software AG

Tamino Schema Editor

Version 9.5 SP1

November 2013

tamino XML Server

This document applies to Tamino Schema Editor Version 9.5 SP1.

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

Document ID: XSC-DOC-95SP1-20131030

Table of Contents

Tamino Schema Editor

The Tamino Schema Editor supports you in creating schemas that conform to both the Tamino schema language (TSD) and the XML Schema standard (XSD).

This documentation covers the following topics:

1 Introducing the Tamino Schema Editor

A Tamino schema conforms to the XML Schema standard, with Tamino-specific information written in annotations to XML Schema constructs.

Using the Tamino Schema Editor to create Tamino schemas has the following advantages:

- The graphical user interface of the Tamino Schema Editor shields you from having to type in schema language syntax, thus making schema creation much faster and less error-prone.
- The Tamino Schema Editor shields you from the complexity of the XML Schema standard by offering dialogs specifically for the creation of Tamino schemas. Schema constructs required by the XML Schema standard are added automatically to ensure that valid schemas are generated.
- The Code view helps you to see the generated schema in plain text. Text in this view can be edited and the changes will reflect in the graphical view.

It is also possible to import existing DTDs and SQL schemas and thus convert them to Tamino schemas. If you have created schemas with a previous version of Tamino (TSD3, TSD4), you can also import and thus convert them to the schema language supported by the current version of Tamino.

For detailed information on the Tamino schema language, see the *Tamino XML Schema User Guide* and the *Tamino XML Schema Reference Guide* in the Tamino XML Server documentation.

As a source of valuable information, guidance and help, developers using the Software AG Tamino platform can refer to the Tamino Developer Community web site at ***http://communities.software-ag.com/ecosystem/communities/public/developer/webmethods/default.html?product=tamino***.

2 First Steps with the Tamino Schema Editor

This tutorial provides a very simple and brief introduction to the Tamino Schema Editor. It is intended to get you, the schema developer, started with the basic steps that are required to create a simple schema from scratch.

The steps are presented in the following order:

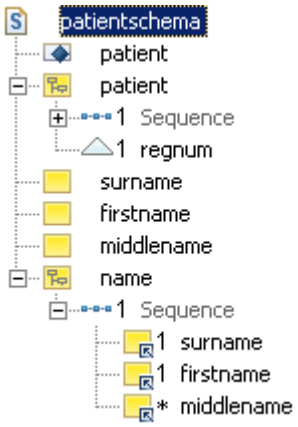
Estimated duration for this tutorial: 1 hour.

About this Tutorial

First-time users are recommended to work through this tutorial to obtain an overview of how to work with the Tamino Schema Editor. This tutorial is not intended to be a comprehensive description of the full range of possibilities provided by the Tamino Schema Editor. Therefore, explanations are kept to a minimum. For a full description of each feature, refer to the remainder of this documentation.

This tutorial illustrates the creation of a schema which contains elements that are to appear in various content models. You will declare global elements and insert references to them in the content models in which they are to be included.

When you have completed all steps of this tutorial, your schema tree will look as follows:



Instances of the schemas that you will create in the course of this tutorial are to be stored in Tamino's native XML data store.

Important background information can be found in the *XML Schema Part 0: Primer* on the W3C site ***http://www.w3.org/TR/xmlschema-0/***.

Starting the Tamino Schema Editor

The Tamino Schema Editor can be invoked from all supported Windows and UNIX platforms.

To start the Tamino Schema Editor on Windows

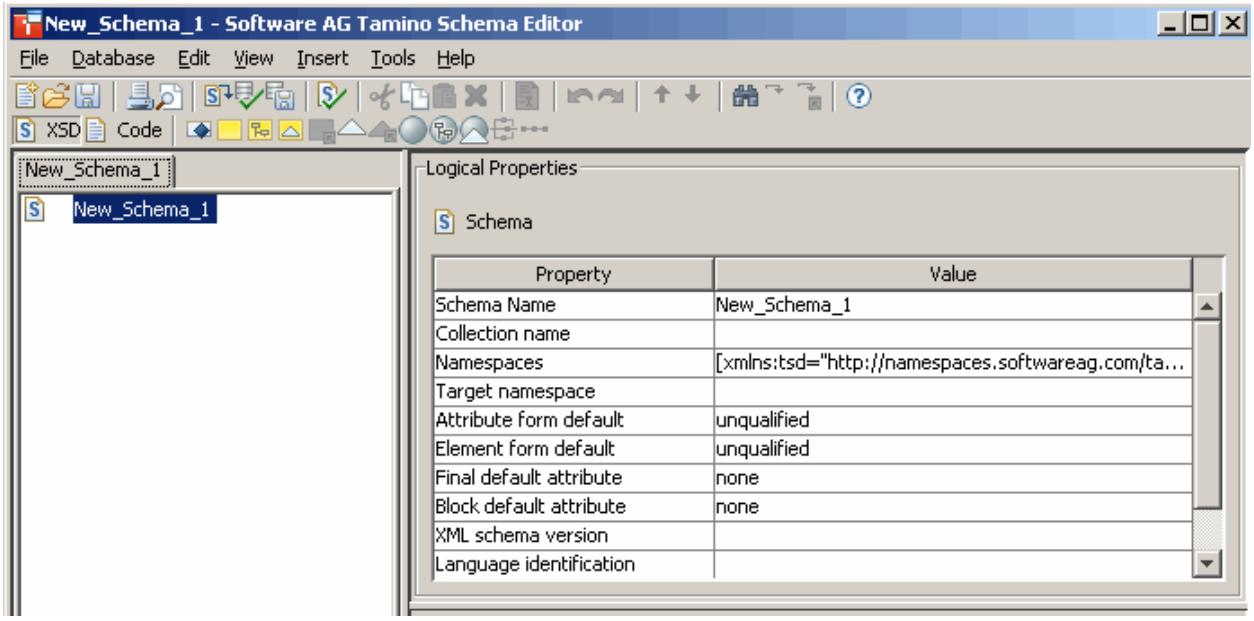
- From the Windows **Start** menu choose **Tamino Schema Editor** in the Tamino program group.

To start the Tamino Schema Editor on UNIX platforms

- Start the following script from the command line:

```
inoschema.sh
```

When you have started the Tamino Schema Editor, its application window appears. An empty schema with the name "New_Schema_1" is shown in the schema tree. Its logical properties are shown on the right side of the application window. You can immediately start with the creation of a new schema.



Specifying a Schema Name and a Collection

You will now specify the name under which the schema is to be defined in Tamino, and the name of the collection in which the schema is to be defined in Tamino.

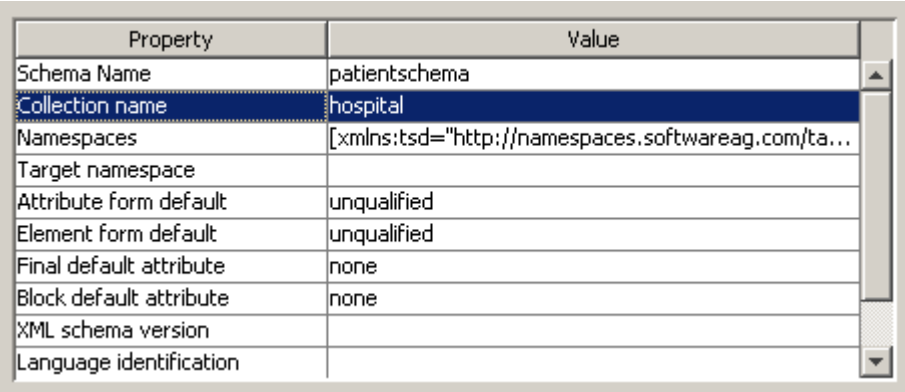
To specify schema name and collection

1 Specify the name "patientschema" for the schema.

You can do this either directly in the schema tree, or in the **Value** column of the logical properties.

Tip: Keyboard users can switch between schema tree and logical properties as follows: Use CTRL+TAB to switch to the schema tree group tab, then select the schema, then use TAB to get the schema node selected. The input field in the schema tree or in the **Value** column can be activated using F2.

2 In the logical properties, specify "hospital" as the collection name.



For a description of all available properties, see *Properties Explained* later in this documentation.

Checking the Generated Code in the Code Editor

You will now check the code that has so far been generated for the schema.

To invoke the code editor

1 From the **View** menu, choose **Code**.

Or:

Choose the following toolbar button:

The code editor is now shown in the application window. The following XML Schema code has been generated:

```
<?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 = "patientschema">
        <tsd:collection name = "hospital"></tsd:collection>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

Note that the XML Schema constructs refer to the namespace prefix "xs", and the Tamino-specific constructs refer to the Tamino schema namespace "tsd".

Before continuing with the next exercise, you will return to tree view.

Caution: If you modify the schema in code view and define elements that are not supported in tree view, it is not possible to switch back to tree view. The output panel at the bottom of the application window will then inform you why the switch is not possible.

2 From the **View** menu, choose **XSD**.

Or:

Choose the corresponding toolbar button:

The schema tree is shown again.

Inserting the Root Element and the Doctype

You will now insert the complex element "patient" in your schema. This is the root element for all instances of this schema. When the root element has been inserted, you will also insert the doctype. The names of the root element and doctype must be identical.

Note: It is possible to create several root elements and doctypes.

To insert the root element

1 In the schema tree, select the schema node with the name "patientschema".

2 From the **Insert** menu, choose **Element complex**.

Or:

Choose the following toolbar button:

A declaration for a complex element with the name "NEW_element_complex" is added to the schema tree. The logical and physical properties for the complex element are shown on the right of the application window.

3 Specify "patient" as the value of the `Name` logical property for the complex element.

Leave the logical property `Mixed content` with its default value (false). This means that the element must not contain arbitrary characters and elements (no mixed content).

In the physical properties, leave the storage type and the index with the default values (native storage type and no index).

When the storage type is "Native", instances will be stored in Tamino's native XML store.

When an index has not been defined, instances of this node will not be indexed.

With the physical properties `Collection reference`, `dereference`, `Node reference` and `Node reference operator`, which are visible when you choose the **Reference** tab of the physical properties, a join with another document instance can be defined. These properties are set to their defaults, that means, no join is defined.

The Tamino Schema Editor generates the following schema code for the "patient" node:

```
<xs:element name = "patient">
  <xs:complexType></xs:complexType>
</xs:element>
```

This is the root node for all element declarations that you will add later.

To define the doctype

1 In the schema tree, select the schema node with the name "patientschema".

2 From the **Insert** menu, choose **Doctype**.

Or:

Choose the following toolbar button:

A doctype node with the name "NEW_doctype" is added to the schema tree.

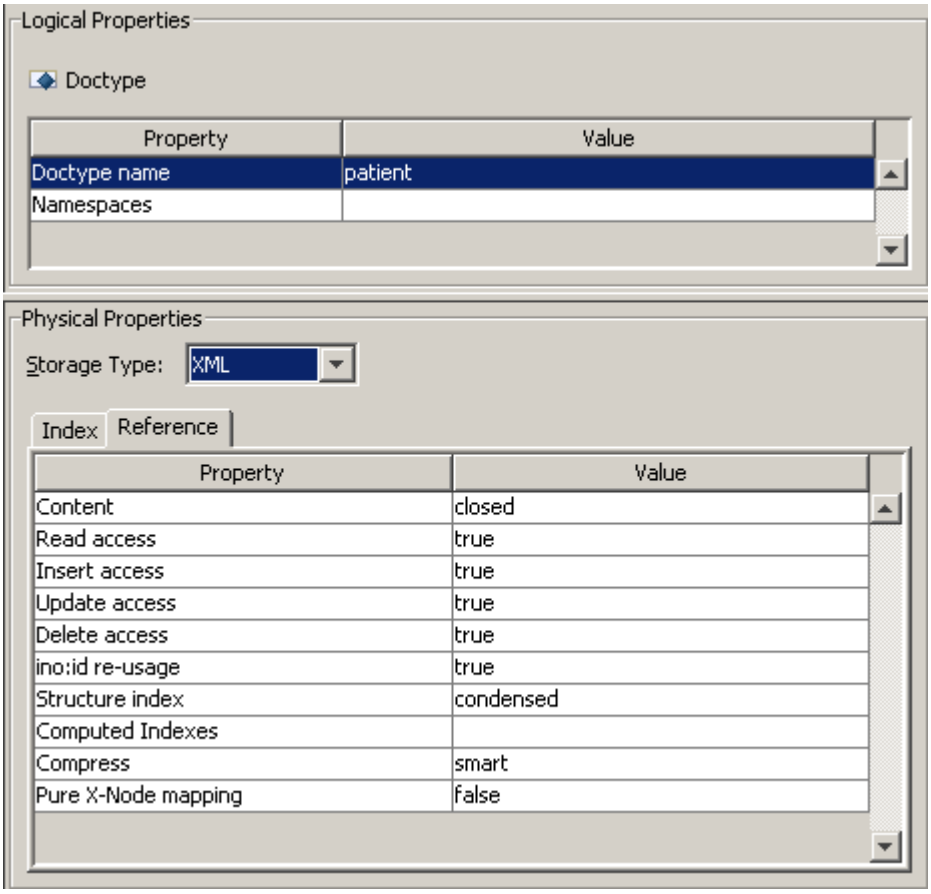
3 In the logical properties, select the "NEW_doctype" entry in the "Value" column.

A drop-down arrow is shown. The drop-down list box provides for selection all elements that have been inserted directly below the schema name.

Tip: Keyboard users can open the drop-down list box using ALT+DOWN-ARROW.

4 From the drop-down list box, select **patient** as the name for the doctype.

In the physical properties, leave the `Content` property and the different access properties with their default values. This means that all instances of doctype "patient" in the Tamino XML store can be read, inserted, updated and deleted.


Logical Properties
Doctype
Property
Value
Doctype name
patient
Namespaces
Physical Properties
Storage Type:
XML
Index
Reference
Property
Value
Content
closed
Read access
true
Insert access
true
Update access
true
Delete access
true
ino:id re-usage
true
Structure index
condensed
Computed Indexes
Compress
smart
Pure X-Node mapping
false

Inserting Simple Elements

You will now insert simple elements that you will later reuse by referencing them.

To insert simple elements

1 In the schema tree, select the schema node with the name "patientschema".

2 From the **Insert** menu, choose **Element simple**.

Or:

Choose the following toolbar button:

A declaration for a simple element with the name "NEW_element_simple" is added to the schema tree.

3 Specify the name "surname" for the simple element.

Leave the logical and physical properties with their default values.

4 Insert two more simple elements by repeating the above steps. Specify the names "firstname" and "middlename", in that order. Make sure to select the schema node with the name "patients-chema" before inserting/pasting a simple element.

Tip: You can also use the **Copy** and **Paste** commands from the **Edit** menu.

The Tamino Schema Editor generates the following schema code for the simple elements. They are inserted in the schema as children of the root element `xs:schema`.

```
<xs:element name = "surname" type = "xs:string"></xs:element>
<xs:element name = "firstname" type = "xs:string"></xs:element>
<xs:element name = "middlename" type = "xs:string"></xs:element>
```

Inserting a Complex Element

You will now insert the complex element "name" to which you will later add the references for the simple elements that you have defined in the previous exercise.

To insert a complex element

1 In the schema tree, select the schema node with the name "patientschema".

2 From the **Insert** menu, choose **Element complex**.

Or:

Choose the following toolbar button:

A complex element with the name "NEW_element_complex" is added to the schema tree.

3 Specify the name "name" for the complex element.

4 Leave the logical properties with their default values.

5 In the physical properties, leave the storage type with its default value "Native" so that instances will be stored in Tamino's native XML store.

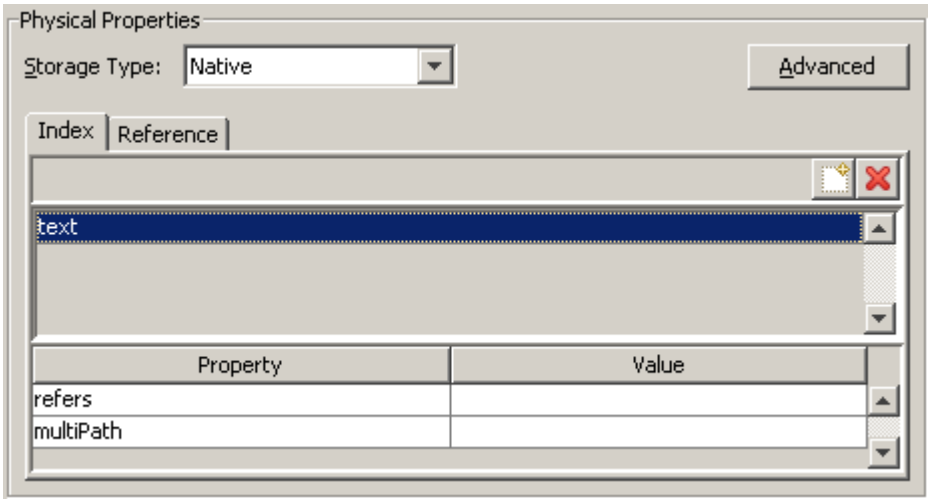
6 Create a "text" index. This means that instances of the element will be indexed for full-text searches. To do so, choose the following button on the **Index** tab:

A new entry with the name "standard" is shown on the **Index** tab.

7 Select the new entry.

A drop-down arrow is shown at the right. The drop-down list box provides for selection different types of indices.

8 From the drop-down list box, select **text**.



Inserting a Sequence

You will now insert a sequence as a child node of "name". You will later add element references to the sequence. In an instance, all elements within a sequence must appear in the same order that has been defined in the schema.

To insert a sequence

- In the schema tree, select the complex element "name".

  From the **Insert** menu, choose **Sequence**.

  Or:

  Choose the following toolbar button:

  A structural element named "Sequence" is added to the complex element. Leave the logical properties with the default values.

Adding Element References to the Sequence

You will now add three element references ("surname", "firstname" and "middlename") to the sequence you have just created.

To add element references to the sequence

1 In the schema tree, select the "Sequence" element.

2 From the **Insert** menu, choose **Element reference**.

Or:

Choose the following toolbar button:

An element reference is added to the sequence.

3 For the logical property `Reference`, select **surname** from the drop-down list box that is available in the **Value** column.

Leave all other properties with the default values.

4 Insert two more element references by repeating the previous steps. Select the names "firstname" and "middlename", in that order.

Tip: You can also use the **Copy** and **Paste** commands from the **Edit** menu.

Make sure to select the "Sequence" element before inserting/pasting an element reference.

5 Select the element reference "middlename".

6 Change the following logical properties: set `Minimum occurrence` to "0" and `Maximum occurrence` to "unbounded".

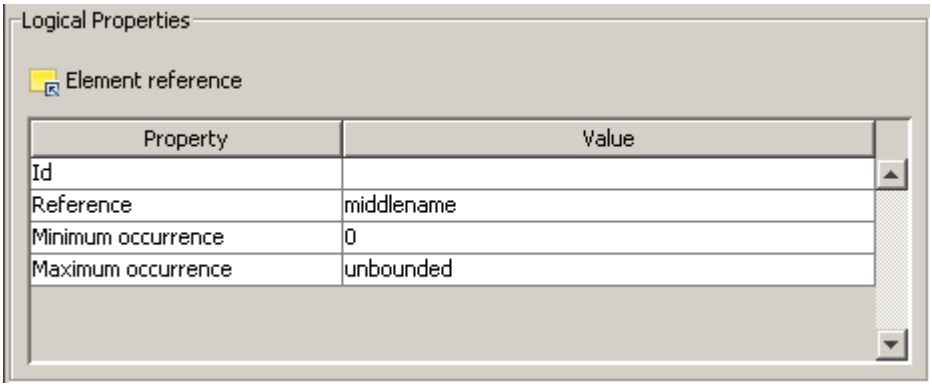
Or:

From the **Edit** menu, choose **minOccurs : maxOccurs > 0 : unbounded**.

This means that "middlename" is optional and can occur as often as desired. This is indicated by an asterisk (*) next to the icon in the schema tree.

For "surname" and "firstname", use the default values (1). They are required to occur once. More occurrences are not allowed. This is indicated by the number 1 next to the icon in the schema tree.

The following example shows the properties for "middlename" for which different occurrence constraints have been specified.



The Tamino Schema Editor generates the following schema code for the "name" element:

```
<xs:element name = "name">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:text></tsd:text>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref = "surname"></xs:element>
      <xs:element ref = "firstname"></xs:element>
      <xs:element ref = "middlename" minOccurs = "0" maxOccurs = ↩
"unbounded"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The default values (1) for the properties `Minimum occurrence` and `Maximum occurrence` have not been changed in the elements "surname" and "firstname". Therefore, no code for these attributes has been generated.

