# stxx Documentation

## Table of contents

## 1. About

## 1.1. Struts for Transforming XML with XSL (stxx)

### 1.1.1. News

**Note:**

**Security warning**: A major security hole has been found in XMLForm that can allow forged requests to execute arbitrary Java code on the server. XMLForm is used to provide XForms support and enable XML Forms, a stxx-provided Struts ActionForm implementation that uses XML as its data model. If an application used these features, the stxx 1.3b3, 1.3rc1, and 1.3rc2 releases are affected. Otherwise, any Struts forms are vulnerable with stxx releases 1.3b3 and 1.3rc1. This hole has been fixed in the XMLForm CVS as of January 21, 2003, and the patched XMLForm library is included in the 1.3 final release. To patch this hole without upgrading to stxx 1.3 final, download stxx 1.3 and copy xmlform.jar to the affected application.

### 1.1.2. Overview

Struts for transforming XML with XSL (stxx) is an extension of the struts framework to support XML and and XML transforming technologies like XSL without changing the functionality of struts.

stxx sits on top of Struts, extending its existing functionality to allow Action classes to return XML that will be transformed by technologies like XSL and Velocity (Anakia) . The idea behind stxx is to remove the need to use JSP and tag libraries for the presentation layer of the Struts framework. However, stxx does not force you to go the XML route, both technologies will work side by side. Struts for transforming XML with XSL (stxx) is an extension of the struts framework to support XML and XSL without changing the functionality of struts.

### 1.1.3. Features

- Support for both the Struts 1.0.x and Struts 1.1.x architectures
- Enhancements to the forwarding functionality of Struts to provide XML transformations based on the content produced by the action and any custom criteria like the user-agent of the client to render HTML/XML/PDF/more
- Automatic serialization of the ActionErrors, Struts Resources, ActionForms and Requests object for use in your XML document
- The ability to write your own transformation classes to make the XML transform to output type you want
- Easy migration path to Cocoon

## 1.2. The Apache Software License, Version 1.1

## 1.3. Who we are

### 1.3.1. The stxx Community

The stxx project operates on a meritocracy: the more you do, the more responsibility you will obtain. This page lists all of the people who have gone the extra mile and are Committers. If you would like to get involved, the first step is to join the mailing list.

We ask that you please do not send us emails privately asking for support. We are non-paid volunteers who help out with the project and we do not necessarily have the time or energy to help people on an individual basis. Instead, we have set up mailing lists which often contain hundreds of individuals who will help answer detailed requests for help. The benefit of using mailing lists over private communication is that it is a shared resource where others can also learn from common mistakes and as a community we all grow together.

### 1.3.2. Committers

- [DB] - Don Brown (mrdon.at.twdata.org)
- [JP] - Jeff Pennal (jeffp.at.oroad.com)

## 1.4. Frequently Asked Questions

### 1.4.1. Questions

1. **Documentation**
    - How can I help write documentation?

### 1.4.2. Answers

#### 1.4.2.1. 1. Documentation

##### 1.1. How can I help write documentation?

This project uses Apache Forrest to generate documentation from XML. Please download a copy of Forrest, which can be used to validate, develop and render a project site.

## 1.5. History of Changes

RSS

### 1.5.1. Version 1.3 (January 28, 2004)

- Fixed a regression bug that prevented stxx working correctly with JBoss due to use if different classloader to locate the stxx.properties file. (DB)
- Updated to new XMLForm jar which fixed a security flaw that allowed arbitrary method invocation. (DB)

### 1.5.2. Version 1.3rc2 (January 7, 2004)

- Added *EXPERIMENTAL* support for automatic node creation during population of XMLForms (DB)
- Added Frank's code to the contrib section (not core) that helps stxx work with bc4j (DB)
- Updated to the latest xmlforms.org jar (DB)
- Fixed the render parameter, when passed via stxx pipeline, wasn't being recognized (DB)
- Fixed extra form population with forms not using xforms (thanks Joe) (DB)

### 1.5.3. Version 1.3rc1 (November 13, 2003)

- Added support to expose xml forms via SOAP using Axis automatically (DB)
- Added a save method on XMLForm to support saving the form outside Struts (DB)
- Added Axis to support SOAP features (DB)
- Added examples for XML Forms, including SOAP interfaces, XML Form documentation, and updated Javadocs (DB)
- Updated website to include downloadable PDF's of the Getting Started Guide and the whole site. (DB)
- Fixed loading of stxx.properties to work when file cannot be loaded from the immediate classloader (thanks rainer.pruy@acrys.com) (DB)

- Fixed serializing of XML from the action losing CDATA and comment sections (thanks kpthottam@yahoo.com) (DB)

### 1.5.4. Version 1.3b3 (September 16, 2003)

- Added a new xml forms framework that allows for easy editing and validation of xml (DB)
- XMLForm.org and supporting libraries now required to support xml form features (DB)
- Form token now added to form xml (DB)
- Improved bean serialization handling of embedded dynabeans (thanks Jay M) (DB)
- Set base directory of includes in FOP when applicable (thanks Jay M) (DB)
- Fixed problem of extra text when limiting message resource serialization with "messages" transform parameter. (DB)
- Fixed attach request xml settings not properly responding to a setting of false (thanks Maryanne) (DB)
- Fixed incorrect rebuilding of action path when using wildcards (DB)
- Fixed exception thrown when debugging fop (thanks Jay M) (DB)

### 1.5.5. Version 1.3b2 (July 24, 2003)

- Added transformer property to VelocityTransformer to specify the template encoding. (DB)
- Added a DTD for the stxx pipeline configuration file (thanks Mark Renouf). (DB)
- Improved URI resolver to better resolve xsl:import and xsl:include's (DB)
- Changed how stylesheets are loaded when certain options are turned off to better accomodate servlet containers that don't implement request.getRealPath() (DB)
- Changed release build so now all relevant jars are included in example wars. (DB)
- Fixed problem with incorrect message when multiple locales are applicable. (DB)
- Fixed problem with XForm transformer, now better handles action errors with parameters. (DB)
- Removed JVM 1.4+ dependency for wildcard action mapping (DB)
- Fixed action messages and errors not using current locale when serializing text (DB)
- Better handling of messages without relevance in the current locale (DB)

### 1.5.6. Version 1.3b1 (June 30, 2003)

- Added ability for XSL and FOP transformers to reload templates if they have been modified since they were cached. (DB)
- Added Action helper class to remove the requirement of extending the stxx-enhanced Action class. Helper class handles all functionality previously only exposed to classes that extended the stxx Action class. (DB)
- Added support for auto-reloading of XSL templates when they have been modified on disk since being cached. (DB)

