updated in the Tamino XML data store.
Parent element	`tsd:accessOptions`
Child elements	None
Attributes	None
Type	Empty

Example

See `tsd:accessOptions`.

115 tsd:version

Purpose	This element contains the version number of TSD schema.
Parent element	`tsd:adminInfo`
Child elements	None
Attributes	None
Type	`xs:NMTOKEN`
Restriction	This element is generated by Tamino. It is neither necessary to provide this element explicitly nor may you modify this information.

Example

See `tsd:adminInfo`.

116 tsd:which

Purpose	This element allows for different physical schema information, if a node can be referenced via multiple paths because of references to global elements or attributes. Physical schemas with the same absolute path expression can be grouped together. Therefore multiple `tsd:which` elements are allowed within a single `tsd:physical` element. If no `tsd:which` element is specified for a node, all paths have the same physical schema information. The `tsd:which` element contains a string (of type `xs:string`) describing a path expression.
Parent element	`tsd:physical`
Child elements	None
Attributes	None
Type	`xs:string`
Restrictions	The following constraints exist on the ***XPath*** expression for `tsd:which`: ■ Only the absolute path expression `/ doctypeName / element /.../{ currentElementName | @ currentAttributeName }` must exist in the current schema. All **XPath** expressions of all `tsd:physical` children of one element must be unique. This includes `tsd:physical` elements below `tsd:elementInfo` and `tsd:attributeInfo` elements below `tsd:schemaInfo` which are attached to the current element or attribute using the `context` attribute. ■ At most one `tsd:physical` without a `tsd:which` element is allowed within a single `tsd:elementInfo` or `tsd:attributeInfo`. The default for non-recursive elements covers all absolute paths to this element. If no `tsd:which` element is specified, the default applies: The default for non-recursive elements covers all absolute paths to this element. In case of possible recursions in the path to the current element or attribute only the first level of recursion is assumed.

Example

The following example contains two `tsd:physical` elements within a single `tsd:elementInfo`: the first one specifies physical schema information that only applies if the `TITLELENGTH` element occurs at one of the two paths specified by the `tsd:which` child elements. The second `tsd:physical`

does not contain a `tsd:which` ("default which"), thus it applies to all instances of the `TITLELENGTH` element not occurring at one of the explicit paths specified by the `tsd:which` elements inside other `tsd:physical` element(s)."

```
<xs:element name = "TITLELENGTH">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <!-- for /a/b/c/TITLELENGTH and /a/d/c/TITLELENGTH:
               SQL mapping and no (default) indexing -->
          <tsd:which> /a/b/c/TITLELENGTH </tsd:which>
          <tsd:which> /a/d/c/TITLELENGTH </tsd:which>
          <tsd:map>
            <tsd:nodeSQL Column="title"/>
          </tsd:map>
        </tsd:physical>
        <tsd:physical>
          <!-- default for all occurrences of TITLELENGTH:
               default native storage and standard index -->
          <tsd:native>
            <tsd:index>
              <tsd:standard/>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

117 tsd:xTension

Purpose	This element specifies mapping information for logical nodes in the schema that are mapped to the Tamino Server Extension.
Parent element	tsd:map
Child elements	tsd:onDelete, , tsd:onProcess, tsd:onUpdate
Attributes	None

Example

This example illustrates the mapping to Tamino Server Extensions:

```
<xs:element name = "firstname" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:xTension>
            <tsd:onProcess>SXSBMapping.SxsStoreToFile</tsd:onProcess>
            <tsd:onDelete>SXSBMapping.SxsDeleteFile</tsd:onDelete>
          </tsd:xTension>
        </tsd:physical>
      </tsd:elementInfo
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Index

U

W

X