Adding a Sequence with an Element Reference to the Root Element

You will now add an element reference to your root element. This element references the complex element "name" which in turn references the simple elements "surname", "firstname" and "middlename".

To add a sequence with an element reference to the root element

1 In the schema tree, select the complex element "patient".

From the **Insert** menu, choose **Sequence**.

Or:

Choose the following toolbar button:

A structural element named "Sequence" is added to the schema tree.

2 Select this new "Sequence" element in the schema tree.

3 From the **Insert** menu, choose **Element reference**.

Or:

Choose the following toolbar button:

An element reference is added to the sequence.

4 For the logical property `Reference`, select **name** from the drop-down list box.

Leave all other properties with the default values.

The Tamino Schema Editor generates the following schema code for the "patient" element:

```
<xs:element name = "patient">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref = "name"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Inserting an Attribute

You will now add an attribute to complex element "patient".

To insert an attribute

1 In the schema tree, select the complex element "patient".

2 From the **Insert** menu, choose **Attribute**.

Or:

Choose the following toolbar button:

An attribute with the name "NEW_attribute" is added to the complex element.

3 Specify the name "regnum" (for registration number) for the attribute.

4 For the logical property `Data type`, select the value **xs:integer** from the drop-down list box in the **Value** column.

This is a data type from a namespace that is by default defined for your schema.

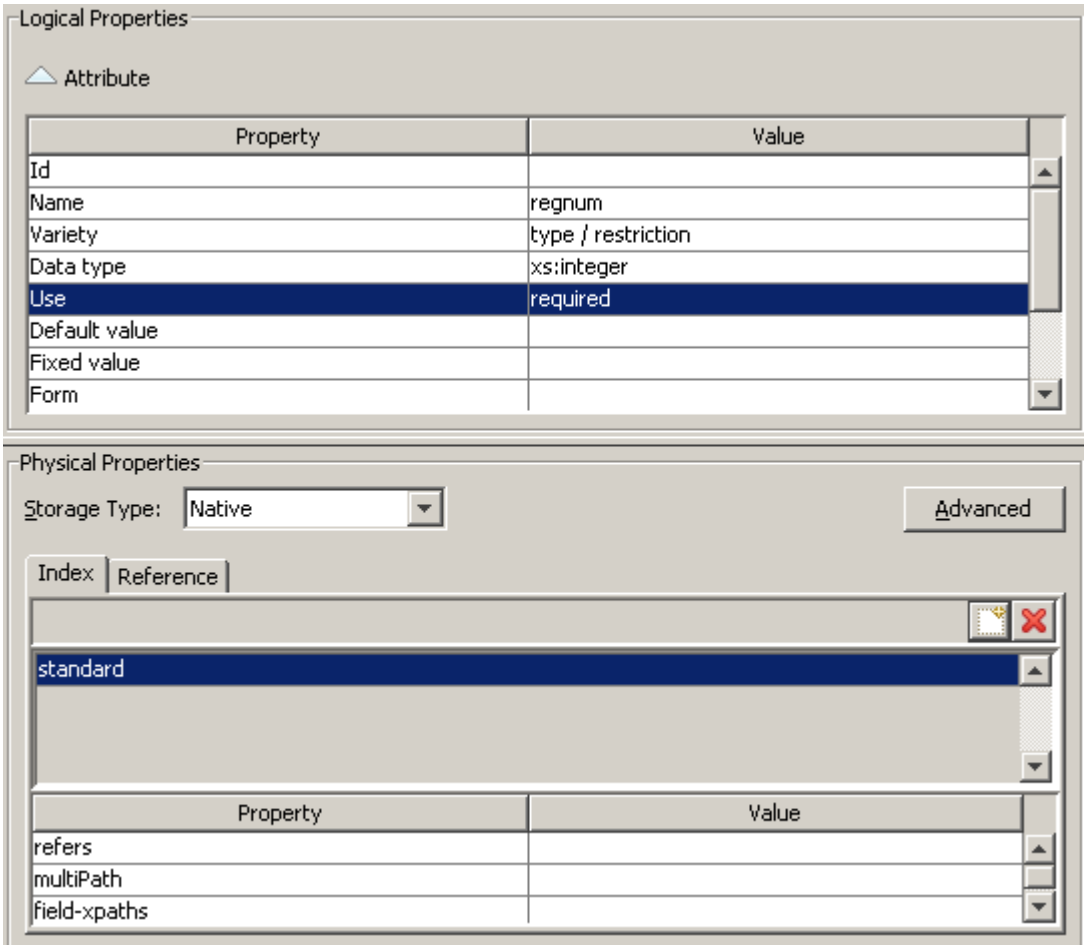
5 For the logical property `Use`, select the value **required** from the drop-down list box.

When an attribute is required, this is indicated by the number 1 next to the attribute icon in the schema tree.

6 In the physical properties, leave the storage type with the default value "Native" so that instances will be stored in Tamino's native XML store.

7 Create a "standard" index. This means that instances of the attribute will be indexed to optimize queries that use numerical comparisons in retrieval expressions. To do so, choose the following button on the **Index** tab:

A new entry ("standard") is shown on the **Index** tab.



The Tamino Schema Editor generates the following schema code for the attribute:

```
<xs:attribute name = "regnum" type = "xs:integer" use = "required">
  <xs:annotation>
    <xs:appinfo>
      <tsd:attributeInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:standard></tsd:standard>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:attributeInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

Note the constructs for the physical schema within the `attributeInfo` element.

Saving the Schema to the File System

You will now save your new schema to the file system.

You can also define your new schema in Tamino. However, this is not part of this tutorial. See *Defining and Updating a Schema in Tamino* later in this documentation.

To save the schema to the file system

1 From the **File** menu, choose **Save As**.

The **Save As** dialog box appears.

2 Select the folder in which the schema is to be saved and specify a file name, for example, *patientschema*.

3 Select the file type with which the schema is to be saved. For a Tamino schema, the file type is "tsd".

4 Select the encoding with which the schema is to be saved.

The encoding is written to the XML declaration of the schema file. The default encoding is UTF-8.

Note: The **Encoding** drop-down list box is editable.

5 Choose the **Save** button.

You have successfully completed this tutorial.

3 Starting and Leaving the Tamino Schema Editor

This chapter covers the following topics:

Starting the Tamino Schema Editor

The Tamino Schema Editor can be invoked from all supported Windows and UNIX platforms.

To start the Tamino Schema Editor on Windows

- From the Windows **Start** menu choose **Tamino Schema Editor** in the Tamino program group.

To start the Tamino Schema Editor on UNIX platforms

- Start the following script from the command line:

```
inoschema.sh
```

Using Help

The complete Tamino Schema Editor documentation is available in HTML format.

To invoke the overview page for the documentation

- From the **Help** menu, choose **Contents**.

  Or:

  Press F1.

  Or:

  Choose the following toolbar button:

To invoke context-sensitive help for an element of the application window

1 From the **Help** menu, choose **Help on Item**.

The mouse pointer changes:

2 Click the element of the application window (for example, the schema tree) to display help for this element.

To invoke context-sensitive help for a dialog box

- Choose the **Help** button in the dialog box.

  Or:

  Press F1.

Leaving the Tamino Schema Editor

When you leave the Tamino Schema Editor, all open connections to databases are disconnected automatically. Visited databases are remembered. The four recently opened files are remembered (see the commands at the bottom of the **File** menu).

To leave the Tamino Schema Editor

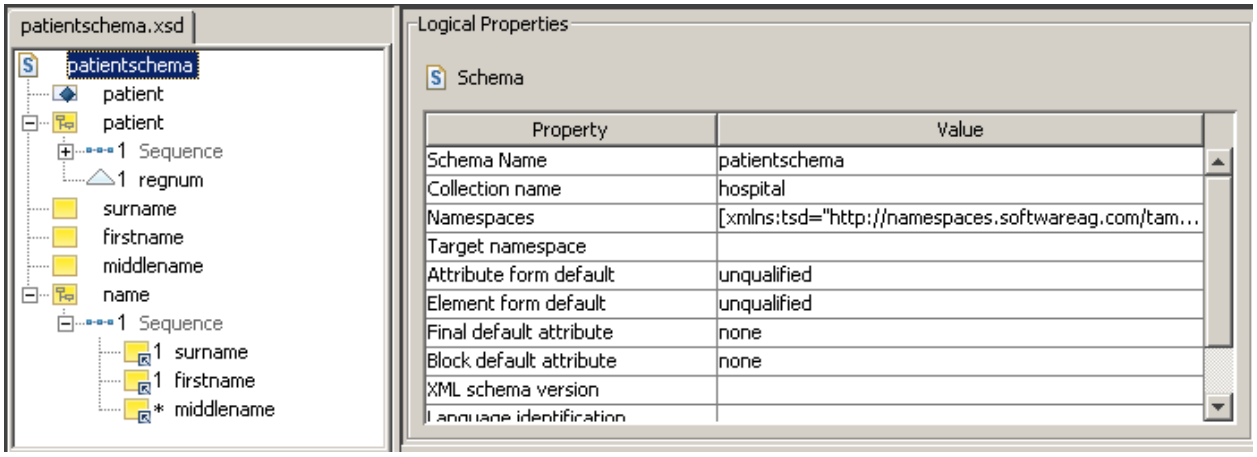
- From the **File** menu, choose **Exit**.

  Or:

  From the Control menu, choose **Close**.

4 Elements of the Application Window

When you start the Tamino Schema Editor, its application window appears. When a schema has been loaded into the application window, it may look as follows:



The application window contains several panes. To modify the size of a pane, move the mouse pointer over the border separating the panes until the pointer changes, showing two arrows pointing in opposite directions. Then drag the border using the mouse until the panes have the desired size.

For details on keyboard access not mentioned in this documentation, consult the documentation of the individual operating system or software product.

The following topics are covered below:

Menu Bar

The following menus are available:

Menu	Using the commands in this menu, you can ...
File	Create a new schema, open and save a schema in your file system, import a DTD or a schema and thus convert it to the Tamino schema syntax, or exit the Tamino Schema Editor.
Database	Get a schema from Tamino, validate the schema, define a new schema/multiple schema in Tamino, update existing schema/schemas in Tamino, or undefine a schema/group of schemas in Tamino.
Edit	Edit the current schema (for example, cut, copy and paste information, find and replace information, move an element up and down in the schema tree, or go to the declaration for a reference).
View	Switch on and off different elements of the application window, switch from tree view to code view (and vice versa), or open the referenced schema in a new window.
Insert	Insert structures for the logical properties of a schema. The commands in this menu vary, depending on the current selection in the schema tree.
Tools	Map an SQL table to a Tamino schema (ODBC wizard), map an Adabas file to a Tamino schema (Adabas wizard), transform a schema construct, load elements from the referenced schema into a selection list, validate the XML schema code, or specify options for external schemas.
Help	Invoke the online documentation.

For reference information on each menu command, see ***Menu Commands***.

Toolbar

You can execute the most important functions using a toolbar button. The following toolbars are available:

- Standard Toolbar
- Insert Toolbar

Standard Toolbar

The following toolbar buttons are available from the standard toolbar:

- Create new schema
- Open schema in file system
- Save schema in file system
- Get schema from Tamino
- Validate Schema
- Define/update schema in Tamino
- Validate XML schema code
- Cut information
- Copy information
- Paste previously cut or copied information
- Delete information
- Undo previous action
- Redo previous undo action
- Move element up in the tree
- Move element down in the tree
- Find information
- Go to declaration for a reference
- Go to next reference
- Access online documentation

To switch the standard toolbar display on and off

- From the **View** menu, choose **Toolbars > Standard**.

When the standard toolbar is displayed in the application window, a check mark is shown next to this menu command.

Insert Toolbar

This toolbar provides buttons for the commands of the **Insert** menu. The available buttons vary, depending on the current selection in the schema tree.

The items in the insert menu allow you to add new nodes to the schema, for example doctypes, elements, attributes, simple and complex types, sequences and choices.

To add a new node to the schema

1 In the graphical view, select the node under which you wish to insert the new node.

2 From the **Insert** menu, choose the required new node.

To switch the insert toolbar display on and off

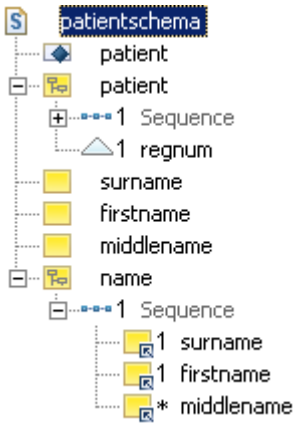
- From the **View** menu, choose **Toolbars > Insert**.

  When the insert toolbar is displayed in the application window, a check mark is shown next to this menu command.

Schema Tree

The Tamino Schema Editor provides different views. See *Switching to Another View* for further information.

When one of the tree views is active, the left side of the application window shows a tree of the current Tamino schema, for example:



Note: When you start the Tamino Schema Editor, the tree shows a new, empty schema named "New_Schema_1". You can now either create a new schema or load an existing schema into the editor. See *Creating a Schema from Scratch* or *Getting and Opening Schemas*.

You can expand or collapse the tree structure by clicking the plus or minus sign in front of an item. When using the keyboard, select the node and press RIGHT-ARROW or LEFT-ARROW.

When you select an item, the **logical** and/or **physical** properties of this item are displayed on the right side of the application window, but the item is not expanded.

The following icons may appear in the tree (see *Schema Tree Items Explained* for further information):

- **All**
- **Annotation**
- **Any**
- **AnyAttribute**
- **Appinfo**
- **Attribute**

 **AttributeGroup**

 **AttributeGroup reference**

 **Attribute info**

 **Attribute reference**

 **Choice**

 **ComplexType**

 **Documentation**

 **Doctype**

 **Element complex**

 **Element info**

 **Element reference**

 **Element simple**

 **Element unknown**

 **Element with attributes**

 **Group**

 **Group reference**

 **Import**

 **Include**

 **Key**

 **Keyref**

 **Notation**

 **Redefine**

 **Schema**

 Sequence

 **SimpleType**

 **SimpleType with attributes**

 **Unique**

Characters may appear to the right of an icon in the schema tree. See *Specifying the Occurrence Constraints* and *Declaring an Element as Optional, Required or Prohibited* for detailed information.

To expand all nodes in the tree

- From the **View** menu, choose **Expand All**.

Schema Status

When you modify a schema, the tab containing the name of the schema is shown with an asterisk "*". This is a reminder that your changes have not yet been saved. When you save the schema, for example using **File > Save**, the asterisk is removed.

Tooltips in the Schema Tree

In the schema tree, tooltips indicating the type of schema construct can be shown. By default, this feature is disabled. To enable this feature, you have to add the following line to the file *parameter.properties* which is located in the root directory of your Tamino Schema Editor installation:

```
com.softwareag.xtools.schematools.schemaeditor.components.SchemaTree.SchemaTree.showTreeNodeToolTip=true
```

Structure Info Text

Structure info text is the gray information in the schema tree (for example, "Sequence", "Choice" or "Annotation"). For better readability, you can disable the display of this information.

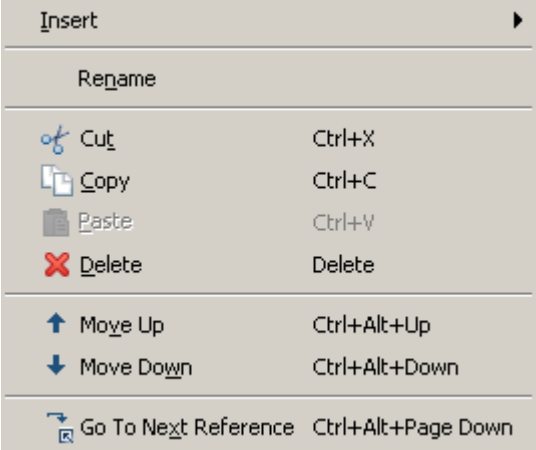
To display/hide the structure info text in the schema tree

- From the **View** menu, choose **Show Structure Info Text**.

  When the structure info text is displayed in the schema tree, a check mark is shown next to this menu command.

Context Menus

Context menus are available for the elements in the schema tree. For example, the following context menu is available when you select a simple element:



To invoke a context menu for an element or attribute

1 Select the element or attribute for which you want to invoke a context menu.

2 Click the right mouse button.

Or:

Press SHIFT+F10.

The context menu appears and you can now choose the required command.

A context menu with features that apply globally to a schema (such as several of the entries in the **File** menu) is also available from the schema tab.

To invoke the context menu for a schema

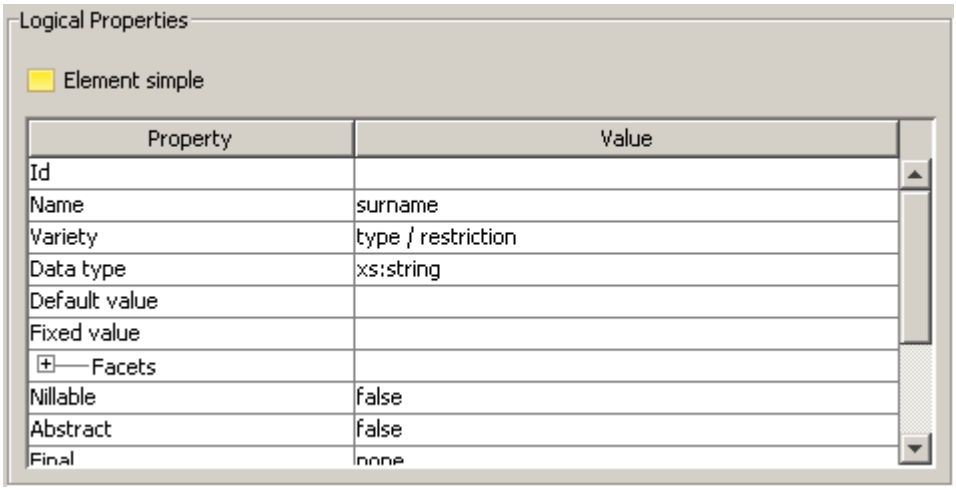
1 Move the cursor to the tab where the name of the schema is displayed.

2 Click the right mouse button.

The context menu appears and you can now choose the required command.

Logical Properties

When one of the tree views is active, the logical properties of the element that is currently selected in the schema tree are shown at the top right of the application window. Example:



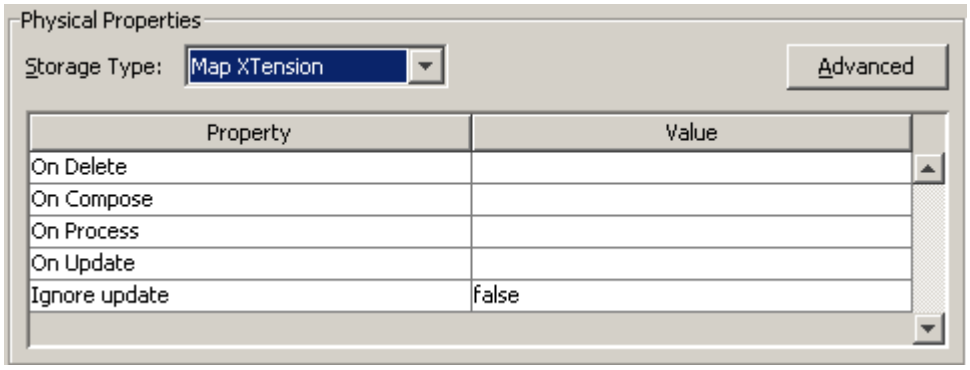
The **Property** column displays the property name. The **Value** column displays the current property value, which you can modify.

