# Tamino

## Tamino Forms Handler

Version 10.11

November 2021

WEBMETHODS

This document applies to Tamino Version 10.11 and all subsequent releases.

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



**Document ID: INS-HTREQ-1011-20211101**

## Table of Contents

# Tamino Forms Handler

The Tamino forms handler is a simple mechanism for providing direct communication between an HTML form and the Tamino server without the need for writing any application code and without requiring translation from HTML code to Tamino X-Machine commands via functionality such as servlets or CGI code. It is typically used for creating and testing prototype applications, or for developing simple applications that require little programming logic.

Two basic kinds of communication with Tamino are possible with the forms handler:

- Create an XML document and assign values to its elements, using values supplied by the user in the fields of an HTML form.
- Send a query to the Tamino server.

The forms handler allows XSL stylesheets to be applied to the response documents. This combination of features allows simple browser-based applications to be constructed without the need for application programming.

The forms handler cannot create Tamino databases, collections or doctypes, so you must create these by other means (using, for example, the Tamino Manager or the Tamino Interactive Interface) before you use the forms handler.

Currently, the forms handler functionality is only available for the Microsoft Internet Explorer.

This document is intended for application developers.

This document consists of the following sections:

# 1 About this Documentation

## Document Conventions

| Convention | Description |
|---|---|
| **Bold** | Identifies elements on a screen. |
| `Monospace font` | Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties. |
| *`Italic`* | Identifies:<br><br>Variables for which you must supply values specific to your own situation or environment.<br>New terms the first time they occur in the text.<br>References to other documentation sources. |
| `Monospace font` | Identifies:<br><br>Text you must type in.<br>Messages displayed by the system.<br>Program code. |
| { } | Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols. |
| \| | Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the \| symbol. |
| [ ] | Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols. |
| ... | Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...). |

## Online Information and Support

### Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **https://documentation.softwareag.com**.

### Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.aspx** and give us a call.

**Software AG Tech Community**

You can find documentation and other technical information on the Software AG Tech Community website at **https://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

## Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

# 2 Creating an XML Document

The Tamino forms handler allows the Tamino server to support the creation of XML documents directly from an HTML form. An HTML form is defined by the `<FORM ...>` statement in HTML code. Refer to the HTML specification at **http://www.w3.org/** for details.

To create an XML document, you must provide the following information in the HTML form:

- The name of the database and collection where the new document should be created;
- The fact that you want to create an XML document (rather than sending a query);
- The names of the elements and attributes in the document and their associated values.

**Note:** The name of the first element that you specify in the HTML form must correspond to the name of a doctype in the specified collection. The generated document will be stored as a document of that doctype. If the schema for the doctype specifies mandatory elements or attributes, these must be included in the HTML form, otherwise Tamino rejects the document.

The following example illustrates the relationship between the HTML code for the form, the structure of the XML document that is created from the form, and the HTML document as it is displayed in a browser. The example is based on the Patient DTD, in which the root element is called `patient`, which has a child element called `name`, which in turn has child elements called `surname` and `forename`. The example assumes that the database is called *mydatabase* and the collection is called *Hospital*.

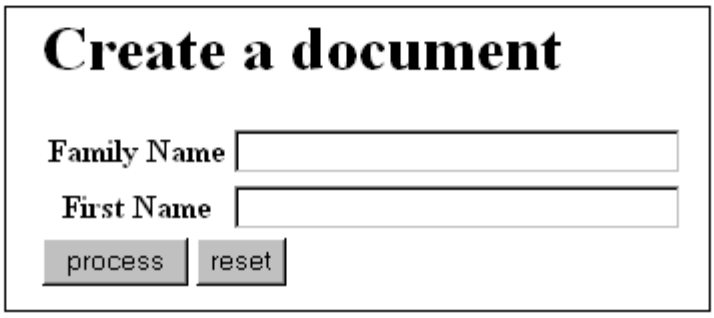
HTML code:

```
<h1>Create a document</h1>
<form name="mynewdoc"
  action="/tamino/mydatabase/Hospital">
 <input  type="hidden" name="_HTMLreq"
       value="_PROCESS">
 <input  type="hidden" name="patient" value="">
 <input  type="hidden" name="name"    value="">
 <table>
  <tr>
   <th>Family Name</th>
   <td>
    <input  type="text" size="30"  name="/surname">
   </td>
  </tr>
  <tr>
   <th>First Name</th>
   <td>
    <input type="text" size="30" name="/firstname">
   </td>
  </tr>
 </table>
 <input  type="hidden" name="name" value="/">
 <input  type="hidden" name="patient" value="/">
 <input  type="submit" value="process">
 <input  type="reset" value="reset">
</form>
```