- Added stxx.properties property to specify what xslt processor to use (DB)
- Updated per-action xml building configuration to include all types of serialized information. Also changed to a more consistent naming scheme and deprecated old variables and methods. (DB)
- Improved URI resolving to either always resolve from the servlet context, or resolve from the directory of the parent template in the cases of xsl:imports and xsl:includes. (DB)
- Updated to Struts 1.1 final (DB)
- Updated Xalan jar to include XSLTC dependencies (DB)
- Updated and added to documentation (DB)
- Render parameter can now override both client and server transforms (patch from Mark Renouf) (DB)

### 1.5.7. Version 1.3a (June 24, 2003)

- Added support for Tiles via automatic detection. (DB)
- Added message lookup to action errors and messages upon xml serialization, also added property information. (thanks Jim Horner) (DB)
- Added a user agent selector that uses regular expressions (donated by Mark Renouf) (DB)
- Redesigned application resources to not load resources on every request and use more of the underlying struts code. New design also aggregates all related resource bundles for the XML rather than just the most specific. This should be a substantial speed improvement for those using serialized resources. (DB)
- Added the DOMWriteTransformer that writes stxx-generated xml to a request, session or servlet context attribute. Can be used to integrate stxx with JSP/JSTL. Also added related examples and documentation. (DB)
- Added support for wildcard-matched Struts action mappings. Code copied from the Struts-Wildcard project. Only works with Struts 1.1+. (DB)
- Added the ability to specify the rendering type for the CachedXSLTransformer as a request parameter. (Mark Renouf) (DB)
- Updated example to new features and better utilize Struts form features. (DB)
- Blocked Struts and stxx objects from automatic xml serialization. (DB)

### 1.5.8. Version 1.2 (June 18, 2003)

- Updated Struts to 1.1RC2 (DB)
- Fixed bug that required ApplicationResources (DB)
- Changed log level of transformation errors to WARN (DB)

### 1.5.9. Version 1.2rc2 (June 6, 2003)

- Added ability to pass stylesheet parameters to XSLT files (DB)
- Added a Velocity example and more documentation (DB)

- Standardized all licenses to the Struts license (DB)
- Updated XML jars and added instructions of their use in the documentation (DB)
- Changed what context objects were populated by the Velocity transformer. (DB)
- Updated Struts 1.0 example webapp with newer examples and marked the ones that require Struts 1.1 (DB)
- Updated Javadocs (DB)
- Fixed JDOM namespace declarations (Stephen Baishya) (DB)
- Fixed a few issues with stxx and Struts 1.0 (DB)

### 1.5.10. Version 1.2rc1 (May 28, 2003)

- Added ability to specify what application resources should be filtered per transform in a pipeline (DB)
- Added more content to advanced topics and updated other sections. (DB)
- Updated to JDOM beta 9 (DB)
- Updated xmlform.jar to one patched by Brent Baxter . Fixes problem with form error population. (DB)
- Reorganized examples to be easier to work with and better demonstrate how to use stxx. (DB)
- Updated Javadocs (DB)
- Fixed problems with DOM support (DB)

### 1.5.11. Version 1.2b2 (May 9, 2003)

- Added ActionMessage and ActionForm serialization (DB)
- Added request attribute object serialization (thanks Silvester van der Bijl) (DB)
- Added XForm (via XMLForm) transformer to support XForms by wrapping ActionForms (DB)
- Redesigned web site with Forrest (DB)
- Created a Getting Started guide (DB)
- Added transformer documentation (DB)
- Started an advanced guide (DB)
- Updated VelocityTransformer to fully take advantage of Akania (DB)
- Changed more access modifiers to better support subclassing (DB)
- Fixed PDF problem with Internet Explorer (thanks Poh Lian) (DB)

### 1.5.12. Version 1.2b1 (May 2, 2003)

- New way to define transforms by defining pipeline types that are used to match multiple requests similiar to Cocoon. This new format is only available with Struts 1.1. (DB)
- Changed access modifiers on several methods to better support subclassing (DB)
- Restructured the XSL transformer to make it easy to subclass and add SAX filters to manipulate the XML before it is transformed. (DB)

- Rewrote example application (Struts 1.1 version) using new format (DB)
- Fixed improper shutdown of stxx with Struts 1.1 (DB)

#### 1.5.13. Version 1.2a (April 15, 2003)

- Possible to use stxx without including JDOM (DB)
- New javadocs (DB)
- Redesigned transformation pipeline to be XML format independent. (DB)
- XSLT and XSL-FO transformers now use SAX event pipelines for transformation. (DB)
- Debugging the CachedFOPTransformer will now return XSL-FO and write the raw XML to file (DB)
- Depreciated "mimeType" attribute in favor of "type" (DB)
- Cleaned up imports (DB)

### 1.6. Todo List

#### 1.6.1. Low Priority

## 2. Getting Involved

### 2.1. Contributing to stxx

#### 2.1.1. Introduction

The stxx project is an Open Source volunteer project released under a very open license. This means there are many ways to contribute to the project - either with direct participation (coding, documenting, answering questions, proposing ideas, reporting bugs, suggesting bug-fixes, etc..) or by resource donations (money, time, publicity, hardware, software, conference presentations, speeches, etc...).

To begin with, we suggest you to subscribe to the stxx mailing list. Listen-in for a while, to hear how others make contributions.

You can get your local working copy of the latest and greatest code (which you find in the stxx module in the CVS code repository. Review the todo list, choose a task (or perhaps you have noticed something that needs patching). Make the changes, do the testing, generate a patch, and post to the mailing list or email the committers.

#### 2.1.2. Help Wanted Here

The rest of this document is mainly about contributing new or improved code and/or documentation, but we would also be glad to have extra help in any of the following areas:

- Answering questions on the mailing list.
- Testing stxx (especially its less-frequently-used features) on various configurations and reporting back.
- Debugging - producing reproduceable test cases and/or finding causes of bugs.
- Specifying/analysing/designing new features - and beyond.
- ... and there is just one other thing - don't forget to tell everyone who asks, how great stxx is! ;-) The more people that know about and start to use stxx, the larger the pool of potential contributors there will be.

### 2.1.3. CVS Overview

This is an overview of how to use CVS to participate in stxx development. Do not be afraid - you cannot accidently destroy the actual code repository, because you are working with a local copy as an anonymous user. Therefore, you do not have the system permissions to change anything. You can only update your local repository and compare your revisions with the real repository.

(Further general CVS usage information is at www.cvshome.org and your local `info cvs` pages or `man cvs` pages or user documentation.)