You can expand or collapse the tree structure by clicking the plus or minus sign in front of an item. When using the keyboard, select the node, press F2 to activate edit mode and then and press CTRL+RIGHT-ARROW or CTRL+LEFT-ARROW.

See *Properties Explained* for detailed information.

Physical Properties

When one of the tree views is active, the physical properties of the element that is currently selected in the schema tree are shown at the bottom right of the application window. The physical properties are shown according to the selected storage type. Example:



The **Property** column displays the property name. The **Value** column displays the current property value, which you can modify. See *Properties Explained* for detailed information.

For some element types, an **Advanced** button is provided. When you choose this button, additional information is shown. You can then define the physical properties of a node that can be accessed via multiple paths. See *Advanced Physical Properties*.

Note: If physical properties cannot be defined for an element (such as Sequence or Choice), the **Physical Properties** frame is not displayed.

Switching to Another View

When you open a schema, it is always shown in a view which is able to display the whole schema.

A schema which does not match XSD is opened in **code view**. You can modify the code manually in code view. However, it is only possible to switch back to one of the tree views if this tree view is able to display the whole schema.

Caution: The undo/redo history is cleared when you switch to another view.

To switch to tree view

This description assumes that you want to switch to tree view or that **code view** is currently active.

- From the **View** menu, choose **XSD**.

  If the current schema contains information which is not supported in another view, switching is not possible. The **output panel** will then inform you why the switch is not possible.

  In the **View** menu, a check mark is always shown next to the menu command which corresponds to the current view. The toolbar button for the current view is always shown in down status.

To switch to code view

This description assumes that one of the tree views is currently active.

- From the **View** menu, choose **Code**.

  Or:

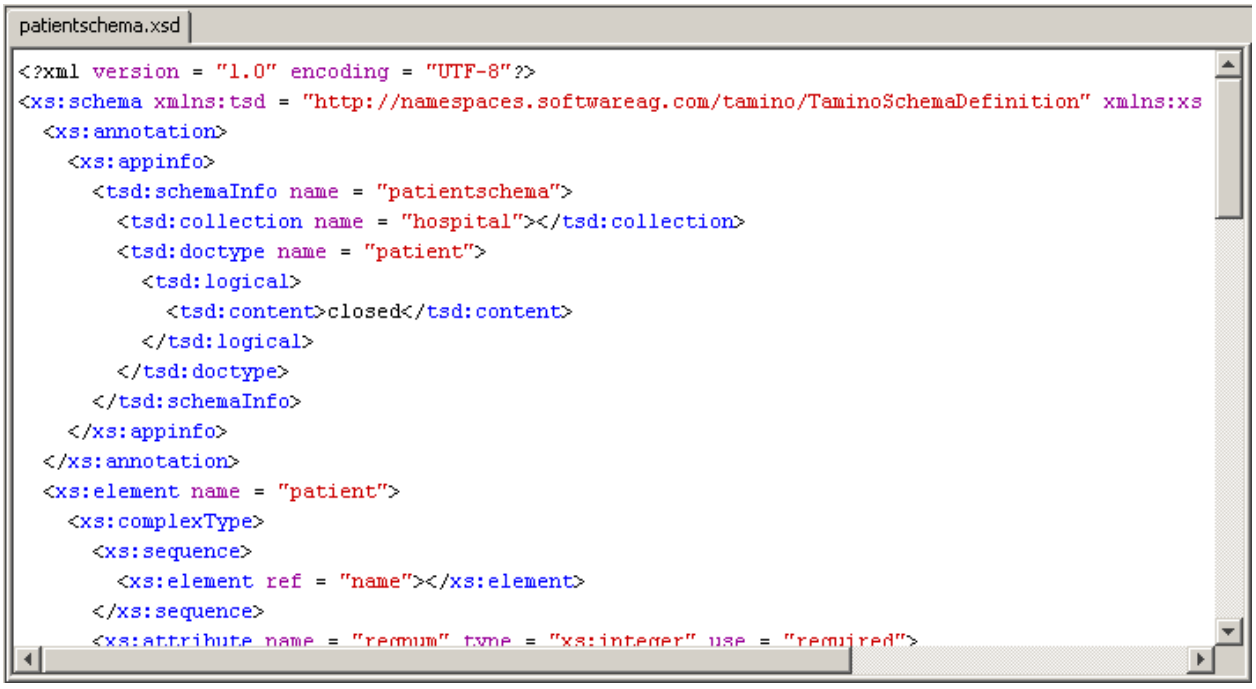
  Select the toolbar button for code view.

  In the **View** menu, a check mark is always shown next to the menu command which corresponds to the current view. The toolbar button for the current view is always shown in down status.

Code View

When you switch to code view, a code editor is shown instead of the schema tree, logical properties and physical properties which are shown in tree view.



Using the commands from the **Edit** menu, you can, for example, cut and paste or find and replace information.

Note: Not all commands are available in code view. Database access is not possible. Thus, commands such as **Define Schema** or **Validate Schema** are not available. The **Insert** menu is only available when one of the tree views is active.

Output Panel

The output panel at the bottom of the application window displays status messages about the last action that has been performed. For example, it tells you that a replace operation has been completed, or it provides information about errors that appear when you try to switch from code view to one of the tree views.

If a hyperlink is provided for an error, you can choose it. You are then positioned at the entry that caused the error. If you are working in code view, the corresponding line and column numbers are provided at the beginning of the hyperlink text.

To switch the output panel display on and off

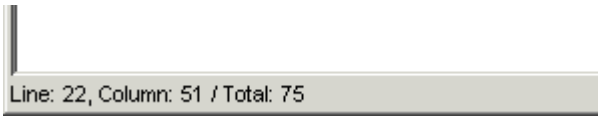
- From the **View** menu, choose **Output**.

  When the output panel is displayed in the application window, a check mark is shown next to this menu command.

Status Bar

The status bar is the horizontal information line at the bottom of the main application window.

In code view, the following information is shown: the numbers of the current line and column, and the total amount of lines in the current schema.



To switch the status bar display on and off

- From the **View** menu, choose **Status Bar**.

  When the status bar is displayed in the application window, a check mark is shown next to this menu command.

Dialog Boxes

Most Tamino Schema Editor dialog boxes can be resized. This is indicated by the following symbol in the lower right corner of a dialog box:

To resize a dialog box

1 Move the cursor to the bottom-right corner of the dialog box. The cursor shape changes to a two-headed arrow.

2 Using the left mouse button, drag the box to the required size.

Resizing a dialog box is helpful if a string is so long that it is not fully shown.

Navigating in the Schema Editor

You can navigate in the Tamino Schema Editor using the standard mouse and/or keyboard techniques for navigating through the menus and selecting items. You can edit names of elements and attributes both in tree view, as well as in the corresponding property pane.

Keyboard users should additionally note the use of the following keys:

Key	Function
F2	Activate the selected input field for editing (edit mode).
ALT+DOWN-ARROW and ALT+UP-ARROW	Open and close a drop-down list box in a dialog box. In edit mode: open and close a drop-down list box in the **Value** column of the logical or physical properties.
CTRL+RIGHT-ARROW and CTRL+LEFT-ARROW	In edit mode: open and close a group tree in the **Property** column of the logical properties.
TAB	Select the next control, or select the next item in a table.
CTRL+TAB	If a control uses TAB for its own purposes (for example, a table), you have to use CTRL+TAB to leave the context of current control.
CTRL+ENTER	If a control uses ENTER for its own purposes (for example, the property editor in which you enter documentation text), you have to use CTRL+ENTER to choose the default button.

5 Managing Schemas in Tamino and in the File System

This chapter covers the following topics:

Creating a Schema from Scratch

When you create a new schema, your new schema is by default created according to the Tamino schema language of the current Tamino version.

When you start the Tamino Schema Editor, the name "NEW_schema_1" is shown in the schema tree and you can immediately create a new schema.

Note: See *First Steps with the Tamino Schema Editor* for a brief tutorial which explains the creation of a simple Tamino schema from scratch.

To create a new schema

1 From the **File** menu, choose **New**.

Or:

Press CTRL+N.

Or:

Choose the following toolbar button:

If the currently displayed schema has been changed, a dialog box appears asking whether you want to save the changes. You can choose either the **Yes** or **No** button.

An empty schema with the name "NEW_schema_1" is shown in the schema tree.

2 Specify a name for the schema.

You can do this either in the schema tree or in the pane displaying the logical properties. This is the name under which the schema will later be defined in Tamino.

3 In the pane displaying the logical properties, specify the collection name.

This is the collection in which the schema will later be defined in Tamino.

4 From the **Insert** menu, choose **Doctype**.

A node with the name "NEW_doctype" is shown in the schema tree.

5 Specify a name for the doctype.

You can do this either in the schema tree or in the pane displaying the logical properties.

Important: Schema name, collection name and doctype name are mandatory. See *Missing Information* for further information.

6 In the pane displaying the physical properties for the doctype, select the required storage type and define all required properties. See *Properties Explained* for descriptions of all logical and physical properties.

7 Use the commands from the **Insert** menu to add further schema constructs (for example, complex elements and element references). See *Inserting an Element in the Schema Tree* for further information.

8 Define the new schema in Tamino and/or save it in your file system. See *Defining and Saving Schemas* for further information.

Missing Information

Certain information items must be specified in a schema before it can be defined to Tamino or validated against a database. These items are:

- schema name
- collection name
- doctype name

If any of these information items are missing, the Schema Editor prompts you to supply the missing information.

See also: *Defining and Updating a Schema in Tamino* and *Validating a Schema*.

Connecting to Tamino

Any interaction with the Tamino Server requires that you are connected to it. If you are not, you have to establish the connection.

Important: The web server as well as the target Tamino database must be up and running.

The connection can be established from the following dialog boxes:

- **Define Schema**
- **Undefine Schema**
- **Get Schema**
- **Validate Schema**

All servers and databases that you have previously accessed are automatically remembered. They are shown in the above dialog boxes. When you do not need access to a specific server or database any longer, you can remove its entry from the dialog box. The following buttons are provided in the above dialog boxes:


Go up one level.


Connect to a Tamino database. See below.


Remove the selected server or database from the dialog box. The selection is removed immediately. You are not asked to confirm the deletion.

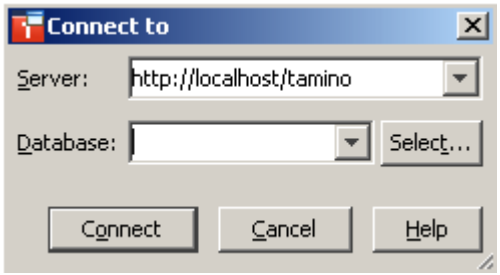

Explore the selected database. See below.

To connect to a Tamino database

1 Access one of the dialog boxes listed above and choose the following button:

Note: The **Connect** dialog box appears automatically when you initiate an action which requires database access. Thus, it is not always necessary to use this button.

The **Connect** dialog box appears.



If this is the first time you are connecting to a server, the drop-down list boxes are empty and you have to type in the required information. The next time you access this dialog box, this information is available and can be selected from the drop-down list boxes.

2 Enter the path to the required server in the following format:

```
http://host/tamino
```

where *host* is the name of the computer on which the Tamino Server resides. You can specify "localhost" for your local computer, or a path such as "yourpc.ourcompany.com" for a remote computer.

Or:

Select the required server from the corresponding drop-down list box.

3 Enter the name of the database to which you want to connect.

Or:

Select the required database from the corresponding drop-down list box.

Or:

Choose the **Select** button to display all databases on the specified server in an additional dialog box. You can then choose the desired database. This feature works only with databases that have been created starting with version 4.2.1. In all other cases, a message appears saying that a list of databases is not available.

4 Choose the **Connect** button.

When a password is required for the database, a database login dialog box appears.

5 Specify the user ID and password with which you are defined to Tamino security and/or the authentication file of your web server (if authentication is active), and then choose the **Login** button.

If you have to specify a domain name with your user ID, specify it in the following way:

```
domain-name/user-ID
```

If the authentication of your web server is not active (which is the default setting) and you have no explicit user ID/password entry in the Tamino security file, you can enter any combination here. You are then assigned default access rights in Tamino.

The last user ID you entered in this text box is automatically remembered.

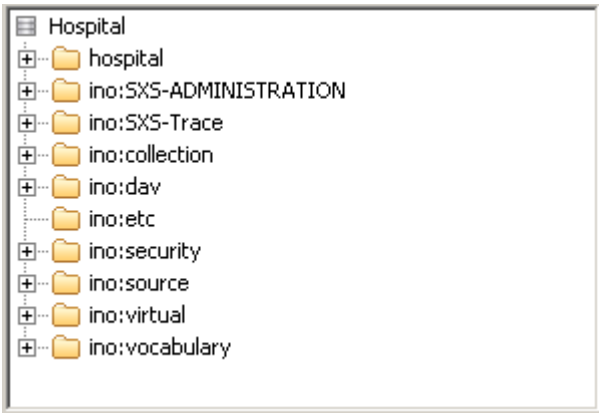
To explore the selected database

1 Select a database in one of the dialog boxes from which a connection can be established (see the list at the beginning of this topic).

2 Choose the **Explore Database** button, which is located next to the **Look in** drop-down list box:

Note: The **Connect** dialog box appears automatically when the connection to this database has not yet been established. In this case, you have to proceed as described above.

The **Explore** dialog box appears, showing the structure of the database. Here is an example for the *Hospital* database:



You can now explore the contents of the database.

3 Choose the **Close** button to close this dialog box.

Validating a Schema

Only valid schemas can be defined in Tamino. Even if the XML schema code is valid, the schema can only be defined in Tamino if the Tamino-specific properties are valid.

It is recommended that you validate the schema before defining the schema to Tamino.

To validate a schema

- From the **Database** menu, choose **Validate Schema**.

  Or:

  Choose the following toolbar button:

  All errors are reported in the **output panel**.

Validating the XML Schema Code

To validate the XML schema code for the current schema

- From the **Tools** menu, choose **Validate XML Schema**.

  Or:

  Choose the following toolbar button:

  All errors are reported in the **output panel**.

Defining and Saving Schemas

There are different ways for storing and updating a schema, depending on the location in which it is (to be) stored. Different commands are used for this purpose (see below).

Defining and Updating a Schema in Tamino

The **Define Schema** command is used for two purposes:

- To define one or more newly created schemas to Tamino.
- To update one or more existing Tamino schemas which you have modified.

Important: An existing Tamino schema is only overwritten with the new schema, if its doctype declaration has the physical property `Update access` set to "true".

If you attempt to update a schema in such a way that existing instances would become invalid, Tamino rejects the update operation.

To define or update one or more schemas in Tamino

1 From the **Database** menu, choose **Define Schema**.

Or:

Choose the following toolbar button:

2 The **Define Group of Schemas** dialog opens. This dialog allows you to select which of the currently open schemas you wish to define or update.

Mark the checkbox of each schema that you wish to define or update.

If you are updating a schema that is already defined in Tamino, you can check that the previously stored document instances that conform to this schema are valid against the updated schema. Do this by checking the box **With instance validation**.

3 Choose the **Define** button.

The **Define Schema** dialog box appears.

If this is the first time you are connecting to a server, the list box is empty. In this case, you have to proceed as described under *Connecting to Tamino*. This section also provides information on the buttons that are located next to the **Look in** drop-down list box.

4 Choose the required server.

The databases on this server are now shown.

5 Select the database in which you want to define or update the schema or schemas.

6 Choose the **Define** button.

If the database is not yet connected, the **Connect** dialog box appears and you must proceed as described under *Connecting to Tamino*.

If you are updating a schema or schemas, a dialog box appears, asking whether you want to update the existing schema or schemas. To update, choose the **Yes** button.

Successful definition or update of the schema or schemas is reported with a message in the **output panel**.

If a schema is invalid, any errors in the schema are listed in the output panel. In this case, the schema is not defined/updated in Tamino.

While the schema is defined to Tamino, it is possible to cancel the define process and stop processing of the schema. Choose the **Cancel** button to stop schema definition. A message is displayed, confirming the action.

Saving a Schema in the File System

You can save a schema from the Tamino Schema Editor to your file system.

To save the current schema in the file system (under a new name)

1 From the **File** menu, choose **Save As**.

The **Save As** dialog box appears.

2 Browse to the drive and folder in which the schema is to be saved and specify a file name.

3 Select the file type with which the schema is to be saved.

For a Tamino schema, the file type is "tsd".

4 Select the encoding with which the schema is to be saved.

The encoding is written to the XML declaration of the schema file. The default encoding is UTF-8.

Note: The **Encoding** drop-down list box is editable.

5 Choose the **Save** button.

The schema is written to the file system.

To save the modifications to the current schema

This command is only available if the schema has previously been saved in the file system.

- From the **File** menu, choose **Save**.

  Or:

  Press CTRL+S.

  Or:

  Choose the following toolbar button:

The modifications are written to the existing file in the file system.

Getting and Opening Schemas

There are different ways for loading a schema into the Tamino Schema Editor, depending on the location in which it is stored. Different commands are used for this purpose (see below).

The following topics are covered below:

Getting a Schema that is Stored in Tamino

You can load an existing Tamino schema into the Tamino Schema Editor.

To get an existing schema from Tamino

1 From the **Database** menu, choose **Get Schema**.

Or:

Choose the following toolbar button:

The **Get Schema** dialog box appears.

If this is the first time you are connecting to a server, the list box is empty. In this case, you have to proceed as described under *Connecting to Tamino*. This section also provides information on the buttons that are located next to the **Look in** drop-down list box.

2 Choose the required server.

The databases on this server are now shown.

3 Choose the database from which you want to get the schema.

If the database is not yet connected, the **Connect** dialog box appears and you must proceed as described under *Connecting to Tamino*.

When the connection to the database has been established, all collections of this database are shown in the **Get Schema** dialog box. If the list of collections is not visible, expand the database structure by, for example, double-clicking on the name of the database.

4 Choose the collection containing the required schema.

The schemas in this collection are now shown.

5 Select the required schema.

6 Choose the **Get** button.

Successful loading of the schema is reported with a message in the **output panel**.

Opening a Schema that is Stored the File System

You can open and modify a schema file that is stored in your file system. When you open a schema, it is always shown in a view which is able to display the whole schema. See also: *Switching to Another View*. In the dialog for opening a schema, you can choose to open several schemas at the same time.

The most recently opened files are always listed at the bottom of the **File** menu. Instead of using the **Open** command, you can open one of these recent files directly by choosing the corresponding item from the list.

To open one or more existing schemas from the file system

1 From the **File** menu, choose **Open**.

Or:

Press CTRL+O.

Or:

Choose the following toolbar button:

When the modifications to the current schema have not yet been modified, a dialog box appears, asking whether you want to save the currently displayed schema. To save the schema, choose the **Yes** button.

The **Open** dialog box appears.

2 Browse to the drive and folder containing the required schema or schemas.

Select the schema you wish to open. You can also select multiple schemas.

3 Choose the **Open** button.

To open one of the most recently opened files

- From the bottom of the **File** menu, choose the item which reflects the path to the desired file.

When the modifications to the current schema have not yet been modified, a dialog box appears, asking whether you want to save the currently displayed schema. To save the schema, choose the **Yes** button.

Undefining a Schema

When you undefine a schema, it is deleted from the Tamino database in which is has been defined. All instances related to the schema are also removed. A new schema with the same name can then be defined in Tamino.

The **Undefine Schema** command does not automatically undefine the schema that is currently shown in the schema editor. You first have to select the schema to be undefined from a dialog box. This may be the schema that is currently displayed. But it can also be any other schema.

To undefine a schema in Tamino

1 From the **Database** menu, choose **Undefine Schema**. The submenu items of **Undefine Schema** allow you to choose between undefining a single schema and undefining multiple schemas.

The remainder of this description covers the case of undefining a single schema; the dialog for undefining multiple schemas follows the same principle.

If this is the first time you are connecting to a server, the list box for the **Undefine Schema** dialog is empty. In this case, you have to proceed as described under *Connecting to Tamino*. This section also provides information on the buttons that are located next to the **Look in** drop-down list box.

2 Choose the required server.

The databases on this server are now shown.

3 Choose the required database.

If the database is not yet connected, the **Connect** dialog box appears and you must proceed as described under *Connecting to Tamino*.