Structure of the document generated from the HTML code:

```
<patient>
 <name>
  <surname>
     ...
  </surname>
  <firstname>
     ...
  </firstname>
 </name>
</patient>
```

Browser view of the HTML code:




**Creating an XML document**

## Specifying the Database, Collection and Doctype

The name of the database, collection and doctype are specified in an attribute of the `<FORM>` statement, using the following syntax:

```
<FORM action="CollectionURL">
```

*CollectionURL* is the URL of the collection in which you want to create the new document. *CollectionURL* is either an absolute URL (for example, *http://mynode/tamino/mydb/mycollection*) or a URL relative to the URL used to retrieve the HTML form. Hence, the HTML document must be retrieved using HTTP rather than directly from the file system, since Tamino cannot be accessed using a file URL (*file://...*).

**Example**

```
<FORM action="/tamino/mydb/mycollection">
```

Here, "mydb" is the name of the database and "mycollection" is the name of the collection.

## Creating an XML Document

To create a new XML document, use the following code in the HTML form:

```
<input type=hidden name="_HTMLreq" value="_PROCESS">
```

The command `_HTMLreq` is a special X-Machine command used only in the context of HTML forms for creating XML documents.

Note the specification of `type=hidden`, which means that there is no visible entry in the HTML form to which this corresponds. This mechanism allows required information to be passed to Tamino without being displayed in the browser.

## Creating the Opening and Closing Tags of an Element

To create the opening tag of an XML element, use the following code in the HTML form:

```
<input type=hidden name="ElementName" value="">
```

where *ElementName* is the name of the element you want to create. Note the `type=hidden` specification on this statement.

To create the closing tag of an element, use the following code in the HTML form:

```
<input type=hidden name="ElementName" value="/">
```

## Defining the Attributes of the Most Recently Opened Element

To assign a user-defined value to an attribute of the most recently opened element, use an HTML statement of the following form:

```
<input type="text" size="SizeOfField" name="@AttributeName">
```

If there are several attribute/value pairs for the element, use one such statement for each pair.

If you want to define an attribute of the most recently opened element with a fixed (i.e. not user-defined) value, use a statement of the form:

```
<input type=hidden name="@AttributeName" value="AttributeValue">
```

where *AttributeName* is the name of the attribute (note the "@" character preceding it), and *AttributeValue* is the value to be assigned to the attribute. If the element has multiple fixed attributes, use one such HTML statement per attribute.

If you want to define attributes and child elements for the most recently opened element, you must specify `<input>` statements for all attributes before you specify the `<input>` statement for the first child element.

## Creating an Element Without Attributes and Specifying Content Directly

To create an element's start tag, non-structured contents (e.g. PCDATA content as defined in a DTD) and end tag in a single step, use an HTML statement of the following form:

```
<input type="FieldType" [size="SizeOfField"] name="/ElementName">
```

where *FieldType* is a standard field type as defined for the HTML <input> statement (such as "text"), *SizeOfField* optionally specifies the size of the input field in the browser, and *ElementName* is the name of the element to be created.

**Example**

```
<input type="text" size="30" name="/surname">
```

Here, a text field is specified that appears with a width of 30 characters in the browser representation of the HTML code. The specification of name="/surname" indicates that an opening <surname> tag is to be written to the output stream, then the value that the user inserts into this input field is written, followed by a closing </surname> tag, producing a result such as the following:

```
<surname>
 ...user input ...
</surname>
```

## Creating an Element with Attributes and Content

To create an element's start tag with attributes, followed by non-structured contents (e.g. PCDATA content as defined in a DTD), followed by the closing tag, use HTML statements as follows:

```
<input type="FieldType" name="ElementName">
<input type="text" size="SizeOfField" name="@AttributeName">
...
<input type=hidden name="ElementName" value="/">
```

where *FieldType* is a standard field type as defined for the HTML <input> statement (such as "text"), *ElementName* is the name of the element to be created, *SizeOfField* specifies the size of the input field in the browser and *AttributeName* is the name of the attribute to be created.

# 3 Sending Queries to Tamino

The Tamino forms handler allows database queries to be automatically constructed using the values in multiple fields of an HTML form.

The commands used for building and sending a query to Tamino are very similar to those required for creating an XML document.