For example, using the commandline version of cvs, to checkout a copy of stxx you would enter:

```
cvs -d
:pserver:anonymous@cvs.stxx.sourceforge.net:/cvsroot/stxx co
stxx
```

### 2.1.4. CVS Committer with Secure Shell access

After a developer has consistently provided contributions (code, documentation and discussion), then the rest of the dev community may vote to grant this developer commit access to CVS.

You will need secure access to the repository to be able to commit patches. Here are some resources that help to get your machine configured to use the repository over SSH.

- The CVS Book
- www.cvshome.org
- - See the bottom of the page for links to tips for UNIX and Windows. Even if you are on UNIX, the Windows page will also help.

### 2.1.5. Contribution Notes and Tips

This is a collection of tips for contributing to the project in a manner that is productive for all parties.

- Every contribution is worthwhile. Even if the ensuing discussion proves it to be off-beam, then it may jog ideas for other people.
- Use sensible and concise email subject headings. Search engines, and humans trying to browse a voluminous list, will respond favourably to a descriptive title.
- Start new threads with new Subject for new topics, rather than reusing the previous Subject line.
- Keep each topic focused. If some new topic arises then start a new discussion. This leaves the original topic to continue uncluttered.
- Whenever you decide to start a new topic, then start with a fresh new email message window. Do not use the "Reply to" button, because threaded mail-readers get confused (they utilise the `In-reply-to` header). If so, then your new topic will get lost in the previous thread and go unanswered.
- Prepend your email subject line with a marker when that is appropriate, e.g. `[Patch]`, `[Proposal]`, `[RT]` (Random Thought which quickly blossom into research topics :-), `[STATUS]` (development status of a certain facility).
- When making changes to XML documentation, or any XML document for that matter, use a validating parser (one that is tried and true is OpenSP/onsgmls). This procedure will detect errors without having to go through the whole `build docs` process to find them. Do not expect Forrest or the build system to detect the validation errors for you - they can do it, but that is not their purpose. (Anyway, nsgmls validation error messages are more informative.)
- Remember that most people are participating in development on a volunteer basis and in their "spare time". These enthusiasts will attempt to respond to issues. It may take a little while to get your answers.
- Research your topic thoroughly before beginning to discuss a new development issue. Search and browse through the email archives - your issue may have been discussed before. Do not just perceive a problem and then rush out with a question - instead, delve.
- Try to at least offer a partial solution and not just a problem statement.
- Take the time to clearly explain your issue and write a concise email message. Less confusion facilitates fast and complete resolution.
- Do not bother to send an email reply that simply says "thanks". When the issue is resolved, that is the finish - end of thread. Reduce clutter.
- You would usually do any development work against the HEAD branch of CVS.
- When sending a patch, you usually do not need to worry about which CVS branch it should be applied to. The maintainers of the repository will decide.
- If an issue starts to get bogged down in list discussion, then it may be appropriate to go into private off-list discussion with a few interested other people. Spare the list from the gory details. Report a summary back to the list to finalise the thread.
- Become familiar with the mailing lists. As you browse and search, you will see the way

other people do things. Follow the leading examples.

*This page is based off the Contributing to Forrest page*

# 3. Documentation

## 3.1. Getting Started

### 3.1.1. Getting Started

#### 3.1.1.1. Introduction

To get started with stxx, you will need a working web application that uses Struts. This guide will walk you through how to modify your Struts web application to enable stxx functionality. It is targeted for web developers familiar with XML technologies anxious to add XML transformation features to their web application.

This guide does assume that you are familiar with the Struts framework. If you are not, familiar with Struts, please review their documentation first. You can find all of this information on the Apache Jakarta Struts web site .

### 3.1.2. Obtaining

#### 3.1.2.1. Download a package

The easiest way to get started with stxx is to download a precompiled version of stxx. With each release, three packages are created:

- `stxx-VERSION-nojars` - Contains compiled jar and example wars
- `stxx-VERSION.zip` - Contains compiled jar and example wars; stxx, Struts 1.0, and Struts 1.1 libraries
- `stxx-VERSION-full` - Contains compiled jar and example wars; stxx, Struts 1.0, Struts 1.1, FOP, XMLForm, and Velocity libraries

For full functionality, download the "full" version, however if you only need basic XSL transformation capabilities, download the second package as it contains the most commonly needed third-party libraries for default stxx operation.

#### 3.1.2.2. Retrieve from CVS

If you want to access the latest code, you can checkout the latest copy from the concurrent versioning system (CVS). For more information, see the CVS overview .

### 3.1.3. Building

#### 3.1.3.1. Build from Source (Optional)

If you downloaded a stxx release package, it is not necessary to build stxx from source, however it is possible. If you obtained stxx from CVS, you must build the stxx binary.

To build stxx, you must have Apache's Ant installed on your system and its executable script in the classpath. To compile both the stxx jar and the example web applications for Struts 1.0 and Struts 1.0, execute `ant dist_web` (UNIX) or `build.bat dist_web` (Windows). Other commonly-used ant targets are:

- **dist_jar** - Builds the stxx jar
- **javadoc** - Creates the Javadocs for stxx.
- **clean** - Cleans out compiled classes and backup files.
- **compile** - Compiles the stxx code.
- **release** - Creates the three stxx release packages.

### 3.1.4. Configuring

#### 3.1.4.1. Configuring the XML Libraries

##### Overview

Stxx requires a later version of a JAXP 1.2-compatible XML library and if Xalan/Xerces are used, a later version of both libraries are strongly recommended. The use of the XML libraries varies depending on what version of Java is being used.

##### Java 1.4+

Since Sun's Java 1.4 comes with an older version of Xalan, the XML libraries in the directory `libs/core/endorsed` need to be copied to $JAVA_HOME/lib/endorsed. For more information, see the Xalan FAQ.

##### Java 1.3.1 or earlier

Since older versions of Java don't contain any XML libraries, simply copying the jars in `libs/core/endorsed` to `/WEB-INF/lib` of your web application will suffice.

#### 3.1.4.2. Configuring with Struts 1.0

##### web.xml

Stxx works with Struts 1.0.x by sub-classing the ActionServlet class to insert the stxx processing instructions where they are needed. To facilitate this, changes to the web.xml file are required.