When the connection to the database has been established, all collections of this database are shown in the **Undefine Schema** dialog box.

4 Choose the required collection.

All schemas in this collection are now shown.

5 Select the schema that is to be undefined.

6 Choose the **Undefine** button.

You are now asked whether you really want to undefine the schema and thus remove all related instances.

7 Choose the **Yes** button to undefine the schema.

Successful undefine of the schema is reported with a message in the **output panel**.

6 Importing DTDs, TSD2 Schemas and SQL Schemas

This chapter covers the following topics:

Importing a DTD

You can import a DTD and thus convert it to a Tamino schema.

The imported DTD is written to the schema which is currently shown in the Tamino Schema Editor. This may be either an existing schema or an empty schema that you have just created using the **New** command. In the latter case, the DTD name is automatically used as the schema name.

Since a Tamino schema can contain several doctypes, you can read multiple DTDs into the same schema.

A DTD file, *patient.dtd*, is provided for testing purposes. See *Examples*, *The Patient Database* in the Tamino XML Server documentation for further information.

Note: The DTD-to-TSD conversion can also be run using a command line tool. See ***Command Line Tools for Schema Conversions***.

To import a DTD

1 Make sure that the required schema (either an empty schema or an existing schema) is shown in the Tamino Schema Editor.

2 From the **File** menu, choose **Import DTD**.

The **Import DTD** dialog box appears.

3 Browse to the drive and folder containing the required DTD and select the DTD.

4 Choose the **Import** button.

The DTD is imported and shown in the schema tree. If the DTD is imported into an existing schema, it is inserted at the end of the schema tree.

The converter scans the DTD, declares the DTD elements globally, and then builds the schema structure, generating references to the global elements.

To inspect the new schema, you can switch to code view. You will see that the converter has generated a skeleton schema that conforms to the XML Schema (logical properties) below the doctype declaration, that is, without any physical properties.

If an element cannot be converted (for example, an entity), the converter writes a comment into the code.

Caution: If you wish instances of the new Tamino schema to conform to the original DTD, you should not change any of the logical properties `name`, `Minimum occurrence`, `Maximum occurrence` or `Mixed content`.

5 Select the nodes that you want to map to the Tamino schema and define their physical properties.

For a description of all available properties, see *Properties Explained*.

Tip: When an element declaration or element reference is selected in the schema tree, you can use the following commands from the **Edit** menu: **Go to Declaration** and **Go to Reference**.

6 Declare the Tamino doctype. To do so, select the schema node and from the **Insert** menu, choose **doctype**.

A new doctype with the name "NEW_doctype" is now shown in the tree.

7 Specify a name for the new doctype. It must be the same name as that of the top-level DTD element.

8 Define/update the current schema in Tamino and/or save it in your file system.

See *Defining and Saving Schemas*.

Importing a TSD2 Schema

You can import a TSD2 schema and thus convert it to the schema definition language which is supported by the current version of Tamino. See also *Migration Guide* in the Tamino XML Server documentation.

The imported TSD2 schema is written to the schema which is currently shown in the Tamino Schema Editor. This may be either an existing schema or an empty schema that you have just created using the **New** command.

Note: The DTD-to-TSD conversion can also be run using a command line tool. See *Command Line Tools for Schema Conversions*.

To import a TSD2 schema

1 Make sure that the required schema (either an empty schema or an existing schema) is shown in the Tamino Schema Editor.

2 From the **File** menu, choose **Import TSD2 Schema**.

The **Import TSD2 Schema** dialog box appears.

3 Browse to the drive and folder containing the required TSD2 schema and select the schema.

Note: TSD2 schema files have the extension "xml".

4 Select the encoding with which the imported schema is to be saved.

The encoding is written to the XML declaration of the schema file. The default encoding is "platform".

Note: The **Encoding** drop-down list box is editable.

5 Select the conversion mode: loose or strict.

Use the default mode (loose) unless you know for sure that your instances will validate against the Tamino schema generated in strict mode. In other words, if the TSD2 schema originates from a DTD and the instances validate against that DTD, convert using strict mode. In all other cases, use loose mode.

6 Choose the **Import** button.

The TSD2 schema is imported and shown in the schema tree. If the TSD2 schema is imported into an existing schema, it is inserted at the end of the schema tree.

7 Define/update the current schema in Tamino and/or save it in your file system.

See *Defining and Saving Schemas*.

Importing an SQL Schema Using the ODBC Wizard

You can import a schema from an external data source and thus convert it to a Tamino schema or schema sub-tree. The ODBC Wizard guides you in mapping SQL tables to a Tamino schema.

The imported SQL schema is written to the schema which is currently shown in the Tamino Schema Editor. This may be either an existing schema or an empty schema that you have just created using the **New** command.

Important: The ODBC Wizard can only be used if the appropriate ODBC driver for the external data source has been installed.

To import an SQL schema using the ODBC Wizard

1 Make sure that the required schema (either an empty schema or an existing schema) is shown in the Tamino Schema Editor.

2 If you want to import an SQL schema into an existing schema, select the element below which the SQL schema is to be inserted.

3 From the **Tools** menu, choose **ODBC Wizard**.

This command is only available, if the current selection in the tree allows the insertion of an SQL schema.

The first dialog of the ODBC Wizard appears.

4 Enter the ODBC DSN (Data Source Name) in the appropriate text box.

5 If your database requires security information, enter your user ID and password in the appropriate text boxes.

6 Choose the **Next** button.

The next dialog of the ODBC Wizard appears, providing a tree view of the database and its tables.

7 Expand the nodes in the tree and select the table you want to map.

8 Choose the **Next** button.

The next dialog of the ODBC Wizard appears, listing each column of the previously selected table with its name and data type.

9 Select the check box for the column that is used as the primary key.

The primary key may already be selected. This depends on the ODBC driver that you are using.

10 Choose the **Next** button.

The next dialog of the ODBC Wizard appears, listing each column of the table with its name and data type. It also shows the previously selected primary key.

11 Select the check box for each column that you want to use for generating the sub-tree.

You can also use the **Select All** or **Deselect All** button.

The primary key cannot be changed in this dialog.

12 Choose the **Finish** button.

The sub-tree is generated at the position that has previously been selected in the schema tree.

You can now view the properties of the schema and of each node. For a description of all available properties, see ***Properties Explained***.

Note: For further information on mapping to SQL tables and columns, refer to *Mapping to SQL Tables and Columns* in the *Tamino XML Schema User Guide* (located in the Tamino XML Server documentation).

7 Importing Adabas

You can import Adabas files and thus convert them to a Tamino schema or schema sub-tree. The Adabas Wizard helps you in mapping Adabas data to a Tamino schema.

The imported Adabas file meta-data is embedded in the schema which is currently shown in the Tamino Schema Editor. This may be an existing schema or an empty schema that you have just created using the **New** command.

Important: The Adabas database needs to be active, and the Entire Network server must be started. For detailed information about Tamino configuration to access Adabas, see the chapter *X-Node Access to Adabas* in *X-Node: Mapping to External Databases* of the Tamino Server documentation.

To import Adabas meta-data using the Adabas Wizard

1 Make sure that the required schema (either an empty schema or an existing schema) is shown in the Tamino Schema Editor.

2 If you want to import Adabas data into an existing schema, select the element below which the Adabas data is to be inserted.

3 From the **Tools** menu, choose **Import from Adabas**.

This command is only available, if the current selection in the tree allows the insertion of Adabas meta-data.

The dialog for the Adabas Wizard is displayed.

4 To import data directly from an Adabas database, enter the database identification and the number of the file in the appropriate text box. If you want to import from a corresponding Natural DDM File, additionally enter the file name in the **Natural DDM File** text box. The Natural file must reside on your computer. You can also select the Natural DDM File from the file chooser dialog box.

5 Choose the **Next** button.

The next dialog of the Adabas Wizard appears, providing the Adabas columns for generating the tree in the Schema Editor. If you have not entered a Natural DDM file name, only the short names of the Adabas fields are displayed as column names.

If you have entered a Natural DDM file name, the full column names appear.

6 Select the check box for each column that you want to use for generating the sub-tree.

You can also use the **Select All** or **Deselect All** button.

7 Choose the **Finish** button.

The schema is generated.

You can now view the properties of the schema and of each node. In the Physical Properties pane, for example, you can see that the storage type is XML and that a pure X-Node mapping has been performed. For a description of all available properties, see ***Properties Explained***.

8 Editing a Schema

When a schema has been loaded into the Tamino Schema Editor, you can modify it. For example, you can select a node in the schema tree and then change the properties of the selected node. Or you can invoke a context menu for the selected node and then choose the desired command.

This chapter covers the following topics:

Inserting an Element in the Schema Tree

You can use the commands from the **Insert** menu to insert structures for the logical properties of a schema. The commands vary, depending on the current selection in the schema tree. For example, for attribute declarations, only annotations can be inserted.

The **Insert** menu is not available in code view.

The Tamino Schema Editor displays the Tamino schema in such a way that the nodes of the tree coincide with the elements or attributes of the instance as much as possible. For this reason, there may be nodes which correspond to individual schema constructs as well as nodes that do not. These nodes are either composed of several schema constructs, or are determined by an attribute of a schema construct. Examples:

- Types and elements are compound nodes where one property determines the construct that is to be generated. Thus, a simple type can be defined using a restriction, list or union.
- The definition of a group results either in a group or group reference, depending on the setting of the `Reference` property.

When you load a schema into the Tamino Schema Editor or when you switch from code view to tree view, the Tamino Schema Editor determines each type of node on the basis of the schema constructs. It may happen that the type of an element cannot be determined and that the type "element unknown" is therefore used. This occurs when a type is referenced using the `Datatype` property and the Tamino Schema Editor does not recognize the declaration for this type. In this case, the Tamino Schema Editor cannot determine whether a simple element, an element with attributes or a complex element is referenced.

To insert an element in the schema tree

1 If code view is currently active, switch to one of the tree views.

2 In the schema tree, select the element below which you want to insert a new element.

3 From the **Insert** menu, choose the command which corresponds to the desired schema construct.

For a description of all available commands and corresponding schema constructs, see *Schema Tree Items Explained*.

4 Specify all required properties for the element you have just inserted.

For a description of all available properties, see ***Properties Explained***.

Editing Properties

When you edit the logical and physical properties of a schema construct, different types of input fields are available in the **Value** column:

- Simple text box in which you enter text.
- Drop-down list box from which you select one of the allowed values. The drop-down arrow is only visible after the input field has been selected.
- A button for invoking a property editor in which you specify the required information (see below). The button is only visible after the input field has been selected.

Using the Property Editor

Some properties are specified using a property editor (for example, the logical property `Enumeration` which can be found in the `Facets` group tree).

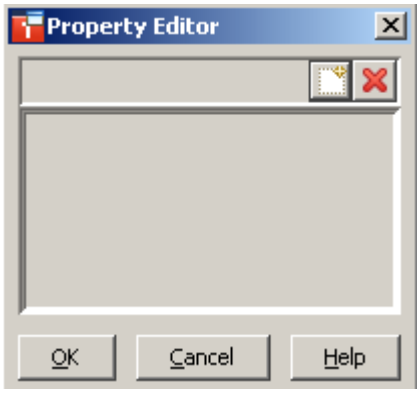
Note: Different types of property editor are available. See below.

To invoke the property editor

1 Select the input field in the **Value** column.

2 If provided, choose the following button:

The **Property Editor** dialog box appears. For the `Enumeration` property, it looks as follows:



The following buttons are provided:

 Add a new item.

 Delete the selected item.

3 Add/delete all required items.

4 Choose the **OK** button.

The dialog box is closed and the information you have specified is shown in the **Value** column.

Namespaces

A property editor is provided for the logical property `namespaces`. This property is available for **schemas** and **doctypes**. For information on how to invoke a property editor and how to add items, see ***Using the Property Editor***.

When you add a namespace, you have to specify a prefix and an URI. If a namespace is to be used as the target namespace, make sure that the corresponding option button in the **Target** column for the namespace is selected. It is only possible to use one or no target namespace.

Server Extensions

A property editor is provided for:

- the logical properties in the `triggers` (see ***Logical Properties for XML Elements***), and
- the physical properties of the storage type `Map XTension` (see ***Properties for SXS Mapping***).

For information on how to invoke a property editor, see ***Using the Property Editor***.

To select a server extension

1 If this is the first time you are connecting to a server, the list box which usually contains a list of servers is empty. In this case, you have to proceed as described under ***Connecting to Tamino***. This section also provides information on the buttons that are located next to the **Look in** drop-down list box.

2 Choose the required server.

The databases on this server are now shown.

3 Choose the database on which the required server extension has been installed.

The server extensions are now shown.

4 Select the server extension.

5 Choose the **OK** button.

The name of the server extension is now shown in the **Value** column.

Documenting a Schema

You can document a schema by inserting annotations at any level within the schema. Within an annotation, the documentation element is used to provide the documentation text. The documentation text is added using a property editor.

To add annotation and documentation elements

1 In the schema tree, select the node to which you want to add documentation.

2 From the **Insert** menu, choose **Annotation**.

This inserts an annotation as a child of the selected node. You can optionally assign an `id` value to it.

3 In the schema tree, select the annotation element.

4 From the **Insert** menu, choose **Documentation**.

This inserts a documentation element as a child of the annotation. The documentation element is automatically selected and its logical properties are shown.

5 In the logical properties, select the **Value** column for the property `Documentation text`.

6 Choose the following button.

The **Property Editor** dialog box appears.

7 Enter your annotation text.

To start text on a new line, simply press ENTER.

Using the commands from the **Edit** menu, you can cut, copy, paste, find and replace text. Further information on these features is provided later in this section.

8 Choose the **OK** button.

The dialog box is closed and the information you have specified is shown in the **Value** column.

Specifying the Occurrence Constraints

You can define occurrence constraints for an element.

Exception: occurrence constraints cannot be defined for top-level elements.

In the schema tree, one of the following characters may appear to the right of an icon. These characters indicate the occurrence constraints.

Character	Description	Values	
		Minimum occurrence	Maximum occurrence
?	Optional. Can only occur once.	0	1
1	Must occur exactly once.	1	1
*	Optional. Can be any number.	0	unbounded
+	Must occur at least once. Can be any number.	1	unbounded
n	Fixed values for Minimum occurrence and Maximum occurrence.	*n*	*m*

See the *XML Schema Part 0: Primer*, section 2.2.1 at ***http://www.w3.org/TR/xmlschema-0/#OccurrenceConstraints*** for more information on occurrence constraints.

To specify the occurrence constraints

1 In the schema tree, select the element for which you want to specify the occurrence constraints.

2 In the pane displaying the logical properties, select the value for the property `Minimum occurrence` or `Maximum occurrence` and select the required value from the drop-down list box.

In the schema tree, a character indicating the new constraints is now shown next to the corresponding icon.

Declaring an Element as Optional, Required or Prohibited

Elements can be declared as optional, required or prohibited.

Exception: top-level elements cannot be declared as optional, required or prohibited.

In the schema tree, one of the following characters may appear to the right of an icon:

Character	Value
?	Optional
1	Required
0	Prohibited

See the *XML Schema Part 0: Primer*, section 2.2.1 at ***http://www.w3.org/TR/xmlschema-0/#OccurrenceConstraints*** for more information.

To declare an element as optional, required or prohibited

1 In the schema tree, select the desired element.

2 In the pane displaying the logical properties, select the value for the property `use` and select the required value from the drop-down list box.

In the schema tree, one of the above characters is now shown next to the corresponding icon.

Displaying the Declaration for a Reference

You can immediately jump to the declaration for the selected reference.

Note: This feature is not available in code view.

To display the declaration for a reference

1 In the schema tree, select the desired reference.

2 From the **Edit** menu, choose **Go to Declaration**.

Or:

Press CTRL+ALT+PAGE-UP.

Or:

Choose the following toolbar button:

The declaration is now selected in the schema tree.

Displaying the Next Reference

You can immediately jump to the next reference for the selected declaration, or to the next reference that matches the selected reference.

Note: This feature is not available in code view.

To display the next reference

1 In the schema tree, select either a declaration or reference.

2 From the **Edit** menu, choose **Go to Next Reference**.

Or:

Press CTRL+ALT+PAGE-DOWN.

Or:

Choose the following toolbar button:

The next reference is now selected in the schema tree.

Moving an Element Up and Down in the Schema Tree

You can change the sequence of elements in the schema tree. You can move an element up or down, including all of its child nodes, within the allowed hierarchy in the schema.

Note: This feature is not available in code view.

To move up the selected element

- From the **Edit** menu, choose **Move Up**.

  Or:

  Press CTRL+ALT+UP-ARROW.

  Or:

  Choose the following toolbar button:

To move down the selected element

- From the **Edit** menu, choose **Move Down**.

  Or:

  Press CTRL+ALT+DOWN-ARROW.

  Or:

  Choose the following toolbar button:

Cutting, Copying and Pasting Information

You can cut, copy and paste the following:

- A node in the schema tree, including all child nodes. A node is treated as a fragmental XML document.
- A logical or physical property value where direct text input is possible (for example, a name or ID).
- Text in code view or in a property editor.

Caution: This feature uses the Windows clipboard which may also contain information from another application. Due to performance reasons, the information from the clipboard is not parsed. Watch the output panel for messages concerning your last action. When you cut information and paste it at a position in the tree where it is not allowed, this information will be lost. A message will be shown in the output panel. The drag-and-drop feature, which is described **below**, provides more intelligent support for copying and moving information.

It is also possible to copy an entire message line from the output panel, for example, to another application.

Note: The cut-and-paste feature cannot be used to change the sequence of the nodes within the schema tree. If you want to change the sequence, see *Moving an Element Up and Down in the Schema Tree*.

To cut the selection and transfer it to the clipboard

- From the **Edit** menu, choose **Cut**.

  Or:

  Press CTRL+X.

  Or:

  Choose the following toolbar button:

  The selection is deleted at its original position and is copied to the clipboard.

To copy the selection to the clipboard

- From the **Edit** menu, choose **Copy**.

  Or:

  Press CTRL+C.

  Or:

  Choose the following toolbar button:

  The selection remains at its original position and is copied to the clipboard.

To paste the content of the clipboard

1. In tree view, select the node below which you want to paste the information from the clipboard.

   In tree view, it is only possible to paste information at a position which syntactically fits into the schema definition.

   Or:

   In the logical or physical properties, select the value field to which you want to paste the previously cut or copied information. If required, move the insertion point to the required position.

   Or:

   In code view, place the insertion point at the position at which you want to paste the previously cut or copied information.
2. From the **Edit** menu, choose **Paste**.

   Or:

   Press CTRL+V.

   Or:

   Choose the following toolbar button:

In tree view, the Tamino Schema Editor tries to insert the previously cut or copied node below the selected node. If single nodes cannot be inserted (because they are not allowed at this point of the schema or because the Tamino Schema Editor cannot handle them), they will be lost and a warning will be written to the output panel.

In the logical or physical properties, the previously cut or copied information is inserted in the selected value field.

In code view, the previously cut or copied information is inserted at the insertion point. There is no immediate validity check. However, any errors are reported in the output panel when you try to switch to one of the code views.

Using Drag-and-Drop

You can copy or move information in the schema tree using drag-and-drop. Since drag-and-drop can only be used within the schema tree, the origin of a node to be pasted is known and can be parsed prior to insertion. Thus, it is only possible to drop a node at a position in the tree where it is allowed. As long as you are dragging a node, the mouse pointer always indicates whether a node can be dropped or not.

Caution: Only the child nodes on the first level are parsed. When the first level is allowed at a position in the schema tree, the node can be dropped. However, if there are further child nodes on the second level or lower which cannot be inserted at that position (because they are not allowed at this point of the schema or because the Tamino Schema Editor cannot handle them), they will be lost and a warning will be written to the output panel. You can undo the drag-and-drop action immediately. See *Undoing and Redoing the Previous Action*.

As long as you are dragging a node, you can use the following features:

- Place the mouse pointer briefly over a node which is preceded by a plus sign. This expands the node so that you can examine its contents.
- Place the mouse pointer briefly over the upper or lower border of the pane containing the schema tree. This scrolls the top or bottom of the schema tree into view.