To compose a query and send it to Tamino, encode the following into the HTML form:

- The name of the database and collection to be queried;
- The fact that you want to compose a query (rather than create an XML document);
- The query itself.

The following example illustrates the principle:

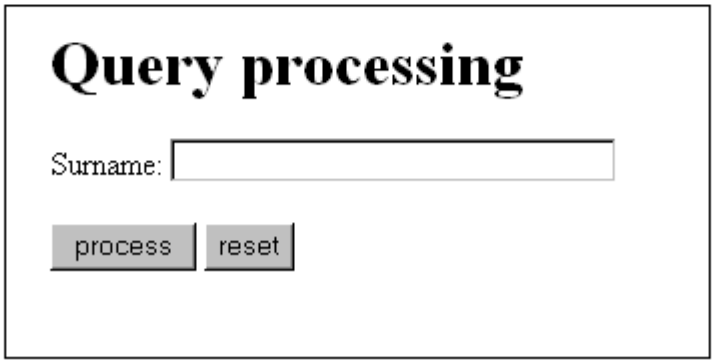
HTML code:

```
<h1>Query processing</h1>
<form name="myquery"
     action="/tamino/mydatabase/Hospital">
  <input type="hidden" name="_HTMLreq" value="_XQL">
  <input type="hidden" name="XQLNode" value="patient"
  <table>
    <tr>
      <th>Surname:</th>
      <td>
         <input type="text" size="30"
              name="name/surname">
      </td>
    </tr>
  </table>
  <br><br>
  <input  type="submit" value="process">
  <input  type="reset" value="reset">
</form>
```

Generated query:

```
patient[name/surname="... "]
```

Browser view:




**Performing a Query**

## Specifying the Database, Collection and Doctype

This is the same as for creating an XML document, as described **above**.

## Composing a Query

To indicate that you want to use the HTML form to compose a query, use the following code in the HTML form:

```
<input type=hidden name="_HTMLreq" value="_XQL">
```

## Specifying the Query's Nodes and Filters

A query consists of node selectors and filters. A node selector specifies the family of elements to be retrieved. A filter specifies a required subset of the elements (i.e. the part of a query that is enclosed in square brackets "[" and "]").

The node selector is specified in an HTML `input` statement with the following syntax:

```
<input type=hidden name="XQLNode" value="NodeAddress">
```

where *NodeAddress* is an XPath expression that identifies the location of the element within the document.

The filter is specified in another HTML `input` statement for a field that is visible in the browser.

**Example**

```
<input type=hidden name="XQLNode" value="patient">
<input type="text" size="30" name="name/surname~">
```

This generates a query of the form

```
/patient[name/surname~="ValueOfInputEntry"]
```

where *ValueOfInputEntry* stands for the value that you type in to the input field in the browser.

**Note:** the "~" character in this example. This can be used to perform a search using the X-Query `contains` operator for the given string.

# 4 Using XSL Processing Instructions

You can instruct Tamino to include an XSL processing instruction of the form:

```
<?xsl:stylesheet href='stylesheet'?>
```

in the response document. Browsers that are capable of interpreting such processing instructions (such as Microsoft Internet Explorer 5) will apply the XSL transformation to the response document.

To do this, append the URL of the required XSL stylesheet to the "_XQL" specification (see **Composing a Query** above), for example:

```
_XQL//aaa/bbb.xslt
```

This expands to `<? xsl:stylesheet href='/aaa/bbb.xslt' ?>`. If the browser is able to perform the XSL transformation, the response document is transformed according to the styles specified in the stylesheet.

# 5 Limitations

Certain restrictions apply when using the Tamino forms handler. These restrictions are described in this section. The following topics are covered:

## Mixed Content

The forms handler supports mixed content in cases where the text content of a given element precedes all child elements of the given element. However, the creation of arbitrary mixed content where text content and child elements appear in any order (other than in the case just mentioned), such as in:

```
<para>this is <em>bold</em> text</para>
```

is not supported.

## Encoding types

The forms handler is only guaranteed to work with HTML forms using:

```
Method="POST" Enctype="application/x-www-form-urlencoded"
```

which is the default encoding type. The forms handler is not designed to work with:

```
Enctype="multipart/form-data".
```

## Context of _HTMLreq command

Tamino analyses the list of parameters as they occur. After receiving the `_HTMLReq` command, Tamino assumes that all following verbs belong to the context of the HTMLReq command.

# Index

## Symbols



## C



## F



## H



## L



## Q



## X