Since the ActionServlet class has been overridden by stxx, so the web.xml needs to point to the new implementation.

```
<servlet-name>action</servlet-name>
<servlet-class>
   com.oroad.stxx.action.ActionServlet
</servlet-class>
```

The factory attribute needs to point to the stxx application resources factory class. This is because stxx changes the way that application resources are used.

```
<init-param>
  <param-name>factory</param-name>
  <param-value>
  com.oroad.stxx.util.PropertyMessageResourcesFactory
  </param-value>
</init-param>
```

The forward attribute needs to point to the stxx implementation of the ActionForward class that supports transformations.

```
<init-param>
  <param-name>forward</param-name>
  <param-value>
  com.oroad.stxx.action.ActionForward
  </param-value>
</init-param>
```

Once this has been configured, a parameter will need to be set which points to the stxx properties file where further configuration parameters can be set. This file is expected to be loaded from the classpath, so it must reside in `/WEB-INF/classes` to be found.

```
<init-param>
  <param-name>stxxInit</param-name>
  <param-value>/stxx.properties</param-value>
</init-param>
```

| Name | Value | Status |
|---|---|---|
| action | `com.oroad.stxx.action.ActionServlet` | Required |
| factory | `com.oroad.stxx.util.PropertyMessageResourcesFactory` | Required |
| forward | `com.oroad.stxx.action.ActionForward` | Required |
| stxxInit | `/stxx.properties` | Required |

**Table 1: All web.xml init-params used by stxx**

**struts-config.xml**

The next change you need to make is to the struts-config.xml file. The changes here are to add a new tag called <transform> that nests underneath the <forward> tag. This tag allows you to add as many transformation possibilities as you need, but only one will be run for a particular forward.

The changes will look like this:

```
<action path="/contactListExample"
    type="com.oroad.stxx.example.ContactListExampleAction"
    scope="request">
    <forward name="success">
        <transform name="default"
        path="/contactListExample.xsl"/>
        <transform name="Mozilla"
        path="/contactListExample_Mozilla.xsl"/>
        <transform name="MSIE"
        path="/contactListExample_MSIE.xsl"/>
    </forward>
</action>
```

The name attribute of the transform tag is the most important. It allows for stxx to choose the transformation to perform on the XML for a given criteria. By default, stxx will choose the transformation based on the type of browser the client is using.

In this case, the value of the name attribute maps to the list of defined user-agents in the stxx.properties file. A value of default is the value to be used if no match to the user-agents can be found.

For example, you could put the name "Mozilla" in the stxx.properties user-agent property and assign various Mozilla user-agents strings as the value, then stxx will try to match the user agent of the client to whatever is in the Mozilla property. For more information, please check out the section Configuring stxx.properties.

The path defines the location that stxx can find the XSL file to transform the XML into HTML. The path is referenced from the root of the web application, for example "/foo.xsl" will match to the URL http://www.foo.com/webapp/foo.xsl.

The "type" attribute is the key that tells stxx which transformation process to use. By default, it is set to 'html.' This tells stxx that the XML you create should be transformed with the 'html' transformer, which is usually configured to combined the XML with the XSL template you specify and output HTML as it is in this example. Stxx supports transformation types that use XSL-FO (for PDF's, SVG's, RTF's, etc), Cocoon, Velocity, XMLForm, and of course XSL transformations for any text-based output type.

| Attribute | Value | Status |
|---|---|---|
| name | The name of the selector | Required |
| path | The path to the XSL file | Required |
| render | `client \| server` (default) | Optional |
| debug | `true \| false` (default) | Optional |
| type | Which transformer to use, `html` is the default | Optional |

**Table 1: All struts-config.xml attributes used by the stxx transform tag**

The same transform tag also works within the global-forwards tag.

#### 3.1.4.3. Configuring with Struts 1.1

##### web.xml

Struts 1.1 introduced a significant number of changes to its architecture. So much, that stxx had to be changed radically in order to co-exists with Struts. Many of these changes to the Struts framework were geared towards making it more extensible. One of these changes, the Struts Plug-in framework, is what stxx uses to make transformations happen. Using stxx with Struts 1.1 requires only a few changes to the web.xml file. Instead of changing the pointer to the main ActionServlet class as you did with Struts 1.0, you can leave all of the Struts default parameters the same, just adding the following parameter, which points to the stxx properties file where further configuration parameters can be set. This file is expected to be loaded from the classpath, so it must reside in `/WEB-INF/classes` to be found. This properties file is used by both versions of stxx.

```
<init-param>
  <param-name>stxxInit</param-name>
  <param-value>/stxx.properties</param-value>
</init-param>
```

| Name | Value | Status |
|---|---|---|
| stxxInit | `/stxx.properties` | Required |

**Table 1: All web.xml init-params used by stxx**

##### struts-config.xml

In Struts 1.1, the config file is no longer hard-coded to struts-config.xml. The file and it's location are defined by the plugin property 'pipeline-config' for stxx 1.2 or greater and 'transform-config' for stxx 1.1. In this file, you will need to define the plug-in parameters for stxx to be registered in Struts as a plug-in.

| Name | Value | Status |
|---|---|---|
| pipeline-config | Location of pipeline configuration file located from the root of the web application. | Required for stxx 1.2 unless `transform-config` is set. |
| transform-config | Location of transform configuration file located from the root of the web application. | Required for stxx 1.1 or earlier, or if `pipeline-config` isn't set. |

**Table 1: All struts-config.xml plugin properties**

The plug-in configuration will look like this:

```
<plug-in className="com.oroad.stxx.plugin.StxxPlugin" >
    <set-property property="pipeline-config"
                  value="/WEB-INF/stxx-transforms.xml" />
</plug-in>
```

This tells Struts to load the class StxxPlugin and to pass the parameter pipeline-config to it. This property tells the plug-in where to find all the transformations to perform.

If you plan to use the localization features of Struts, you also need to configure Struts to use a special factory when loading the message resources. The configuration should look like this:

```
<message-resources parameter="com.oroad.stxx.example.ApplicationResources"
                     factory="com.oroad.stxx.util.PropertyMessageResourcesFactory"/>
```

Transformations are no longer defined in struts-config.xml. The plug-in is configured to look for any forward with a '.dox' extension. In the case of `pipeline-config`, Struts forwards with '.dox' extensions are matched by a pipeline that contains one or more transform. In the case of `transform-config`, forwards are matched to a <transform> tag in the `transform-config.xml` file.

To make a Struts action map to a transform, simply make the path attribute of the forward tag end with a '.dox', such as:.

```
<forward name="success" path="index.dox"/>
```

The path index.dox will match a pipeline in the `pipeline-config.xml` file.

### Transform Definitions

As of stxx version 1.2, you can wrap your transform definitions in pipelines that are used to match multiple Struts action forwards. A simple wildcard matching scheme is used to match forward types. Previously, you had to write one transform per Struts action forward which could result in hundreds of lines for a moderately sized application, all of which the needed to be in sync with the Struts configuration files. Now, a moderately sized application (>50 action mappings) requires only a couple of pipelines.