Note: The drag-and-drop feature cannot be used to change the sequence of the nodes within the schema tree. If you want to change the sequence, see *Moving an Element Up and Down in the Schema Tree*.

To copy a node to another position in the tree

1 In tree view, click the node to be copied and hold down the mouse button.

2 Hold down CTRL.

3 Drag the node to another position in the tree at which it can be dropped.

When the node can be dropped, the mouse pointer shows a rectangle containing a plus sign.

4 Release the mouse button and then CTRL.

The node is copied to the new position.

To move a node to another position in the tree

1 In tree view, click the node to be moved and hold down the mouse button. Do not press any key.

2 Drag the node to another position in the tree at which it can be dropped.

When the node can be dropped, the mouse pointer shows an empty rectangle.

3 Release the mouse button.

The node is moved to the new position.

Copying the Path to the Clipboard

You can copy the path for the currently selected element or attribute to the clipboard. It is copied as an XPath expression. This is helpful, for example, if you want to use the path in the physical properties of a node that can be accessed via multiple paths (see *Advanced Physical Properties*), or if you want to copy the path to another application.

The format of the path depends on the setting of the `form` property for the selected element or attribute:

- When the `form` property is set to "qualified", the prefix from the **target namespace** definition is used.

  Example for an element:

  ```
  /myPrefix:Telephone/myPrefix:Lastname
  ```

  Example for an attribute:

  ```
  /myPrefix:Telephone/@myPrefix:EntryID
  ```

- When the `form` property is set to "unqualified", a prefix is not used.

  Example for an element:

  ```
  /myPrefix:Telephone/Lastname
  ```

  Example for an attribute:

  ```
  /myPrefix:Telephone/@EntryID
  ```

- When a value has not been defined for the `form` property, the value ("qualified" or "unqualified") defined for one of the following **schema properties** is used:
  - `elementFormDefault` in the case of an element.
  - `attributeFormDefault` in the case of an attribute.

To copy the path to the clipboard

1 In tree view, select the desired element or attribute.

2 From the **Edit** menu, choose **Copy Path**.

The path is copied to the clipboard. You can now use the **Paste** command to insert the path at the desired position.

Deleting Information

You can delete the following:

- A node in the schema tree, including all child nodes.
- Text in code view.

To delete the selection

- From the **Edit** menu, choose **Delete**.

  Or:

  Press DEL.

  Or:

  Choose the following toolbar button:

  The selection is deleted immediately. You are not asked to confirm the deletion.

  However, when you delete a referenced element, group or attribute group in tree view, a dialog box appears, asking whether you also want to delete the references to the deleted node. When you choose the **Yes** button, you confirm the deletion.

  When you delete a type, all schema elements using the deleted type are listed in the **output panel**. Hyperlinks are provided. When you choose a hyperlink, the node which uses the deleted type is selected in the schema tree. It is your responsibility to define an existing type.

Renaming an Item in the Schema Tree

You can rename elements, types, groups and attribute groups in the schema tree. When an item cannot be renamed (for example, an element reference), the **Rename** command is not available.

To rename an element in the schema tree

1. Select the tree element that you want to rename.
2. From the **Edit** menu, choose **Rename**.

   Or:

In the pane displaying the logical properties, select the value for the property `name`.

Or:

In the context menu of the selected tree element, choose **Rename**.

3 Specify the new name.

4 Press ENTER or click any other position in the application window.

When you rename an element, type, group or attribute group, a dialog box appears for each node in the schema tree which uses the renamed item, and you are asked whether you want to adapt the corresponding property for this node. When you choose the **Yes** button, the property value is renamed.

Finding and Replacing Information

You can find and replace a node, property or string within the current schema.

To find information

1 In the schema tree, select the node that is to be the starting point for your search.

Or:

In code view, place the insertion point at the position that is to be the starting point for your search.

2 From the **Edit** menu, choose **Find**.

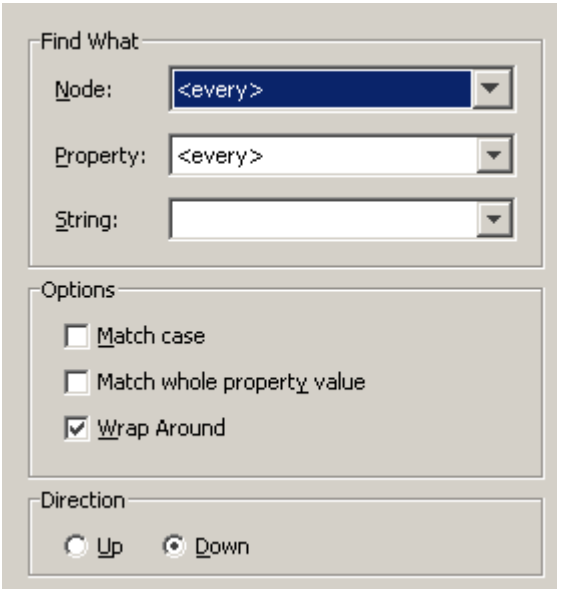
Or:

Press CTRL+F.

Or:

Choose the following toolbar button:

The **Find** dialog box appears. The information that can be specified in the dialog box depends on the view in which it has been invoked. The **Find** dialog box appears in tree view.



Note: When you invoke this dialog box in code view or from a property editor, the drop-down list boxes **Node** and **Property** and the check box **Match whole property value** are not shown.

3 Specify all required search criteria:

Node
Not available in code view.

This drop-down list box provides for selection all elements that may occur in a schema (for example, a simple element or annotation).

Property
Not available in code view.

This drop-down list box provides for selection all properties that are valid for the selected node (for example, a name or ID). For information concerning the possible entries, see *Properties Explained*.

String
This drop-down list box provides for selection the search strings that you have previously entered. You can also enter a new string.

Match case
When this check box is selected, the search is case-sensitive.

Match whole property value
Not available in code view.

When this check box is selected, the search is restricted to matching whole values only. This is useful for searching for "a" or "A" without finding all words including "a".

Wrap around
When this check box is selected, the complete schema is searched, even if your starting point for the search is in the middle of the schema.

Direction
Select the option button **Up** or **Down** to search the schema in the desired direction.

4 Choose the **Find Next** button.

If the specified information is found in tree view, the node containing this information is expanded and the found element is selected.

If the specified information is found in code view, it is selected.

5 To find the next occurrence, choose the **Find Next** button once more.

Or:

Find the next or previous occurrence as described below; see *Finding the Next or Previous Occurrence*.

6 To close the dialog box, choose the **Close** button.

To replace information

1 In the schema tree, select the node that is to be the starting point for your search.

Or:

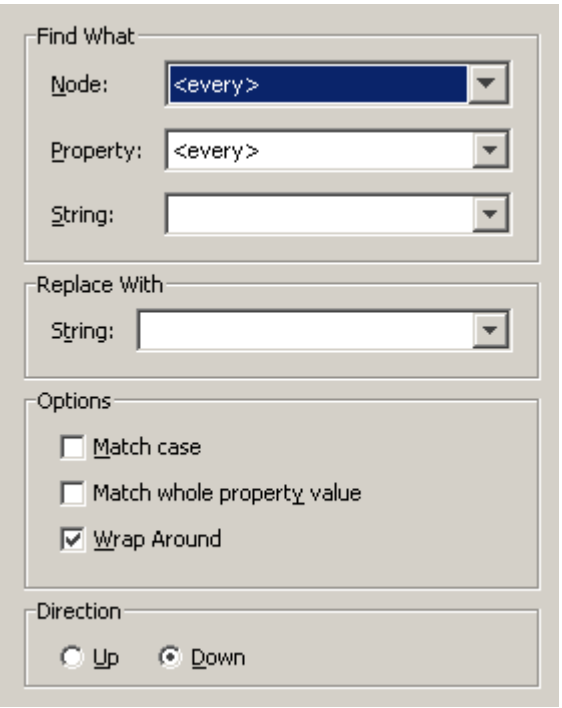
In code view, place the insertion point at the position that is to be the starting point for your search.

2 From the **Edit** menu, choose **Replace**.

Or:

Press CTRL+R.

The **Replace** dialog box appears. The information that can be specified in the dialog box depends on the view in which it has been invoked. The **Replace** dialog box appears in tree view.



This dialog box contains the same options as the **Find** dialog box. It contains an additional drop-down list box in which you enter the replace string, and it contains additional command buttons.

Note: When you invoke this dialog box in code view or from a property editor, the drop-down list boxes **Node** and **Property** and the check box **Match whole property value** are not shown.

3 Specify all required search criteria. See the above description of the **Find** dialog box for detailed information.

4 Specify the string with which a found occurrence is to be replaced.

5 To replace all occurrences that match your search criteria at once, choose the **Replace All** button.

If **Wrap around** has not been specified, only the occurrences between the current position and the end or start of the schema (depending on the specified direction) are replaced.

Note: The **Replace All** button only replaces those occurrences that are allowed to be replaced with the current replace string. Strings that have not been replaced are reported in the **output panel**.

Or:

To replace one occurrence after the other, choose the **Find Next** button. This selects the next occurrence and you can now choose the **Replace** button, or the **Find Next** button if you do not want to replace the current occurrence.

Note: The **Find Next** button only finds those occurrences that are allowed to be replaced with the current replace string.

6 To close the dialog box, choose the **Close** button.

Finding the Next or Previous Occurrence

The commands for finding the next or previous occurrence always use the information that has previously been entered in the **Find** or **Replace** dialog box (see *Finding and Replacing Information*). This is helpful, if these dialog boxes have been closed.

You can use these commands in both tree view and code view. If the occurrence is found in tree view, the node containing this occurrence is expanded and the occurrence is selected. If the occurrence is found in code view, it is selected.

To find the next occurrence

- From the **Edit** menu, choose **Find Next**.

  Or:

  Press F3.

To find the previous occurrence

- From the **Edit** menu, choose **Find Previous**.

  Or:

  Press SHIFT+F3.

Undoing and Redoing the Previous Action

You can undo and redo your previous actions in the current view.

Caution: The undo/redo history is cleared when you switch to another view.

To undo the previous action

- From the **Edit** menu, choose **Undo**.

  Or:

  Press CTRL+Z.

  Or:

  Choose the following toolbar button:

To redo the previous undo action

- From the **Edit** menu, choose **Redo**.

  Or:

  Press CTRL+Y.

  Or:

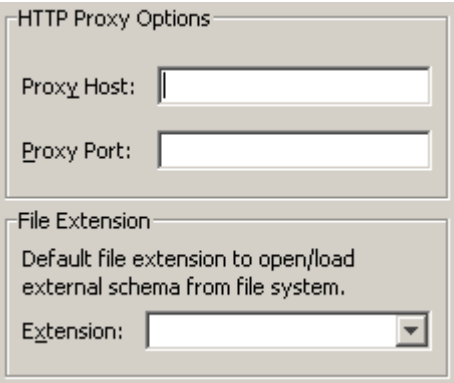
  Choose the following toolbar button:

Defining the Options

You can define the options for external schemas.

To open the **Options** dialog, choose **Tools > Options**



HTTP Proxy Options

The address (host name and port number) for your proxy server has to be defined in the following cases:

- When a referenced schema is to be opened in a new window via HTTP. See *Opening a Referenced Schema in a New Window*.
- When elements from an external schema are to be loaded via HTTP. See *Loading and Unloading the Elements from an External Schema*.

File Extension

When you import an external schema, Tamino requires that the schema location is relative, i.e. an extension must not be defined.

When a default file extension has been defined, an external schema for which a file extension has not been defined is automatically **opened** or **loaded** from the file system using the default file extension that has been defined in this dialog box.

Browsing for a Schema Location

When an Import, Include or Redefine node is available in the schema tree, you can define the disk location where the appropriate schema component is located. A file system browser is available for navigation to the required location. The location path is subsequently stored in the logical property `Schema location` of the Import, Include or Redefine node.

To browse for a schema location

1 Select the Import, Include or Redefine node in the schema tree.

2 Select the logical property `Schema location`.

3 Position the cursor in the **Value** field next to `Schema location`.

To the right of the **Value** field, a browse button () appears.

4 Select the **Browse** button. The dialog box **Browse for Base of Location** is displayed.

5 Browse to the schema file and choose the **Open** button.

The value of the logical property `Schema location` is inserted into the **Value** field of the Schema Editor.

Opening a Referenced Schema in a New Window

When an Import, Include or Redefine node is available in the schema tree and the logical property `Schema location` has been specified, you can open the referenced schema in a new instance of the Tamino Schema Editor.

Note: If you want to open the schema via HTTP, you have to define the address for your proxy server. You can also define a default file extension to be used when opening a schema from the file system. See *Defining the Options*.

When you modify one of the schemas, this has no effect on the other schema.

To open the referenced schema in a new instance of the Tamino Schema Editor

1 In the schema tree, select an Import, Include or Redefine node.

2 From the **Tools** menu, choose **External Schema > Open**.

Note: The command **Open** is only available, if a location has been specified with the property `Schema location`.

A cascading menu with the following commands is available when a relative path has been specified in the `Schema location` property:

- **From File System**
- **From Database**

When an absolute path to the file system or to a database has been specified, a cascading menu with the above commands is not available. In this case, the referenced schema is opened in the same way as the current schema: it is either loaded from the file system or from the database.

Loading and Unloading the Elements from an External Schema

When an Import, Include or Redefine node is available in the schema tree and the logical property `Schema location` has been specified, you can load the elements from the referenced external schema into the internal model, in addition to the current schema. The global elements or types of the external schema can then be selected from a drop-down list box in the logical properties (properties `Reference`, `Datatype`, `Base type`, `itemType` and `memberTypes`).

The external schema itself is not visible in the Tamino Schema Editor.

The Tamino Schema Editor uses the global types information of the external schema to represent the elements that use these types with the correct icon in the schema tree. In the case of `<xs:element type="userDefinedType"/>`, the editor can thus decide whether to show the element as "Element simple", "Element with attributes" or "Element complex". When an external schema has been loaded, each "Element unknown" for which the type is defined in the external schema is shown with the corresponding icon. When the external schema has been unloaded, these elements are shown again as "Element unknown".

The Tamino Schema Editor does not automatically receive updates to the external schema. If you want to work with the updated schema, you have to unload the schema and then load it once more.

Note: If you want to load the elements from an external schema via HTTP, you have to define the address for your proxy server. You can also define a default file extension to be used when loading a schema from the file system. See *Defining the Options*.

To load the elements from an external schema

1 In the schema tree, select an Import, Include or Redefine node.

2 From the **Tools** menu, choose **External Schema > Load**.

Note: The command **Load** is only available, if a location has been specified with the property `Schema location`.

A cascading menu with the following commands is available when a relative path has been specified in the `Schema location` property:

- **From File System**
- **From Database**

When an absolute path to the file system or to a database has been specified, a cascading menu with the above commands is not available. In this case, you can directly choose the **Load** command.

An external schema can only be loaded when it does not contain any errors.

When the elements of the external schema have been loaded, a small triangle is shown to the right of the icon for the Import, Include or Redefine node.

9 Transforming and Converting Schema Constructs

This chapter covers the following topics:

General Information

The Tamino Schema Editor allows you to perform transformations that facilitate the schema design process. See *Making a Schema Construct Local or Global* and *Converting a Choice, Sequence or All*

Using a Transformation Wizard

You can transform a schema construct into another schema construct, using items from the menu bar or using items from context menus.

The **Transformation** command in the **Tools** menu offers several transformation commands in a cascading menu. The available transformation commands depend on the current selection in the schema tree and on the current view.

When you select a transformation command, a wizard appears providing different dialogs. The number of dialogs and the amount of information in these dialogs depends on the elements that have been defined for the selected schema construct.

To use a transformation wizard

1 In the schema tree, select the schema construct that is to be transformed.

2 From the **Tools** menu, choose **Transformation >** *`transformation-command`*.

Descriptions of the transformation commands are provided later in this section.

A wizard appears, providing different dialogs.

Read the information in each dialog carefully. The wizard informs you about all items that will be lost or that will not be adopted. Some dialogs require that you select one or more options (for example, whether you want to copy the documentation/appinfo of a reference), or that you enter additional information (for example, when an element with the same name exists already or when you have to decide on a new parent for an annotation).

3 Use the **Next** button to proceed to the next dialog.

Or:

If no further dialogs are available, choose the **Finish** button to execute the transformation.

Note: You can undo each transformation immediately. See *Undoing and Redoing the Previous Action*.

Making a Schema Construct Local or Global

The following transformation commands are provided in the **Tools > Transformation** menu:

Make Local
Replaces the selected reference with its declaration.

Extract Global Type
Moves the selected element or attribute to the top level of the schema. A reference is created at the selected position.

If an element with the same name exists already at the top level of the schema, you have to enter a new name for the element that is to be moved to the top level.

If the selected element has an annotation, you have to specify the new parent of the annotation. This can either be a new global element (i.e. a declaration) or a new local element (i.e. a reference). It is also possible not to specify a new parent; in this case, the information will be lost.

Converting a Choice, Sequence or All

The following transformation commands are provided in the **Tools > Transformation > Convert To** menu:

All
Converts the selected Choice or Sequence to All. Information may be lost. Therefore, a wizard is used.

Choice
Converts the selected Sequence or All to a Choice. This is a simple conversion. Information is not lost and therefore a wizard is not used.

Sequence
Converts the selected Choice or All to a Sequence. This is a simple conversion. Information is not lost and therefore a wizard is not used.

Converting an Element

The following transformation commands are provided in the **Tools > Transformation > Convert To** menu:

Attribute
Converts the selected element, that currently has no attributes, into an attribute.

Element with Attribute
Converts the selected element, that currently has no attributes, into an element with an attribute.

10 Schema Tree Items Explained

This chapter describes all commands that can be chosen from the **Insert** menu:

Note: When not indicated otherwise, the commands are available in all views.

For the corresponding XML Schema constructs, references are made to the W3C site:

- *XML Schema Part 0: Primer* at ***http://www.w3.org/TR/xmlschema-0/***.
- *XML Schema Part 1: Structures* at ***http://www.w3.org/TR/xmlschema-1/***.
- *XML Schema Part 2: Datatypes* at ***http://www.w3.org/TR/xmlschema-2/***.

All

Groups elements within a complex element declaration.

Schema construct:

```
xs:all
```

The constraints are described in the ***XML Schema Part 0: Primer*, section 2.7**.

Annotation

Adds documentation or application-specific information to a declaration.

Schema construct:

```
xs:annotation
```

See ***XML Schema Part 0: Primer*, section 2.6**.

Any

Extends the current content model with well-formed XML, usually one or more elements belonging to a different namespace.

Schema construct:

```
xs:any
```

See ***XML Schema Part 0: Primer*, section 5.5**.

AnyAttribute

Extends the current content model using an attribute belonging to a different namespace.

Schema construct:

```
xs:anyAttribute
```

See ***XML Schema Part 0: Primer*, section 5.5**.

Appinfo

Adds application-specific information to a declaration.

Schema construct:

```
xs:appinfo
```

See ***XML Schema Part 0: Primer*, section 2.6**.

Attribute

Adds an attribute declaration to an element.

Different values can be defined for the property `Variety`:

Variety: type / restriction

- When a facet has *not* been set:

  Defines an attribute by referencing an existing simple type using the `Datatype` property.

  Available if the simple type is a predefined type.

  Only available if the simple type is a user-defined type.

  Schema construct:

  ```
  xs:attribute type="name"
  ```

  See ***XML Schema Part 0: Primer*, section 2.2**.

- When a facet has been set:

  Defines an attribute by restricting an existing simple type which is referenced by the `Datatype` property. This property is then mapped to the `Base type` attribute.

  Available if the simple type is a predefined type.

  Only available if the simple type is a user-defined type.

  Schema construct:

  ```
  xs:attribute xs:simpleType xs:restriction base ="name"
  ```

  See ***XML Schema Part 0: Primer*, section 2.2** and **section 2.3**.

Variety: restriction with local simpleType

Defines an attribute. The type is determined by the local simple type declaration.

Schema construct:

```
xs:attribute xs:simpleType xs:restriction
```

See *XML Schema Part 0: Primer*, **section 2.2** and *XML Schema Part 2: Datatypes*, **section 4.1.2**.

Variety: list

Defines an attribute. The type of the attribute is a list type. The type of the list values is determined by the `itemType` property. The `itemType` property may reference other simple types.

Schema construct:

```
xs:attribute xs:simpleType xs:list itemType="name"
```

See *XML Schema Part 0: Primer*, **section 2.2** and **section 2.3.1**.

Variety: list with local simpleType

Defines an attribute. The type of the attribute is a list type. The type of the list values is determined by the local simple type declaration.

Schema construct:

```
xs:attribute xs:simpleType xs:list
```

See *XML Schema Part 0: Primer*, **section 2.2** and *XML Schema Part 2: Datatypes*, **section 4.1.2**.

Variety: union

Defines an attribute. Using a union, the type of the attribute is composed of other types. These types are simple types which are referenced by the `memberTypes` property and/or are defined using local simple type declarations.

Schema construct:

```
xs:attribute xs:simpleType xs:union memberTypes="name name1"
```

See *XML Schema Part 0: Primer*, **section 2.2** and **section 2.3.2**.

AttributeGroup

Groups all attribute declarations in a global attribute group.

Schema construct:

```
xs:attributeGroup
```

See ***XML Schema Part 0: Primer*, section 2.8**

AttributeGroup reference

Adds an attributeGroup declaration by referring to an existing attributeGroup.

Schema construct:

```
xs:attributeGroup ref="name"
```

Attribute info

Defines physical properties for an attribute. The attribute is addressed by the path in the `Context` property.

Schema construct:

```
tsd:attributeInfo context = "path"
```

Attribute reference

Adds an attribute declaration by referring to an existing attribute.

Schema construct:

```
xs:attribute ref="name"
```

See *XML Schema Part 0: Primer,* **section 2.2**.

Choice

Declares a choice of elements: only one element within the choice can appear in an instance.

Schema construct:

```
xs:choice
```

See ***XML Schema Part 0: Primer*, section 2.7**.

ComplexType

A complex type may contain other element and attribute declarations.

Different values can be defined for the property `Derivation method`:

Derivation: new
Declares a new global complex type.

Schema construct:

```
xs:complexType name="name"
```

See ***XML Schema Part 0: Primer*, section 2.5** and **section 2.7**.

Derivation: extension
Defines a global complex type by extending an existing complex type which is referenced by the `Base type` property.

Schema construct:

```
xs:complexType xs:complexContent xs:extension base="name"
```

See ***XML Schema Part 0: Primer*, section 4.2**.

Derivation: restriction
Defines a global complex type by restricting an existing complex type which is referenced by the `Base type` property.

Schema construct:

```
xs:complexType xs:complexContent xs:restriction base="name"
```

See *XML Schema Part 1: Structures*, **section 3.4**.

Doctype

Specifies the doctype of the Tamino schema. One schema can contain multiple doctypes. The name must be the same as a global element in the schema.

Schema construct:

```
tsd:doctype
```

Documentation

Adds documentation information to a declaration.

Schema construct:

```
xs:documentation
```

See *XML Schema Part 0: Primer*, **section 2.6**.

Element complex

A complex element may contain other element and attribute declarations.

Different values can be defined for the property `Derivation method`:

Derivation: new
Declares a new complex element.

Schema construct:

```
xs:element xs:complexType
```

See *XML Schema Part 0: Primer*, section 2.5 and section 2.7.

Derivation: none

Declares a complex element by referencing an existing complex type.

Schema construct:

```
xs:element type="name"
```

See *XML Schema Part 0: Primer*, section 2.5.

Derivation: extension

Defines a complex element by extending an existing complex type which is referenced by the `Base type` property.

Schema construct:

```
xs:element xs:complexType xs:complexContent xs:extension base ="name"
```

See *XML Schema Part 0: Primer*, section 4.2.

Derivation: restriction

Defines a complex element by restricting an existing complex type which is referenced by the `Base type` property.

Schema construct:

```
xs:element xs:complexType xs:complexContent xs:restriction base ="name"
```

See *XML Schema Part 1: Structures*, section 3.4.

Element info

Defines physical properties for an element. The element is addressed by the path in the `Context` property.

Schema construct:

```
tsd:elementInfo context = "path"
```

Element reference

Adds an element declaration by referring to an existing element.

Schema construct:

```
xs:element ref="name"
```

See ***XML Schema Part 0: Primer*, section 2.2**.

Element simple

Adds a simple element declaration.

Different values can be defined for the property `Variety`:

Variety: type / restriction

- When a facet has *not* been set:

  Defines an element by referencing an existing simple type using the `Datatype` property.

  Available if the simple type is a predefined type.

  Only available if the simple type is a user-defined type.

  Schema construct:

```
xs:element type="name"
```

See *XML Schema Part 0: Primer*, section 2.2.

- When a facet has been set:

  Defines an element by restricting an existing simple type which is referenced by the `Datatype` property. This property is then mapped to the `Base type` attribute.

  Available if the simple type is a predefined type.

  Only available if the simple type is a user-defined type.

  Schema construct:

  ```
  xs:element xs:simpleType xs:restriction base ="name"
  ```

  See *XML Schema Part 0: Primer*, section 2.2 and section 2.3.

Variety: restriction with local simpleType

Defines an element. The type is determined by the local simple type declaration.

Schema construct:

```
xs:element xs:simpleType xs:restriction
```

See *XML Schema Part 0: Primer*, section 2.2 and *XML Schema Part 2: Datatypes*, section 4.1.2.

Variety: list

Defines an element. The type of the element is a list type. The type of the list values is determined by the `itemType` property. The `itemType` property may reference other simple types.

Schema construct:

```
xs:element xs:simpleType xs:list itemType="name"
```

See *XML Schema Part 0: Primer*, section 2.2 and section 2.3.1.

Variety: list with local simpleType

Defines an element. The type of the element is a list type. The type of the list values is determined by the local simple type declaration.

Schema construct:

```
xs:element xs:simpleType xs:list
```

See *XML Schema Part 0: Primer*, **section 2.2** and *XML Schema Part 2: Datatypes*, **section 4.1.2**.

Variety: union

Defines an element. Using a union, the type of the element is composed of other types. These types are simple types which are referenced by the `memberTypes` property and/or are defined using local simple type declarations.

Schema construct:

```
xs:element xs:simpleType xs:union memberTypes="name name1"
```

See *XML Schema Part 0: Primer*, **section 2.2** and **section 2.3.2**.

Element unknown

Adds an element declaration that references an existing type. This schema tree item is usually chosen if a user-defined type is referenced and the type is not known to the Schema Editor. Thus, the Schema Editor cannot determine whether the element is an element simple, element complex or element with attributes. If the declaration of the type is loaded into the Schema Editor, the tree item is updated.

Schema construct:

```
xs:element type = "name"
```

Element with attributes

An element with attributes may not contain other elements, but may carry attributes and may contain simple values.

Different values can be defined for the property `Derivation method`:

Derivation: none
Declares an element with attributes by referencing an existing element with attributes.

Schema construct:

```
xs:element type="name"
```

See ***XML Schema Part 0: Primer*, section 2.5**.

Derivation: extension
Defines an element with attributes by extending an existing simple type or simple type with attributes which is referenced by the `Base type` property.

Schema construct:

```
xs:element xs:complexType xs:simpleContent xs:extension base="name"
```

See ***XML Schema Part 1: Structures*, section 3.4**.

Derivation: restriction
Defines an element with attributes by restricting an existing simple type with attributes or complex type which is referenced by the `Base type` property. When a complex type is restricted, the type of the simple content must be determined by a local simple type declaration.

Schema construct:

```
xs:element xs:complexType xs:simpleContent xs:restriction base="name"
```

See ***XML Schema Part 1: Structures*, section 3.4**.

Group

Groups element declarations in a global group.

Schema construct:

```
xs:group
```

See *XML Schema Part 0: Primer*, **section 2.7**.

Group reference

Adds a group declaration by referring to an existing group.

Schema construct:

```
xs:group ref="name"
```

See *XML Schema Part 0: Primer*, **section 2.7**.

Import

References another schema that has another target namespace. Enables the schema components to be referenced by components of this schema.

Schema construct:

```
xs:import
```

See *XML Schema Part 0: Primer*, **section 5.4**.

Include

References another schema that has the same target namespace. Enables the schema components to be used by components of this schema without modifications. Big schemas may be distributed into several schema documents by this mechanism.

Schema construct:

```
xs:include
```

See ***XML Schema Part 0: Primer*, section 4.1**.

Key

Defines a key.

Schema construct:

```
xs:key xs:selector xs:field
```

See ***XML Schema Part 0: Primer*, section 5.2**.

Keyref

References a key.

Schema construct:

```
xs:keyref xs:selector xs:field
```

See ***XML Schema Part 0: Primer*, section 5.2**.

Notation

Adds notation declarations to the schema.

Schema construct:

```
xs:notation
```

See *XML Schema Part 1: Structures*, **section 3.12** on the W3C site.

Redefine

References another schema that has the same target namespace. Enables schema components to be used by components of this schema with or without modifications. Groups and types of the other schema may be modified by adding them with modifications below the Redefine node.

Schema construct:

```
xs:redefine
```

See *XML Schema Part 0: Primer*, **section 4.5**.

Schema

Schema root node. This node is always present and cannot be deleted. Therefore, a corresponding command is not available in the **Insert** menu.

Schema construct:

```
xs:schema
```

See *XML Schema Part1: Structures*, **section 3.15**.

Sequence

Declares a sequence of elements: elements within a sequence group must appear in an instance in that order.

Schema construct:

```
xs:sequence
```

See ***XML Schema Part 0: Primer*, section 2.7**.

SimpleType

Adds a simple type declaration.

Different values can be defined for the property `Variety`:

Variety: restriction
Defines a simple type by restricting an existing simple type which is referenced by the `Base type` attribute.

Available if the simple type is a predefined type.

Only available if the simple type is a user-defined type.

Schema construct:

```
xs:simpleType xs:restriction base ="name"
```

See ***XML Schema Part 0: Primer*, section 2.3**.

Variety: restriction with local simpleType
Defines a simple type. The type is determined by the local simple type declaration.

Schema construct:

```
xs:simpleType xs:restriction
```

See ***XML Schema Part 2: Datatypes*, section 4.1.2**.

Variety: list
Defines a simple type. The type is a list type. The type of the list values is determined by the `itemType` property. The `itemType` property may reference other simple types.

Schema construct:

```
xs:simpleType xs:list itemType="name"
```

See ***XML Schema Part 0: Primer*, section 2.3.1**.

Variety: list with local simpleType
Defines a simple type. The type is a list type. The type of the list values is determined by the local simple type declaration.

Schema construct:

```
xs:simpleType xs:list
```

See *XML Schema Part 2: Datatypes*, **section 4.1.2**.

Variety: union

Defines a simple type. Using a union, the type is composed of other types. These types are simple types which are referenced by the `memberTypes` property and/or are defined using local simple type declarations.

Schema construct:

```
xs:simpleType xs:union memberTypes="name name1"
```

See *XML Schema Part 0: Primer*, **section 2.3.2**.

SimpleType with attributes

A simple type with attributes may contain other element and attribute declarations.

Different values can be defined for the property `derivation`:

Derivation: extension
Defines a simple type with attributes by extending an existing simple type or simple type with attributes which is referenced by the `Base type` property.

Schema construct:

```
xs:complexType xs:simpleContent xs:extension base="name"
```

See ***XML Schema Part 1: Structures*, section 3.4**.

Derivation: restriction
Defines a simple type with attributes by restricting an existing simple type with attributes or complex type which is referenced by the `Base type` property. When a complex type is restricted, the type of the simple content must be determined by a local simple type declaration.

Schema construct:

```
xs:complexType xs:simpleContent xs:restriction base="name"
```

See ***XML Schema Part 1: Structures*, section 3.4**.

Tsd unique

Occurs below `tsd:doctype`. Defines a uniqueness constraint over all instances of the doctype where the key is defined.

Schema construct:

```
tsd:unique
```

For more information on defining unique keys, see *Physical Schema for Elements and Attributes* in the *Tamino XML Schema User Guide* (located in the Tamino XML Server documentation).

Unique

Defines a uniqueness constraint over different parts of the same instance.

Schema construct:

```
xs:unique xs:selector xs:field
```

See ***XML Schema Part 0: Primer*, section 5.1**.

11 Properties Explained

This chapter describes the properties that can be defined in a Tamino schema. It covers the following topics:

Note: When not indicated otherwise, the properties are available in all views.

Schema Properties

A schema has only logical properties.

Logical constructs are described in detail in the *XML Schema Part 0: Primer* on the W3C site ***http://www.w3.org/TR/xmlschema-0/***.

Schema Editor Property	Schema Language Syntax	Short Description
Schema name	`<tsd:schemaInfo name = "name">`	Name of the schema.
Collection name	`<tsd:collection name = "name">`	Name of the Tamino collection to which the schema is to belong. If the collection does not exist, it is created.
Namespaces	`<xs:schema xmlns:prefix1 = "namespace1" xmlns:prefix2 = "namespace2" targetNamespace = "namespace2">`	The Tamino Schema Editor provides a property editor that allows you to specify namespace declarations, including the target namespace declaration. See ***Namespaces***. The Tamino Schema Editor moves all namespace declarations to the schema node and manages them there. For namespaces, see W3C's *Namespaces in XML* recommendation at ***http://www.w3.org/TR/REC-xml-names/***. For target namespace, see ***XML Schema Part 0: Primer*, section 3.1**.
Target namespace		Displays the target namespace (read only). This can be changed between the defined namespaces.
Attribute form default	`<xs:schema attributeFormDefault = "unqualified|qualified"...>`	Specifies whether attributes in instances of the schema must be qualified with a namespace prefix. Default is "unqualified". See ***XML Schema Part 0: Primer*, section 3.1**.
Element form default	`<xs:schema elementFormDefault = "unqualified|qualified"...>`	Specifies whether elements in instances of the schema must be qualified with a namespace prefix. Default is "unqualified". See ***XML Schema Part 0: Primer*, section 3.1**.
Final default attribute	`<xs:schema finalDefault="#all | List of (extension | restriction)">`	Determines the default behavior in the case of derivation. See ***XML Schema Part 0: Primer*, section 4.8**.

Schema Editor Property		Schema Language Syntax	Short Description
XML Schema version		`<xs:schema version = "version">`	Version of the schema.
Language identification		`<xs:schema xml:lang = "en">`	Language identification. See W3C's XML recommendation at *http://www.w3.org/TR/REC-xml/#sec-lang-tag*.
ID		`id="id"`	Unique identifier for the schema element within the schema XML document. See W3C's XML recommendation at *http://www.w3.org/TR/REC-xml/#id*.
Tamino schema information			Open the group tree to display the following properties.
	Version	`<tsd:version>version</tsd:version>`	Version of the schema. Value provided by the Tamino Server (read-only).
	Created	`<tsd:created>created</tsd:created>`	Creation date of the schema. Value provided by the Tamino Server (read-only).
	Modified	`<tsd:modified>modified</tsd:modified>`	Date the schema was last saved. Value provided by the Tamino Server (read-only).
	Server	`<tsd:server>server</tsd:server>`	Version of the Tamino Server. Value provided by the Tamino Server (read-only).

The following example shows the generated code for a schema named "patientschema" in the Tamino collection "hospital". Element and attribute form are unqualified.

```
<?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:documentation>Schema modelling a hospital patient's record.</xs:documentation>
    <xs:appinfo>
      <tsd:schemaInfo name = "patientschema">
        <tsd:collection name = "hospital">
....
....
```

Tamino Doctype Properties

A doctype has logical and physical properties.

Logical Properties

Schema Editor Property	Schema Language Syntax	Short Description
Doctype name	`<tsd:doctype name = "name">`	Name of the Tamino doctype. This name must be the same as the name of the root element declaration.
Namespaces	`<xs:doctype xmlns:prefix1 = "namespace1" xmlns:prefix2 = "namespace2">`	This namespace declaration serves as hint for namespace cleaning and Tamino X-Query processing. Tamino X-Query processing requires namespace URIs. Since the X-Query language allows only prefixes to be used, the mapping between the prefix and the URI for a doctype is done by these namespace declarations. For namespaces, see W3C's *Namespaces in XML* recommendation at ***http://www.w3.org/TR/REC-xml-names/***. For query processing, see *Namespace Handling for Specific XMachine Requests* in the *XML Namespaces in Tamino* section of the Tamino XML Server documentation.

Physical Properties

The physical properties for a Tamino doctype depend on the storage type that has been specified for the doctype.

The following table lists all properties in alphabetical order that are available for the different storage types (XML, Non-XML and shadow XML).

Schema Editor Property	Schema Language Syntax	Storage Type	Short Description
Compress	`<tsd:compress>always </tsd:compress>`	XML	Specifies data compression on data storage. ■ **smart** Default. Tamino checks the data to be stored and finds the best compromise between speed and storage space.

Schema Editor Property	Schema Language Syntax	Storage Type	Short Description
			■ **always** Always compress as much as possible. This setting makes sense if the primary issue is storage size. Especially for small documents, this will increase retrieval time, but will use as little space on disk as possible. ■ **none** Do not compress small data records. This setting makes sense if you expect most of your documents to be reasonably small (smaller than 8000 characters), and you want to optimize processing speed, sacrificing storage space. Large documents are not effected by this setting. ■ **off** Do no compression at all. ■ **utf8** 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.
Computed Indexes	`<tsd:computedIndex></tsd:computedIndex>`	XML and Shadow XML	Specifies the use of one or more computed indexes, and allows you to define the XQuery function to be used for each computed index. You can select the property editor for this field by choosing the ... button. In the property editor, supply values for the required collation properties. Also in the property editor, in the **Name** column, select the line containing "New_Computed_Index", and choose the button ... in the

Schema Editor Property	Schema Language Syntax	Storage Type	Short Description
			corresponding **Computed Index Function** column. The resulting dialog allows you to select a stored XQuery function from an XQuery module in the Tamino database. If you wish to define more than one computed index, use the icon to create a new entry in the list of computed indexes. The computed indexes are displayed in the list of physical properties in the form `[comp1, comp2, ...]`, where `comp1`, `comp2` etc. are the names of the computed indexes.
Content	`<tsd:content>closed </tsd:content>`	XML	Default is "closed", meaning instances containing instances not defined in the schema are rejected. The value "open" means instances are stored even if they contain structures not defined in the schema. You should coordinate this value with `Structure index`.
Delete access	`<tsd:delete> </tsd:delete>`	All storage types	Default is "true", meaning instances may be deleted from the Tamino XML data store.
ino:id re-usage	`<tsd:systemGeneratedIdentity reuse = "booleanValue"> </tsd:systemGeneratedIdentity>`	XML	Specifies whether an ino:id is to be reused if an instance is removed from the data store or not. Default is "true". See the description of the element `tsd:systemGeneratedIdentity` in the *Tamino XML Schema Reference Guide* (located in the Tamino XML Server documentation).
Insert access	`<tsd:insert> </tsd:insert>`	All storage types	Default is "true", meaning instances may be inserted into the Tamino XML data store.
No conversion	`<tsd:nonXML> <tsd:noConversion></tsd:noConversion> </tsd:nonXML>`	Non-XML	Possible values are "true" and "false". Default is "true", meaning

Schema Editor Property	Schema Language Syntax	Storage Type	Short Description
			that the doctype node is of type non-XML. Text indexing is not possible for data stored into doctypes with the noConversion flag set.
Pure X-Node mapping	`<tsd:map><tsd:pure> </tsd:pure></tsd:map>`	XML	The default is "false", meaning that this property is not active. This property is only applicable to Tamino schemas that have no other function than expressing a mapping to one Adabas file. If this property is set to "true", Tamino stores all data except the schema in the Adabas file. If elements of your schema are mapped to Adabas PE groups or MU fields, it is recommended that you set the property `Minimum occurrence` to "0" and the property `Maximum occurrence` to "unbounded". Otherwise, schema validation conflicts may occur during update processing.
Read access	`<tsd:read> </tsd:read>`	All storage types	Default is "true", meaning instances may be read from the Tamino XML data store.
Structure index	`<tsd:structureIndex>`*value* `</tsd:structureIndex>`	XML	Specifies that all nodes not declared by the schema will be registered in the repository. Possible values: ■ **condensed** Default. The repository registers the existence of an undeclared node for the doctype. ■ **full** The repository registers the existence of undeclared nodes, as well as the instances in which they occur.