This is an example of a more complex pipeline definition to show its syntax:

```
<pipeline match="browser/*.dox">
    <transform type="html">
        <param name="path" value="/{1}.xsl" />
        <param name="path" value="/style.xsl" />
        <param name="render" value="server" />
    </transform>
    <transform type="html" when="MSIE">
        <param name="path" value="/{1}_IE.xsl" />
        <param name="render" value="client" />
    </transform>
</pipeline>
```

A pipeline is used to match multiple action forwards. This pipeline will match "browser/index.dox" or "browser/foo.dox". A "*" matches any text other than "/", and a "**" matches any text _including_ "/". This way you can encode information in the "path" attribute of your "forward" element from your struts-config.xml like using the scheme "PIPELINE_TYPE/NAME.dox". This works exactly like Cocoon 2, in fact the wildcard code is copied from Cocoon. It should make the transition to Cocoon easier.

A transform two attributes: "type" and "when". The type refers to the name of the transformer to use to handle the operation (in this case, an instance of CachedXSLTransformer). The "when" attribute is the same as the "selector" attribute was in the previous schema or the "name" attribute is in the Struts 1.0 version. It basically is the switch to determine which transform gets used. In this case, the transform is selected by determining which browser the client is using. See stxx.properties for more information.

In the transform's params, the "{1}" is used to match the first wildcard in the pipeline's "match" attribute. You can have up to 9 wildcards. Notice there can be more than one instance of a parameter. These values are made available to the Transformer as a List.

Any other parameters, like "render" or "debug", would be used similiarly. For Transformer creators, you can define your own parameters to suite your purposes.

### 3.1.5. Creating stxx Actions

#### 3.1.5.1. The stxx Action

First of all, when creating a new Action class, you will need to change the base class that your Action class extends. Previously, you would want to extend `org.apache.struts.action.Action`. Now, you will want to extend `com.oroad.stxx.action.Action`. This changes the Action class slightly introduces a new method called saveDocument(HttpServletRequest, org.jdom.Document OR org.w3c.Document). This new method works in the same ways as the Struts saveErrors method that you use when using the ActionErrors object. The document in this case is saved to the request object as an attribute. The object is later retrieved by the stxx transformer to create the output type.

In the code for the perform method of the Action class, you would populate the Document object with XML. This document object will then get retrieved later by the stxx transformer where it will be processed. This usually means an XSL file will transform it. Think of the document object as an XML data stream that is the result of running the Action's business logic.

```
public class ContactListExampleAction extends Action {
    public ActionForward perform(ActionMapping mapping,
                                 ActionForm form,
                                 HttpServletRequest request,
                                 HttpServletResponse response)
            throws IOException, ServletException {

        //create a new XML document for this Action with the root
        //element of "contactListExample"
        Document document =
        new Document(new Element("contactListExample"));

        //add some data to the XML document so that the Action
        //will produce XML in the form
        Element contactList = new Element("contactList");
        Element contact = new Element("contact");
        contact.addContent(new Element("display")
                .setText("Blow, Joe"));
        contact.addContent(new Element("email")
                .setText("jblow@work.com"));
        contactList.addContent(contact);

        contact = new Element("contact");
        contact.addContent(new Element("display")
                .setText("Doe, Jane"));
        contact.addContent(new Element("email")
                .setText("jane_doe@hotmail.com"));
         contactList.addContent(contact);

        contact = new Element("contact");
        contact.addContent(new Element("display")
                .setText("Mickey Mouse"));
        contact.addContent(new Element("email")
                .setText("mm@mousehouse.ca"));
        contactList.addContent(contact);
        contact = new Element("contact");
        contact.addContent(new Element("display")
                .setText("Gordie Johnson"));
        contact.addContent(new Element("email")
                .setText("gj@bigsugar.ca"));
        contactList.addContent(contact);

        document.getRootElement().addContent(contactList);
        saveDocument(request, document);

        //return a "success" since nothing could possibly go wrong
        return mapping.findForward("success");
    } //end perform
} //end TestSuiteAction
```

#### 3.1.5.2. The XML Document

When the XML is passed to the transformation process, it's contents will not only include the XML you've defined in your Action, but XML for all parameters and attributes in your request object, all errors in your ActionErrors object, the contents of your ApplicationResources file (included localized versions), all ActionMessages, and the ActionForm (if applicable).

```
<?xml version="1.0" encoding="UTF-8" ?>
<stxx>
  <contactListExample>
    <contactList>
      <contact>
        <display>Blow, Joe</display>
        <email>jblow@work.com</email>
      </contact>
      <contact>
        <display>Doe, Jane</display>
        <email>jane_doe@hotmail.com</email>
      </contact>
      <contact>
        <display>Mickey Mouse</display>
        <email>mm@mousehouse.ca</email>
      </contact>
      <contact>
        <display>Gordie Johnson</display>
        <email>gj@bigsugar.ca</email>
      </contact>
  </contactList>
  <form />
  <messages />
  <request>
    <param name="debug">
      <value>true</value>
    </param>
  </request>
  <applicationResources>
```

```
    <key name="run.example">Run the example</key>
    <key name="debug.example">View XML Source</key>
  </applicationResources>
</contactListExample>
</stxx>
```

The request object will be populated with all the data being sent into the Action class along with all the request attributes that are being set in the Action class as well. The entire ApplicationResources file will be attached as well so that the transformation method has access to all of the defined key-value pairs defined in that properties file.

## 3.2. XML Forms

### 3.2.1. XML Forms

#### 3.2.1.1. Introduction

An XML form is a reusable implementation of a Struts ActionForm that uses XML to model the form data rather than a JavaBean or dynabean. Stxx automatically populates the XML's attributes and text nodes from the HTTP request and can validate the XML using pluggable validation implementations, defaulting to Schematron. XML forms can be integrated with XMLForm.org's implementation of XForms, and optionally use a "view state" mechanism (similiar to ASP.NET) where the form state is saved to a hidden variable in the form. Furthermore, XML forms have no dependencies on Struts or Servlet API's so they can be reused in a message-style SOAP web service, for example, simultaneously.

There are two types of XML forms stxx provides:

| Name | Description |
|---|---|
| com.oroad.stxx.xform.DOMForm | Uses W3C's DOM as the xml model |
| com.oroad.stxx.xform.JDOMForm | Uses JDOM as the xml model |

**Table 1: XML form types**

XML form support requires XMLForm, Jakarta Common's JXPath, and Struts 1.1 or later.