Schema Editor Property	Schema Language Syntax	Storage Type	Short Description
			■ **none** No structural indexing is performed. For details on structure indexing, see *Indexing* in the *Advanced Concepts* section of the Tamino XML Server documentation.
Update access	`<tsd:update> </tsd:update>`	All storage types	Default is "true", meaning existing instances may be updated in the Tamino XML data store.

The Tamino doctype properties in the examples below define a Tamino doctype "patient".

In the first example, the content of the doctype is defined as "closed", meaning only those instances will be stored which contain structures defined in the schema. Instances may be read, but not inserted, updated or deleted from the Tamino store:

```
....
<tsd:doctype name = "patient">
  <tsd:logical>
    <tsd:accessOptions>
      <tsd:read></tsd:read>
    </tsd:accessOptions>
  </tsd:logical>
</tsd:doctype>
....
```

In the second example, all instances will be stored, even if they contain structures not defined in the schema. Instances may be read, inserted, updated or deleted from the Tamino store:

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

Logical Properties for XML Elements

The Tamino Schema Editor always shows the logical properties that are relevant to the element that is currently selected in the schema tree.

The following table lists all logical properties in alphabetical order. Not all properties may be available for the currently selected element.

For detailed information and examples, see the following W3C recommendation:

- *XML Schema Part 0: Primer* at ***http://www.w3.org/TR/xmlschema-0/***.
- *XML Schema Part 1: Structures* at ***http://www.w3.org/TR/xmlschema-1/***.
- *XML Schema Part 2: Datatypes* at ***http://www.w3.org/TR/xmlschema-2/***.

Schema Property		Short Description
Abstract		For elements and types. When an element or type is declared to be abstract, it cannot be used in an instance document. See ***XML Schema Part 0: Primer, section 4.7***.
Block		For complex types and elements. Controls which derivations and substitution groups may be used in instance documents. See ***XML Schema Part 0: Primer, section 4.8***.
Collation		To enter a collation, change the value of the logical property `Collation` from "no" to "yes". The other properties of the collation are now editable. If the value of `Collation` is "yes", a default collation or a collation with the specified property values is generated. If the value of `Collation` is "no", no collation is generated. For information on the ICU Collation Service, see **http://www.icu-project.org/**. Open the group tree to display the following properties.
	Collation language	Only available if `Collation` is set to "yes". Specifies locale by ISO-3166 language code and country for collating purposes. For example, "DE" for German or "DE-AT" for Austrian German. There is no default, meaning collating keys are language neutral. This should provide good results for most languages.
	Collation strength	Only available if `Collation` is set to "yes". Specifies the level of comparison. Possible values: primary: Level 1: base character comparison, e.g. "a" < "b". secondary: Level 2: accents on characters are compared, e.g. "as" < "Ãs" < "at".

Schema Property		Short Description	
		tertiary	Level 3 (default): uppercase and lowercase characters are compared, e.g. "ao" < "Ao" < "aÃ²". 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 levels 1 through 3. Should only be used, if a distinction based on punctuation is required.
		identical	Level 5: Used if levels 1 through 4 yield identical results. The Unicode point values are compared. Note that this level of comparison can impact performance negatively.
	Collation case first	Only available if `Collation` is set to "yes". With "upperfirst" (default), words starting with uppercase will be sorted before words starting with lowercase. A value of "lowerFirst" will reverse this behavior.	
	Collation alternate	Only available if `Collation` is set to "yes". The value "shifted" sorts words containing punctuation marks together (e.g. "bi-weekly" and "biweekly"), that is, punctuation is ignored for levels 1 through 3. The value "non-ignorable" (default) will distinguish these words and sort them separately, that is, punctuation marks will be accounted for.	
	Collation case level	Only available if `Collation` is set to "yes". The value "true" makes 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".	
	Collation french	Only available if `Collation` is set to "yes". The value "true" will produce French accent sorting. The default is "false", but the function is switched on if `Collation language` is "FR".	
	Collation normalization	Only available if `Collation` is set to "yes". The value "true" produces results as if text were normalized. The default is "false" (no normalization) for most languages.	
context		For element info and attribute info. Specifies the element or attribute for which the physical properties are defined. The element or attribute is addressed by a path. For information on the path syntax, see ***Advanced Physical Properties***.	
Data type		The base type of the declared type, element or attribute. See ***Schema Tree Items Explained***.	
Default value		For attribute declarations, specifies the value used if the attribute is absent from an instance (the `use` property must then be "optional").	

Schema Property		Short Description
		For element declarations, specifies the element content if the element occurs in the instance but is empty. Element instance content overrides the default value. The properties `Default value` and `Fixed value` are mutually exclusive. See ***XML Schema Part 0: Primer*, section 2.2.1**.
Default function		For elements and attributes. The name of the server extension function that calculates the default value. The required server extension can be selected from a dialog box. See ***Server Extensions*** for further information.
Derivation method		For complex types and elements. Selects the derivation method to be used. See ***Schema Tree Items Explained***.
Documentation text		The text of the documentation node. The Tamino Schema Editor provides a property editor for entering the text. See ***Documenting a Schema***.
Facets		Open the group tree to display the following properties. When the **Fixed** check box is marked for one of the facets, the value for this facet cannot be derived any further.
	Length	Specifies the string length. See ***XML Schema Part 2: Datatypes*, section 4.3.1**.
	Minimum length	Specifies the minimal string length. See ***XML Schema Part 2: Datatypes*, section 4.3.2**.
	Maximum length	Specifies the maximal string length. See ***XML Schema Part 2: Datatypes*, section 4.3.3**.
	Total digits	Specifies the maximum number of digits allowed in an element marked as datatype "decimal". See also the property `Fraction digits` below. See ***XML Schema Part 2: Datatypes*, section 4.3.11**.
	Fraction digits	Specifies the maximum number of digits in the fractional part of elements marked as datatype "decimal". If `Total digits` is set to "8" and `Fraction digits` is set to "2", the element can contain numerical values with 6 digits before and 2 digits after the decimal. See ***XML Schema Part 2: Datatypes*, section 4.3.12**.
	Min inclusive	Constrains the value space to values with a specific inclusive lower bound. See ***XML Schema Part 2: Datatypes*, section 4.3.10**.
	Max inclusive	Constrains the value space to values with a specific inclusive upper bound. See ***XML Schema Part 2: Datatypes*, section 4.3.7**.
	Min exclusive	Constrains the value space to values with a specific exclusive lower bound. See ***XML Schema Part 2: Datatypes*, section 4.3.9**.
	Max exclusive	Constrains the value space to values with a specific exclusive upper bound. See ***XML Schema Part 2: Datatypes*, section 4.3.8**.
	Regular expression pattern	Contains a regular expression that constrains the value. See ***XML Schema Part 2: Datatypes*, section 4.3.4**.
	Enumeration	Specifies a particular value that the element may contain. Multiple enumeration elements thus specify/restrict the possible values of an element. The Tamino

Schema Property		Short Description
		Schema Editor provides a property editor that allows you to specify new values and/or delete existing ones. See ***Using the Property Editor***. See ***XML Schema Part 2: Datatypes*, section 4.3.5**.
	Whitespace	Specifies how whitespace is to be treated. See ***XML Schema Part 2: Datatypes*, section 4.3.6**.
Field xpaths		For identity constraints. Tsd unique: specifies the xpath from the doctype root element to the field or fields that are to be identical. For more information on defining unique keys, see *Physical Schema for Elements and Attributes* in the *Tamino XML Schema User Guide* (located in the Tamino XML Server documentation). Unique: specifies the xpath from the element to the field or fields that are to be identical. See ***XML Schema Part 0: Primer*, section 5.1**.
Final		For types and elements. Controls whether further derivation is permitted. See ***XML Schema Part 0: Primer*, section 4.8**.
Fixed value		For attributes and elements. ■ **Attributes** Specifies the value that the attribute must have. If the attribute does not appear in an instance, Tamino will insert it with the value given by the `Fixed value` attribute. If the attribute is present in the instance, its value must match the value of the `Fixed value` attribute in the schema. ■ **Elements** Specifies the value that the element must have, if present. If the element is present in the instance, it must either have the same value as the value given by the `Fixed value` attribute, or it must be empty. In the latter case, Tamino will insert the value given by the `Fixed value` attribute. If the element is not present at all, no value will be inserted. The properties `Fixed value` and `Default value` are mutually exclusive. See ***XML Schema Part 0: Primer*, section 2.2.1**.
foreign appinfo		Contains all application information that is not from Tamino. This information may be edited as mixed content text.
Form		Specifies for the element or attribute declaration whether instance element names must be qualified with the namespace prefix. See ***XML Schema Part 0: Primer*, section 3.1**.
ID		Unique identifier for the schema element within the schema XML document. See W3C's XML recommendation at ***http://www.w3.org/TR/REC-xml/#id***.
Item type		For simple types, elements and attributes. Defines the item type of a list. See ***Schema Tree Items Explained***.
Language identification		Language identification. See W3C's XML recommendation at ***http://www.w3.org/TR/REC-xml/#sec-lang-tag***.

Schema Property	Short Description
Maximum occurrence	Indicates the maximum number of occurrences of the node allowed in an instance. For possible values, see *XML Schema Part 0: Primer*, **section 2.2.1**. See also *Specifying the Occurrence Constraints*.
Member types	For simple types, elements and attributes. Defines types that are part of a union. See *Schema Tree Items Explained*.
Minimum occurrence	Indicates the minimum number of occurrences of the node allowed in an instance. For possible values, see *XML Schema Part 0: Primer*, **section 2.2.1**. See also *Specifying the Occurrence Constraints*.
Mixed content	For complex elements only: mixed content is allowed. See *XML Schema Part 0: Primer*, **section 2.5.2**.
Name	Name of the declared node.
Namespace	For `any` and `anyAttribute` declarations. Specifies the namespace for this declaration. For details on possible values, see *XML Schema Part 1: Structures*, **section 3.10.2** (XML representation for a wildcard schema component).
Nillable	For types and elements. Defines whether an empty (instance) element may be flagged by the `nil` attribute or not. See *XML Schema Part 1: Structures*, **section 3.3.1**.
processContents	For `any` and `anyAttribute` declarations. Specifies how the contents should be processed. For details on possible values see *XML Schema Part 1: Structures*, **section 3.10.2** (XML representation for a wildcard schema component).
Public notation identifier	The public identifier of the notation declaration, a URI reference. See *XML Schema Part 1: Structures*, **section 3.12**.
Reference	For element, attribute, group, attribute group references. Contains the name of the referenced item.
Refer	For key references. References a key by its name. See *XML Schema Part 0: Primer*, **section 5.2**.
Schema location	For import, include and redefine declarations. The location of the referenced schema. It can either be stored in a Tamino database or in the file system. You can open the referenced schema in a new instance of the Tamino Schema Editor. See also: *Opening a Referenced Schema in a New Window* *Loading and Unloading the Elements from an External Schema*
selector-xpath	For identity constraints. Selects the identical elements relative to the constraint definition by xpath. See *XML Schema Part 0: Primer*, **section 5.1**.
source	For documentation and appinfo. Describes the source of its content as an URI. See *XML Schema Part 1: Structures*, **section 3.13.2**.
Substitution group	For elements. Defines which elements can be substituted for other elements. See *XML Schema Part 0: Primer*, **section 4.6**.
System	The system identifier of the notation declaration. See *XML Schema Part 1: Structures*, **section 3.12**.
Trigger	An action trigger is a server extension that executes an action when an event in the Tamino Server occurs. This action is executed in addition to the normal

Schema Property	Short Description
	processing on the server. See *X-Tension: Tamino Server Extensions* in the Tamino XML Server documentation for further information. There are three types of triggers available (On Delete, On Insert, and On Update). There can be multiple triggers of same trigger type. The required server extension can be selected from a dialog box. See ***Server Extensions*** for further information. The values for the additional parameters of the selected trigger function can be specified the triggers dialog.
Use	For attribute declarations. Specifies whether the attribute is required, optional or prohibited. See ***XML Schema Part 0: Primer*, section 2.2.1**. See also ***Declaring an Element as Optional, Required or Prohibited***.
Variety	For simple types and elements. Selects the variety to be used. See ***Schema Tree Items Explained***.

Physical Properties for the Different Storage Types

Different storage types are available:

- Properties for Native Storage
- Properties for Adabas Mapping
- Properties for SQL Mapping
- Properties for SXS Mapping

Note: A doctype uses other storage types. See *Tamino Doctype Properties*.

Properties for Native Storage

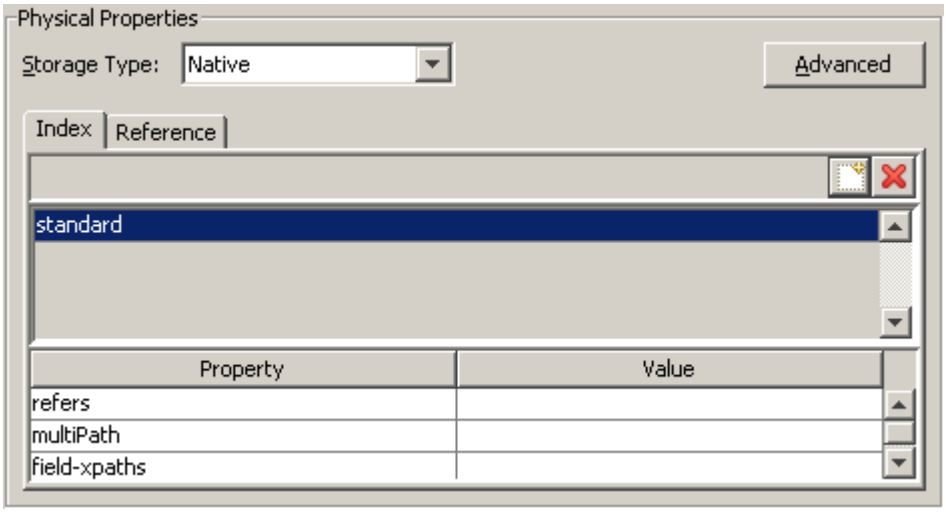
Elements and attributes of XML documents can be stored in Tamino's native XML store by specifying "Native" as the storage type.

See also *Tamino-specific Extensions to Physical Schema* in the *Tamino XML Schema User Guide* (located in the Tamino XML Server documentation).

Note: The properties for the references are described below under *Object References*.

Node Indexing

The properties for the indices and for referencing are available from two different tabs: **Index** and **Reference**.



Note: The properties for the references are described below under *Object References*.

Different types of indices can be defined: standard, text and reference index definitions. Different index technologies can be used: simple index, reference index, multi-path index and compound index. The index technology is defined by selecting a specific type of index and then specifying the required properties: `Refers`, `Multipath` and/or `Field xpaths`. Each node can have several standard and text indices. However, it can have only one reference index.

For more details, refer to *Indexing XML Data for Native Storage* in the *Tamino XML Schema User Guide* (located in the Tamino XML Server documentation.

The following buttons are provided:

	Add a new index.
	Delete the selected index.

For each index, select one of the following values:

Type of Index	Short Description	
text	A full-text search index is generated for the node contents. Recommended for text nodes. The following properties are available:	
	refers	Reference definition for sub-tree indexing. Contains an absolute path to a parent node where a reference index is defined. This type of index enables you to find the parent by using the value for the indexed node. Example: `/B[/C = "value"]` where C is the current node.
	multiPath	Label of a multi-path index. All nodes that have the same multiPath label are indexed to one index. Use case: recursive structures and queries with wildcard expressions.
standard	A standard index is generated for the node contents. This makes some retrieval operations faster (for example, those that use numerical comparisons). Recommended for nodes with numeric values. The following properties are available:	
	refers	See the above description for a text index.
	multiPath	See the above description for a text index.
	field-xpaths	Defines a compound index and contains two or more relative XPath expressions to child nodes. If the node itself is to be addressed, "." is to be used.
reference	Reference index definition for the construction of reference chains. The following property is available:	
	refers	Contains an absolute path to a parent node where another reference index is defined. If the field is empty, the document root is referenced. This type of index enables you to find the node by using the values for one or more referencing child nodes. These child nodes have to reference the node using reference definitions (see above). If a parent is referenced, the parent must also have a reference index definition. Example: `/A/B[/C = "value"]` where A is the referenced parent, B is the current node and C is the referencing child.

Object References

Elements mapped to object references are used to join information retrieved from different Tamino doctypes. Object references can also be stored in Tamino's native XML store by specifying "Native" as the storage type.

Schema Editor Property	Short Description
Collection reference	Name of the Tamino collection containing the doctype to be referenced. The default is the current collection.
Dereference	Specifies whether the contents of the referenced node must always be included in the current node ("true") or not ("false"). The default is "false". This option is not available on attributes, since they have no element content.
Node reference	Path to the node that is referenced.
Node reference operator	Specifies the operator used when referencing another object. Possible operators are: =, != < > <= >= ~=

Example

The following illustrates the native storage of a node "firstname" with a full-text index:

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

Properties for Adabas Mapping

Elements and attributes of XML documents can be mapped to an Adabas file in the following way:

Schema Construct	Possible Storage Type	Meaning
Complex element declaration	Map SubTreeAdabas or Map SubTreeAdabasPE	Adabas file (subtree) or Adabas periodic group.
Simple element declaration	Map NodeAdabasField	Adabas elementary or multiple value field.
Element with attributes declaration	Map NodeAdabasField	Adabas elementary or multiple value field.
Attribute declaration	Map NodeAdabasField	Adabas elementary or multiple value field.

Adabas Subtree (Map SubTreeAdabas)

Schema Editor Property	Short Description
dbid	The database ID of an external Adabas database
fnr	The file number of an external Adabas database.
encoding	The encoding for the node and its children. Default is ISO-8859-1.
password	The user password for the Adabas file to be accessed. **Important:** When you get a schema for which a password has been defined from the database, a single asterisk (*) is shown. This asterisk does not represent the password that has previously been defined. You have to specify the correct password for each subsequent update.
ignoreUpdate	Value "true" means the contents of the node cannot be updated by loading instances using Tamino. Default is "false".

Adabas Periodic Group (Map SubTreeAdabasPE)

Schema Editor Property	Short Description
Short name	Adabas group short name.
Ignore update	Value "true" means the contents of the node cannot be updated by loading instances using Tamino. Default is "false".

Adabas Field (Map NodeAdabasField)

Schema Editor Property	Short Description
Short name	Adabas field short name.
Format	The format of the field.
Length	The length of the field.
Encoding	The encoding for the node and its children. Default is ISO-8859-1. The encoding is available when `format` is "A".
Multiple	Value "true" means the node is mapped to an Adabas multiple value field.
Ignore update	Value "true" means the contents of the node cannot be updated by loading instances using Tamino. Default is "false".

The following illustrates the mapping of a node `middlename` to an Adabas multiple value field (MU):

```
....
<xs:element name = "middlename" type = "xs:string" minOccurs = "0" maxOccurs = ↩
"unbounded">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:nodeAdabasField shortname = "mn" format = "A">
              <tsd:multiple></tsd:multiple>
            </tsd:nodeAdabasField>
            <tsd:ignoreUpdate></tsd:ignoreUpdate>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
....
```

Properties for SQL Mapping

Elements and attributes of XML documents can be mapped to an SQL table in the following way:

Schema Construct	Possible Storage Type	Meaning
Complex element declaration	Map SubTreeSQL	SQL table.
Simple element declaration	Map NodeSQL	Column in SQL table.
Element with attributes declaration	Map NodeSQL	Column in SQL table.
Attribute declaration	Map NodeSQL	Column in SQL table.

Table (Map SubTreeSQL)

Schema Editor Property	Short Description
SQL schema name	Name of an external SQL schema.
SQL table name	Name of SQL table.
User name	User name for SQL access.
password	Password for SQL access.
ODBC data source	Name of the ODBC data source.
Primary key	Primary key for an SQL table.
Access predicate	Specifies parameters to be used to build a `where` clause in retrieval expressions.
Ignore update	Value "true" means the contents of the node cannot be updated by loading instances using Tamino. Default is "false".