### 3.2.2. Configuration

#### 3.2.2.1. Overview

To take full advantage of XML forms, there are three areas that need to be configured: ActionForm configuration, XML model definitions, and validation. In several cases, the configuration is different depending whether the XML forms will be used with

XMLForm.org's XForms implementation or not. Finally, the same XML form can be exposed as SOAP-based web service using Axis.

#### 3.2.2.2. ActionForm Configuration

To use an XML form as a Struts ActionForm, define it as a form bean using the class of the XML form in `struts-config.xml`. This is an example of a form that uses the DOM XML form:

```
<form-bean name="userForm" type="com.oroad.stxx.xform.DOMForm" />
```

If XMLForm.org's XForms implementation will be used to display the form, the location of the XForm needs to be defined. Since Struts doesn't support ActionForm parameters, the dynaform configuration elements are used. This is an example of an XML Form that uses XMLForm's XForms implementation:

```
<form-bean name="userForm"
           type="com.oroad.stxx.xform.DOMForm">
  <form-property name="xml" type="userForm.xml" />
</form-bean>
```

#### 3.2.2.3. XML Model Definition

The XML the form will populate must be defined before the XML form can be used either by a file or set in a Struts Action. Therefore, a form will populate the XML's existing attributes and text nodes. The easiest way to define the XML form's XML model is to let stxx load it either from one file that holds all XML models or, if XMLForm's XForms implementation is used, in each XForm XML.

##### The XML Models File

To define all XML models in one fine, the following property must be set in `struts-config.xml`:

| Name | Description | Default |
|---|---|---|
| xmlform-models | Location of the XML file containing all XML models. | If not specified, no XML form models will be pre-loaded. |

**Table 1: XML models file property in struts-config.xml**

The XML models file can contain one or more XML form models. This is an example of the XML models file that defines the model for the XML form named "userForm":

```
<document>
  <model name="userForm">
    <user>
      <names firstname="jim" lastname="" displayname="Jim bob jr." />
      <internet>
        <email />
        <email1 />
      </internet>
      <phone>
        <workphone />
        <homephone />
        <fax />
      </phone>
    </user>
  </model>
</document>
```

**XMLForm.org XForm File**

If the XML form is to be displayed by XMLForm.org's implementation of XForms, you can define define the XML model right in the XForm XML. This is an example of the XML model defined in an XForm form:

```
<document xmlns:xf="http://www.xmlform.org/2003">
  <model>
    <user>
      <names firstname="jim" lastname="" displayname="Jim bob jr." />
      <internet>
        <email />
        <email1 />
      </internet>
      <phone>
        <workphone />
        <homephone />
        <fax />
      </phone>
    </user>
  </model>
  <xf:form id="requestForm" view="userIdentity" action="xformExample.do" method="GET">
    <xf:label>Personal Information</xf:label>
    <error>
      <xf:violations class="error"/>
    </error>
    <xf:group ref="names">
        <xf:label>Names</xf:label>
    <xf:input ref="/firstname">
      <xf:label>First</xf:label>
      <xf:violations class="error"/>
    </xf:input>
    <xf:input ref="/lastname">
      <xf:label>Last</xf:label>
      <xf:violations class="error"/>
    </xf:input>
    <xf:input ref="/displayname">
      <xf:label>Display</xf:label>
      <xf:violations class="error"/>
    </xf:input>
    </xf:group>
    <xf:group ref="internet">
        <xf:label>Internet</xf:label>
    <xf:input ref="/email">
      <xf:label>Email</xf:label>
      <xf:help>Please check this carefully</xf:help>
      <xf:violations class="error"/>
    </xf:input>
    <xf:input ref="/email1">
      <xf:label>Alternate Email</xf:label>
      <xf:violations class="error"/>
    </xf:input>
    </xf:group>
    <xf:group ref="phone">
        <xf:label>Phone</xf:label>
        <xf:input ref="/workphone">
          <xf:label>Work</xf:label>
          <xf:violations class="error"/>
        </xf:input>
        <xf:input ref="/homephone">
          <xf:label>Home</xf:label>
          <xf:help>Home phone number is required</xf:help>
          <xf:violations class="error"/>
        </xf:input>
        <xf:input ref="/fax">
          <xf:label>Fax</xf:label>
          <xf:violations class="error"/>
        </xf:input>
    </xf:group>
    <xf:selectBoolean ref="debug">
        <xf:label>Debug?</xf:label>
    </xf:selectBoolean>
        <xf:submit id="submit" class="button">
      <xf:label>Submit</xf:label>
      <xf:hint>Add the contact</xf:hint>
    </xf:submit>
  </xf:form>
</document>
```

### 3.2.2.4. Validation

XML form support mulitiple methods of validation. To determine which validation technique to use, stxx uses the schema namespace property. Also, stxx needs the schema file itself. These properties are set as stxx plugin properties in `struts-config.xml`.

| Name | Description | Default |
|---|---|---|

| | | |
|---|---|---|
| xmlform-schema | Location of the schema file containing validation rules. | If not schema specified, no validation will be performed. |
| xmlform-schemaNS | The validation schema namespace used to determine how to validate the XML. | `http://www.ascc.net/xml/schematron` |

**Table 1: XML form validation properties in struts-config.xml**

The error messages reported by validation are checked against the current locale's message resources, if available. This is an example of a validation rule using the default validation method, Schematron:

```
<rule context="/user/names/@firstname">
  <assert test="string-length(.) &gt; 0">err.firstname.none</assert>
</rule>
```

### 3.2.2.5. Apache Axis Configuration

Apache Axis uses a Web Service Deployment Descriptor (WSDD) to configure how a web service will be handled. Typically, a web service will have a WSDD, which will then be sent to the Axis installation via the AdminClient (more info in the Axis User's Guide). If you want your application to be bundled with Axis, you can integrate this configuration into Axis's `server-config.wsdd`, the main WSDD.

stxx requires several stxx-specific parameters in the WSDD:

| Name | Description | Required |
|---|---|---|
| xmlFormClass | The XMLForm-implementing class | Yes |
| xmlFormName | The form name | No |
| xmlFormPhase | The validation phase to use | Required for validation |
| xmlFormSchema | The validation schema file | Required for validation |
| xmlFormSchemaNS | The validation schema namespace | No, defaults to `http://www.ascc.net/xml/schematron` |

**Table 1: stxx parameters in the Axis WSDD**

This is an example of a WSDD entry:

```
<service name="AddAddressService" provider="java:MSG"
         style="message" use="literal">
  <parameter name="allowedMethods" value="process"/>
  <parameter name="scope" value="session"/>
  <parameter name="className" value="com.oroad.stxx.xform.XMLFormService"/>

  <!-- custom parameters used by stxx -->
  <parameter name="xmlFormClass" value="foo.AddAddressForm" />
  <parameter name="xmlFormSchema" value="/WEB-INF/address-schema.xml" />
  <parameter name="xmlFormPhase" value="all" />
</service>
```

For more information, please consult the Apache Axis website.

### 3.2.3. Usage

#### 3.2.3.1. Overview

To create an XML form, you need to implement `com.oroad.stxx.xform.XMLForm`. If the form will be exposed as a Struts ActionForm, the easiest way is to extend `com.oroad.stxx.xform.DOMForm` or `com.oroad.stxx.xform.JDOMForm`. The XML Form can simultaneously be exposed as both a Struts ActionForm and a SOAP web service.

#### 3.2.3.2. Implementing XMLForm

If you are simply wanting to create a Struts ActionForm that lets you modify XML, then you don't need to create your own implementation of `com.oroad.stxx.xform.XMLForm` or even extend `com.oroad.stxx.xform.DOMForm` or `com.oroad.stxx.xform.JDOMForm`, you can use one of the latter two classes directly. If you want to also expose the ActionForm as a web service, you will need to extend `com.oroad.stxx.xform.DOMForm` or `com.oroad.stxx.xform.JDOMForm` and override, at a minimum, the `save()` method. Also, you can override the `validate()` method to perform any extra validation your form requires. This is because when the XMLForm is exposed as a web service, it will not have the Struts Action to perform any saving operations, so this code has to be put in the XMLForm itself.

#### 3.2.3.3. As a Struts ActionForm

Whether you use XMLForm.org's XForms implementation to generate an HTML form or not, the end result will be the same. The HTML form element names should be XPath statements pointing to the location in the XML that the values from the HTML input tags should change.

**Note:**
The node the XPath points to must exist in the XML model.

For example, for the XML form model:

```
<item id="foo">
  <name>Foo</name>
</item>
```

an HTML input text box used to change the name of the item would be defined as:

```
 <input type="text" name="item/name" />
```

and an HTML hidden field for the item's id would look like:

```
 <input type="hidden" name="item/@id" />
```

#### 3.2.3.4. As a SOAP Web Service

To consume the SOAP web service, simply pass one or more XML nodes in a document-style SOAP call with no encoding in the schema expected by the XML form. Each XML node will be processed as an individual form. stxx will then use that XML node, validate it against the schema (if available), and call the XML form's save() method.

To create the return information, for each XML node that was sent, a result element is created. Inside the result element, either the string result of the save() method is placed or a list of errors from the failed validation.

For example, say the XML form was expecting something like this:

```
<item id="foo">
  <name>Foo</name>
</item>
```

and was sent this in the SOAP body:

```
<item id="bar">
  <name>Bar</name>
</item>
```

and upon validation, all checks passed so the save() method was called and returned "success". The XML stxx would return would be:

```
<result id="0">
success
</result>
```

Now, if this XML was sent by the SOAP client:

```
<item>
  <name>Bar</name>
</item>
```

and the item's id attribute was required, the XML form's save() method would not be called and stxx would return something like this:

```
<result id="0">
  <error property="item/@id" name="item.id.missing">
    <text>Each item needs an id value</text>
  </error>
</result>
```

## 3.3.

## 3.4. XML Transformers

### 3.4.1. CachedXSLTransformer

#### 3.4.1.1. Overview

This transformer uses a JAXP 1.2 -implementing XSLT processor to transform the XML with one or more XSL files. It takes advantage of TRaX interfaces to compile and cache stylesheets to speed up the transformation process. It can handle both server and client side transformations.

#### 3.4.1.2. Features

- Multiple XSL stylesheet support
- Configurable cache of compiled stylesheets
- Uses SAX to build the document
- Can be extended to add SAX filters before XSLT processing
- Supports client-side transformations
- Locates stylesheets from the servlet context
- Can filter what application resources are included in the transform
- Can auto-reload XSL files from disk when they have been modified since being cached

#### 3.4.1.3. Configuration

**Global**

| Property | Value | Status |
|---|---|---|
| mimeType | The mime type of the transformed content | Required |
| autoReloadTemplates | `true \| false` (default). If true, the cached Templates will | Optional |

| | | |
|---|---|---|
| | be checked to see if the stylesheet has been modified since and reload as necessary. | |
| allowRenderParameter | `true \| false` (default). If true, the type of rendering can be specified as a request parameter. For example, "render=client" will force a client-side transformation. | Optional |
| resolveFromTemplatePath | `false \| true` (default). If true, URIs used in xsl:import, or xsl:include are resolved from the directory of the parent stylesheet. Otherwise, URIs are resolved from the root directory of the web application. | Optional |
| transformerFactoryClass | The name of the JAXP transformer factory implementation to use. | Optional |

**Table 1: Global stxx.properties properties**

**Note:**

If you are having problems with NullPointerExceptions when stxx is loading stylesheets, it could be because your servlet container doesn't implement request.getRealPath() in a useful manner. If this is the case, set both `resolveFromTemplatePath` and `autoReloadTemplates` to "false" and stxx will load the stylesheets using ServletContext.getResourceAsStream().

**Transform-specific**

These parameters are specified in stxx 1.2+ pipeline configuration format. See Configuring with Struts 1.1 for more information.

| Parameter | Value | Status |
|---|---|---|
| path | The path to the XSL file | Required and can be repeated |
| debug | `true \| false` (default) | Optional |
| render | `client \| server` (default) | Optional |
| messages | Matches which application resources will be included in the transformation. | Optional |
| attachRequestParameters | `false \| true` (default) | Optional |

| | | |
|---|---|---|
| attachRequestAttributes | `false \| true` (default) | Optional |
| attachForm | `false \| true` (default) | Optional |
| attachResources | `false \| true` (default) | Optional |
| attachErrors | `false \| true` (default) | Optional |
| attachMessages | `false \| true` (default) | Optional |
| OTHER | Any other parameter will be passed to the XSLT as a stylesheet parameter. | Optional |

**Table 1: Transform-specific parameters**

All transform parameters will be passed to the XSLT as stylesheet parameters

If you have a browser accessing stxx that can support rendering of XSL, you can off-load the transformation of the XML. If "client" is set, when stxx processes the XML file, it will add a processing instruction at the top of the XML file, pointing to the XSL stylesheet on the server running stxx. Your browser should be able to handle it from there.

To limit which application resources messages are included in the transformation, you can supply one or more `messages` parameters. They match application resource keys using Perl 5- compatible regular expressions.