Column (Map NodeSQL)

Schema Editor Property	Short Description
column	Name of SQL column.
ignoreUpdate	Value "true" means the contents of the node cannot be updated by loading instances using Tamino. Default is "false".

Note that for nodes mapped to SQL columns, the logical properties are used to express SQL data types and related properties. These definitions must reflect the specifications of the SQL schema. For more details, refer to *Mapping to SQL Tables and Columns* in the *Tamino XML Schema User Guide* (located in the Tamino XML Server documentation).

The following illustrates the schema definition of a node "born" to an SQL column. Instances are expected to hold a date of birth. Note the use of the `type` attribute on the element declaration to transport the data type information to the SQL column.

```
....
<xs:element name = "born" type = "xs:date" minOccurs = "0" form = "unqualified">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:nodeSQL column = "Date"></tsd:nodeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
....
```

Properties for SXS Mapping

Elements and attributes of XML documents can be mapped to server extensions by specifying `Map XTension` as the storage type.

Schema Editor Property	Short Description
On Delete	The name of the server extension function that is to be executed when the node is processed on delete.
On Compose	The name of the server extension function that is to be executed when the node is processed on retrieval.
On Process	The name of the server extension function that is to be executed when the node is processed on storage.
On Update	The name of the server extension function that is to be executed when the node is processed on update.
Ignore Update	Value "true" means the node cannot be updated by loading instances using Tamino. Default is "false".

The required server extension can be selected from a dialog box. See ***Server Extensions*** for further information.

The following illustrates the declaration for an element `discharged` that contains information about a hospital patient. When an instance of this element is deleted from the database, an SXS function called *SXSnotify* is triggered. An example of the function could be a message sent to the hospital administration that triggers the building of an invoice and marking the patient's bed as vacant.

```
<xs:element name = "discharged" maxOccurs = "unbounded">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:xTension>
              <tsd:onDelete>SXSnotify</tsd:onDelete>
            </tsd:xTension>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
....
....
</xs:element>
....
....
```

Advanced Physical Properties

For some element types, an **Advanced** button is provided. When you choose this button, additional information is shown. You can then define the physical properties of a node that can be accessed via multiple paths.

The following buttons are provided for defining the paths:

	Add a new path.
	Delete the selected path.

You can define paths to the node, and for each set of paths, the storage type and index. If multiple paths are specified in one input field, the paths have to be separated by a semicolon.

You must specify absolute XPath expressions, starting with the doctype. The syntax for the path is as follows:

```
/doctype-name/element-name/../{current-element-name | @current-attribute-name}
```

Example:

```
/patient/name/surname
```

The path must be valid, i.e. it must point to an existing element.

For all paths not specified here, you can define a default storage type and index. You can use the default for elements without recursion. If recursion is used, the default only applies for the first recursion level.

The Tamino schema construct generated is a `tsd:which` element.

Example

Different paths to the "surname" node could be */patient/name/surname* and */patient/doctor/name/surname*. The patient name could be stored natively with index "text", the doctor's surname could be stored in an SQL table column called "Surname".

Since different storage types are involved, you must specify the paths in different input fields.

The element declaration in the schema would look like the following:

```
<tsd:physical>
  <tsd:which>/patient/name/surname</tsd:which>
    <tsd:native>
      <tsd:index>
        <tsd:text></tsd:text>
      </tsd:index>
    </tsd:native>
</tsd:physical>
<tsd:physical>
  <tsd:which>/patient/doctor/name/surname</tsd:which>
    <tsd:map>
      <tsd:nodeSQL column = "Surname"></tsd:nodeSQL>
    </tsd:map>
</tsd:physical>
```

12 Menu Commands

This chapter provides reference information on each menu command that is available from the application window of the Tamino Schema Editor.

File

New
Creates a new schema. See *Creating a Schema from Scratch*.

Open
Opens an existing schema in your local file system. See *Opening a Schema that is Stored in the File System*.

Close
Closes a schema that is currently open in the Schema Editor.

Close All
Closes all schemas that are opened in the schema editor

Save
Saves the modifications to the current schema in your local file system. See *Saving a Schema in the File System*.

Save As
Saves the current schema under another name in your local file system. See *Saving a Schema in the File System*.

Import DTD
Imports a DTD and converts it to Tamino schema syntax. See *Importing a DTD*.

Import TSD2 Schema
Imports a Tamino version 2.x schema file and converts it to Tamino schema syntax of the current version. See *Importing a TSD2 Schema*.

Print preview
This displays a preview of the XML code of the schema. You can adjust the font size in the display area and send the XML code to a printer using the selected font size.

Print
Opens up the print dialog for print the current document.

<List of recent files>
Lists the most recently opened files. See *Opening a Schema that is Stored the File System*.

Exit
Closes the application window of the Tamino Schema Editor. See *Leaving the Tamino Schema Editor*.

Database

Get Schema
Opens a schema that is stored in Tamino. See *Getting a Schema that is Stored in Tamino*.

Validate Schema
Validates the current schema against a selected database. See *Validating a Schema*.

Define Schema
Defines a new schema to Tamino, or updates the current schema in Tamino. See *Defining and Updating a Schema in Tamino*.

Undefine Schema
Removes the current schema or schemas and all related instances from Tamino. See *Undefining a Schema*.

There are two options:

- Single : undefines single schema
- Multiple: undefines multiple schemas

Edit

Undo
Undoes the previous action. See *Undoing and Redoing the Previous Action*.

Redo
Redoes the previous undo action. See *Undoing and Redoing the Previous Action*.

Cut
Cuts the selection and copies it to the clipboard. See *Cutting, Copying and Pasting Information*.

Copy
Copies the selection to the clipboard. See *Cutting, Copying and Pasting Information*.

Paste
Pastes the contents of the clipboard, if it is allowed at this position in the schema. See *Cutting, Copying and Pasting Information*.

Delete
Deletes the selection. See *Deleting Information*.

Select All
Selects all the text available in code view. This will be working only with Code view.

Copy Path
Copies the path for the selected element in the schema tree to the clipboard. See *Copying the Path to the Clipboard*.

Rename
Renames the selected element in the schema tree. See *Renaming an Item in the Schema Tree*.

Move Up
Moves the selected element up in the schema tree. See *Moving an Element Up and Down in the Schema Tree*.

Move Down
Moves the selected element down in the schema tree. See *Moving an Element Up and Down in the Schema Tree*.

Find
Finds information within the current schema. See *Finding and Replacing Information*.

Find Next
Finds the next occurrence which matches the information that has previously been specified in the **Find** or **Replace** dialog box. See *Finding the Next or Previous Occurrence*.

Find Previous
Finds the previous occurrence which matches the information that has previously been specified in the **Find** or **Replace** dialog box. See *Finding the Next or Previous Occurrence*.

Replace

Finds and replaces information within the current schema. See *Finding and Replacing Information*.

Go to Declaration

Goes to the position in the schema tree where the selected reference is declared. See *Displaying the Declaration for a Reference*.

Go to Next Reference

Goes to the position in the schema tree where the selected declaration is referenced, or goes to the next reference which matches the selected reference. See *Displaying the Next Reference*.

View

XSD
Switches to a tree view which supports W3C's XML Schema language. See *Switching to Another View*.

Code
Switches to code view. See *Switching to Another View*.

Toolbars > Standard
Toggles the display of the standard toolbar in the application window. See *Standard Toolbar*.

Toolbars > Insert
Toggles the display of the insert toolbar in the application window. See *Insert Toolbar*.

Output
Toggles the display of the output panel in the application window. See *Output Panel*.

Status Bar
Toggles the display of the status bar in the application window. See *Status Bar*.

Expand All
Expands all nodes in the tree view of the schema. See *Schema Tree*.

Show Structure Info Text
Toggles the display of the structure info text in tree view. See *Structure Info Text*.

Insert

This menu provides commands for all schema constructs that can be inserted into the currently selected node in your schema. See *Inserting an Element in the Schema Tree*.

For a description of all available commands, see *Schema Tree Items Explained*.

Tools

ODBC Wizard
Maps an SQL schema to a Tamino schema. See *Importing an SQL Schema*.

Import from Adabas
Maps an Adabas schema to a Tamino schema. See *Importing Adabas*.

Transformation > *transformation-command*
Transforms a schema construct into another schema construct. See *Transforming and Converting Schema Constructs*.

Transformation > Convert to > *convert-command*
Converts a Choice, Sequence or All. See *Converting a Choice, Sequence or All*.

External Schema > Open
Opens the schema that is referenced in an Import, Include or Redefine declaration in a new instance of the Tamino Schema Editor. See *Opening a Referenced Schema in a New Window*.

External Schema > Load
Loads the elements from the referenced external schema into the internal model so that they can be selected from a drop-down list box. See *Loading and Unloading the Elements from an External Schema*.

Validate XML Schema
Validates the XML schema code. See *Validating the XML Schema Code*.

Options
Defines the TSD views to be shown in the application window and options for external schemas. See *Defining the Options*.

Help

Contents
Invokes the online documentation. See *Using Help*.

Help on Item
Invokes context-sensitive help for an element of the application window. See *Using Help*.

About Tamino Schema Editor
Displays information about the Tamino Schema Editor in a dialog box (for example, the version number). When you choose the **System Info** button in this dialog box, system information (for example, the Java Runtime Environment version which is used by the Tamino Schema Editor) and Tamino Schema Editor information is shown.

13 Supported Character Encodings

The Tamino Schema Editor supports the standard character encodings and their well known aliases, as shown in the following list. Please observe that Tamino XML Server may support other encodings than those listed here.

Encoding Name	Well known aliases
Big5	950, cp950, csBig5, ibm-1370_VSUB_VPUA, x-big5
cp850	850, csPC850Multilingual, IBM850
cp857	857, csIBM857
cp860	860, csIBM860, IBM860
cp861	861, cp-is, csIBM861, IBM861
cp862	862, cp867, cspc862latinhebrew
cp863	cp863, csIBM863, IBM863
cp864	csIBM864
cp865	865, csIBM865, IBM865
cp866	866, csIBM866
cp868	868, cp-ar, csIBM868, IBM868
cp869	869, cp-gr, csIBM869
EUC-JP	csEUCPkdFmtJapanese, eucjis, Extended_UNIX_Code_Packed_Format_for_Japanese, ibm-33722_VPUA, ibm-eucJP, X-EUC-JP
EUC-KR	csEUCKR, ibm-970_VPUA, ibm-eucKR, X-EUC-KR
gb18030	ibm-1392
GB2312	1383, chinese, cp1383, csGB2312, csISO58GB231280, EUC-CN, gb, gb2312-1980, GB_2312-80, ibm-1383, ibm-1383_VPUA, ibm-eucCN, iso-ir-58, X-EUC-CN
GBK	CP936, ibm-1386_VSUB_VPUA, MS936, zh_cn, windows-936
IBM01140	CCSID01140, CP01140, cpibm1140, ebcdic-us-37+euro
IBM01141	CCSID01141, CP01141, cpibm1141, ebcdic-de-273+euro

Encoding Name	Well known aliases
IBM01142	CCSID01142, CP01142, cpibm1142, ebcdic-dk-277+euro, ebcdic-no-277+euro
IBM01143	CCSID01143, CP01143, cpibm1143, ebcdic-fi-278+euro, ebcdic-se-278+euro
IBM01144	CCSID01144, CP01144, cpibm1144, ebcdic-it-280+euro
IBM01145	CCSID01145, CP01145, cpibm1145, ebcdic-es-284+euro
IBM01146	CCSID01146, CP01146, cpibm1146, ebcdic-gb-285+euro
IBM01147	CCSID01147, CP01147, cpibm1147, ebcdic-fr-297+euro
IBM01148	CCSID01148, CP01148, cpibm1148, ebcdic-international-500+euro
IBM01149	CCSID01149, CP01149, cpibm1149, ebcdic-is-871+euro
IBM037	cpibm37, ebcdic-cp-us, ebcdic-cp-ca, ebcdic-cp-wt, ebcdic-cp-nl, cp37, cp037, 037
IBM1026	CP1026, csIBM1026, Ibm-1026_STD
IBM273	273, CP273, cpibm273, csIBM273, ebcdic-de
IBM277	277, csIBM277, cpibm277, EBCDIC-CP-DK, EBCDIC-CP-NO, ebcdic-dk
IBM278	278, cp278, cpibm278, csIBM278, ebcdic-cp-fi, ebcdic-cp-se, ebcdic-sv
IBM280	280, CP280, cpibm280, csIBM280, ebcdic-cp-it
IBM284	284, CP284, cpibm284, csIBM284, ebcdic-cp-es
IBM285	285, CP285, cpibm285, csIBM285, ebcdic-cp-gb, ebcdic-gb
IBM290	cp290, csIBM290, EBCDIC-JP-kana
IBM297	297, cp297, cpibm297, csIBM297, ebcdic-cp-fr
IBM367	
IBM420	420, cp420, csIBM420, ebcdic-cp-ar1
IBM424	424, cp424, csIBM424, ebcdic-cp-he
IBM500	500, CP500, cpibm500, csIBM500, ebcdic-cp-be, ebcdic-cp-ch
IBM852	
IBM855	
IBM857	
IBM862	
IBM864	
IBM869	
IBM870	CP870, csIBM870, ibm-870, ibm-870_STD, ebcdic-cp-roece, ebcdic-cp-yu
IBM871	871, CP871, cpibm871, csIBM871, ebcdic-cp-is, ebcdic-is
IBM918	CP918, csIBM918, , ebcdic-cp-ar2, ibm-918_STD, ibm-918_VPUA
ISO-2022-CN-EXT	
ISO-2022-CN	
ISO-2022-JP	csISO2022JP
ISO-2022-KR	csISO2022KR
iso-8859-15	

Encoding Name	Well known aliases
ISO-8859-1	8859-1, cp819, csISOLatin1, IBM819, ISO_8859-1:1987, iso-ir-100, l1, latin1
iso-8859-2	8859-2, 912, cp912, csISOLatin2, ISO_8859-2:1987, iso-ir-101, l2, latin2
iso-8859-3	8859-3, 913, cp913, csISOLatin3, iso-ir-109, l3, latin3
iso-8859-4	8859-4, 914, cp914, csISOLatin4, ISO_8859-4:1988, iso-ir-110, l4, latin4
iso-8859-5	8859-5, 915, cp915, csISOLatinCyrillic, cyrillic, ISO_8859-5:1988, iso-ir-144
iso-8859-6	1089, 8859-6, arabic, asmo-708, cp1089, csISOLatinArabic, ecma-114, ISO_8859-6:1987, iso-ir-127
iso-8859-7	813, 8859-7, cp813, csISOLatinGreek, ecma-118, elot_928, greek, greek8, ISO_8859-7:1987, iso-ir-126
iso-8859-8	916, cp916, csISOLatinHebrew, Hebrew, 8859-8, ISO_8859-8:1988, iso-ir-138
iso-8859-9	8859-9, 920, cp920,latin5, csISOLatin5, ISO_8859-8:1989, iso-ir-148, l5
KOI8-R	cp878, cskoi8r, koi8
Shift_JIS	943, cp943, cp932, csShiftJIS, csWindows31J, MS_Kanji, pck, sjis, windows-31j, x-sjis
TIS-620	874, cp874, cp9066, ms874, windows-874
US-ASCII	ANSI_X3.4-1968, ASCII, ANSI_X3.4-1986, cp367, csASCII, ISO_646.irv:1983, ISO_646.irv:1991, ISO646-US, iso-ir-6, us
UTF-16BE	cp1201, UTF16_BigEndian, x-utf-16be
UTF-16LE	cp1200, UTF16_LittleEndian, x-utf-16le
UTF-8	cp1208, cp65001
UTF-16	csUnicode, ISO-10646-UCS-2, ucs-2
windows-1250	cp1250
windows-1251	cp1251
windows-1252	cp1252
windows-1253	cp1253
windows-1254	cp1254
windows-1255	cp1255
windows-1256	cp1256
windows-1257	cp1257
windows-1258	cp1258

14 Command Line Tools for Schema Conversions

This chapter describes the command line tools available for different types of schema conversion.

This information is provided under the following headings:

Conversion Scripts

The conversion scripts are provided in *installation-directory**Tamino**Tamino n.n**X_Tools**Tamino_Schema_Editor*.

The conversion scripts are provided under Windows as *.cmd* files and under UNIX as *.sh* files.

The names of the scripts are:

Name	Description
inodtdconv	For the conversion from DTDs.
inotsd2conv	For the conversion from TSD2 schemas.

Tamino DTD Converter

Converts a DTD to Tamino schema format.

Usage:

```
inodtdconv [options]
```

where *options* are:

-help
Print out this message and exit

-version
Display the version of the converter.

-dtd *file*
Mandatory. Input DTD file. Use the notation: "file://.." (under Windows: "file:///*drive*:\....".

-tsd *file*
Mandatory. Output TSD file. Use system-dependent notation (under Windows: "*drive*:*file-name*.tsd")

-outputEncoding *encoding*
Set the encoding for the output file. The default is UTF-8.

-collection *collectionname*
Collection name to be used in the TSD schema. If this is not specified, no TSD-specific information will be generated. Neither the `-schema` nor the `-doctype` options will be permitted and the schema cannot be defined to Tamino.

-schema *schemaname*
Schema name to be used in the TSD schema. If this option is not specified, the DTD file name is used.

-doctype *doctypename*
Specifies the doctype to be created in the TSD schema. If you specify this option with no value, the schema file name is used. If this option is not specified, no doctype is created.

-noValidation
The DTD is not validated.

Tamino TSD2 Converter

Converts a Tamino version 2.x schema (TSD2) to Tamino schema format.

Usage:

```
inotsd2conv [options]
```

where *options* are:

-help
Print out this message and exit.

-tsd2 *file*
Mandatory. Input TSD2 file.

-tsd *file*
Output TSD file. If this option is not set, the TSD2 file name is used with the extension *.tsd*.

-inputEncoding *encoding*
Set the encoding for the input file. If not set, UTF-8 is used.

-outputEncoding *encoding*
Set the encoding for the output file. The default is UTF-8.

-schema *schemaname*
Schema name to be used in the TSD schema. If this option is not specified, the collection name specified in the TSD2 schema is used.

-collection *collectionname*
Collection name to be used in the TSD schema. If this option is not specified, the collection name specified in the TSD2 schema is used.

-strict
Enable strict conversion mode. If not set, loose mode is used.

- **loose mode**
  This is the default conversion mode. It generates a Tamino schema against which all existing instances are most likely to validate. Note, however, that it may loosen a logical schema quite dramatically.

  V2.x schema nodes defined with Object Type "SEQ" are declared as complex elements containing `xs:choice` with `maxOccurs="*"` (declared child elements can occur in any number in any order).

  V2.x schema nodes defined with Object Type "ANY" are declared as complex elements with `mixed="true"` and containing an `xs:any` element. Any well-formed XML content is allowed (including elements from other namespaces), in mixed content.

The Doctype is always generated with "open" content, and all elements are declared with mixed content:

```
<tsd:doctype name = "...">
  <tsd:logical>
    <tsd:content>open</tsd:content>
  </tsd:logical>
.....
  <xs:element name = "...">
    <xs:complexType mixed = "true"></xs:complexType>
  </xs:element>
```

- **strict mode**
  In strict mode, the conversion attempts to infer the real or imagined DTD from which the V2.x schema was generated in order to maximize the chances that the logical schema will reflect the original DTD. This approach works in cases where it is known that the instances will parse against the original DTD.

  V2.x schema nodes defined with Object Type "SEQ" are declared as complex elements containing `xs:sequence` (the declared child elements must appear in the declared order).

  V2.x schema nodes defined with Object Type "ANY" are declared as complex elements with `mixed="true"` and containing `xs:choice`. The declared child elements can appear in any number and order, in mixed content.

-content *mode*
Mode is "open" or "close".

If mode is set to "open", the content property of all doctypes is set to "open".

If mode is set to "close", the content property of all doctypes is set to "close".

This option overrides the content property value that is specified for strict (default: close) or loose (default: open) mapping.

-version
Print version information.

Index

G

H

I

L

M

N

O

P

R

S

T

U

V