#### Examples

Global configuration:

```
stxx.transformer.html.class=com.oroad.stxx.transform.CachedXSLTransformer
stxx.transformer.html.mimeType=text/html
```

Transform-specific configuration:

```
<pipeline match="simple/*.dox">
 <transform type="html">
    <param name="path"     value="/{1}.xsl" />
    <param name="render"   value="server" />
    <param name="messages" value="simple.*" />
 </transform>
</pipeline>
```

## 3.4.2. CachedFOPTransformer

### 3.4.2.1. Overview

This transformer uses a JAXP 1.2 -implementing XSLT processor and Apache FOP to transform the XML with one or more XSL files into XSL-FO then convert the XSL-FO into PDF or SVG. It takes advantage of TRaX interfaces to compile and cache stylesheets to speed up the transformation process. .

#### 3.4.2.2. Features

- Multiple XSL stylesheet support
- Can output Adobe PDF or SVG format
- Configurable cache of compiled stylesheets
- Uses SAX to build the document
- Can be extended to add SAX filters before XSLT processing

#### 3.4.2.3. Configuration

**Global**

| Property | Value | Status |
|---|---|---|
| outputType | `svg | pdf` (default) | Optional |
| autoReloadTemplates | `true | false` (default). If true, the cached Templates will be checked to see if the stylesheet has been modified since and reload as necessary. | Optional |
| resolveFromTemplatePath | `false | true` (default). If true, URIs used in xsl:import, or xsl:include are resolved from the directory of the parent stylesheet. Otherwise, URIs are resolved from the root directory of the web application. | Optional |
| transformerFactoryClass | The name of the JAXP transformer factory implementation to use. | Optional |

**Table 1: Global stxx.properties properties**

**Note:**

If you are having problems with NullPointerExceptions when stxx is loading stylesheets, it could be because your servlet container doesn't implement request.getRealPath() in a useful manner. If this is the case, set both `resolveFromTemplatePath` and `autoReloadTemplates` to "false" and stxx will load the stylesheets using ServletContext.getResourceAsStream().

**Transform-specific**

These parameters are specified in stxx 1.2+ pipeline configuration format. See Configuring with Struts 1.1 for more information.

| Parameter | Value | Status |
|---|---|---|
| path | The path to the XSL file | Required and can be repeated |
| debug | `true | false` (default) | Optional |
| attachRequestParameters | `false | true` (default) | Optional |
| attachRequestAttributes | `false | true` (default) | Optional |
| attachForm | `false | true` (default) | Optional |
| attachResources | `false | true` (default) | Optional |
| attachErrors | `false | true` (default) | Optional |
| attachMessages | `false | true` (default) | |

**Table 1: Transform-specific parameters**

When this transformer is debugged either by setting the "debug" param or by passing "debug=true" on the querystring (if activated), the debugging XML is the original stxx-produced XML transformed by the XSL files into XSL-FO. This should make it easier to debug FOP processing.

**Examples**

Global configuration:

```
stxx.transformer.pdf.class=com.oroad.stxx.transform.CachedFOPTransformer
stxx.transformer.pdf.outputType=pdf
```

Transform-specific configuration:

```
<pipeline match="pdf/*.dox">
    <transform type="pdf">
        <param name="path" value="/{1}.xsl" />
    </transform>
</pipeline>
```

### 3.4.3. CachedXFormTransformer

#### 3.4.3.1. Overview

This transformer extends CachedXSLTransformer to insert transformed XForm XML (as supported by XMLForm). It combines the form features of struts (dynaforms, validation, etc) with the abstraction of XForms. The XForm transformation process inserts ActionForm values and ActionErrors into the XForm.

#### 3.4.3.2. Features

- All the features of CachedXSLTransformer.
- Support for XForms
- Requires JVM 1.4+ (due to XMLForm dependency)

#### 3.4.3.3. Configuration

**Global**

| Property | Value | Status |
|---|---|---|
| mimeType | The mime type of the transformed content | Required |
| autoReloadTemplates | `true \| false` (default). If true, the cached Templates will be checked to see if the stylesheet has been modified since and reload as necessary. | Optional |
| allowRenderParameter | `true \| false` (default). If true, the type of rendering can be specified as a request parameter. For example, "render=client" will force a client-side transformation. | Optional |
| resolveFromTemplatePath | `false \| true` (default). If true, URIs used in xsl:import, or xsl:include are resolved from the directory of the parent stylesheet. Otherwise, URIs are resolved from the root directory of the web application. | Optional |
| transformerFactoryClass | The name of the JAXP transformer factory implementation to use. | Optional |

**Table 1: Global stxx.properties properties**

**Note:**

If you are having problems with NullPointerExceptions when stxx is loading stylesheets, it could be because your servlet container doesn't implement request.getRealPath() in a useful manner. If this is the case, set both `resolveFromTemplatePath` and `autoReloadTemplates` to "false" and stxx will load the stylesheets using ServletContext.getResourceAsStream().

### Transform-specific

These parameters are specified in stxx 1.2+ pipeline configuration format. See Configuring with Struts 1.1 for more information.

| Parameter | Value | Status |
|---|---|---|
| path | The path to the XSL file | Required and can be repeated |
| xform | The path to the XForm XML file | Required |
| debug | `true \| false` (default) | Optional |
| render | `client \| server` (default) | Optional |
| attachRequestParameters | `false \| true` (default) | Optional |
| attachRequestAttributes | `false \| true` (default) | Optional |
| attachForm | `false \| true` (default) | Optional |
| attachResources | `false \| true` (default) | Optional |
| attachErrors | `false \| true` (default) | Optional |
| attachMessages | `false \| true` (default) | Optional |
| OTHER | Any other parameter will be passed to the XSLT as a stylesheet parameter. | Optional |

**Table 1: Transform-specific parameters**

All transform parameters will be passed to the XSLT as stylesheet parameters

### Examples

Global configuration:

```
stxx.transformer.xform.class=com.oroad.stxx.transform.CachedXFormTransformer
stxx.transformer.xform.mimeType=text/html
```

Transform-specific configuration:

```
<pipeline match="xform/*.dox">
    <transform type="xform">
        <param name="path" value="/{1}.xsl" />
        <param name="xform" value="/{1}.xml" />
        <param name="render" value="server" />
    </transform>
</pipeline>
```

### 3.4.4. VelocityTransformer

#### 3.4.4.1. Overview

This transformer uses Velocity, specifically Anakia to transform the XML with Velocity templates. Anakia uses JDOM XML objects to make it easy to work with the XML and further extends the Element object to add direct support for XPath evaluations.

The following objects will be available to each Velocity template:

| Name | Description |
|---|---|
| $req | The HTTP request object |
| $res | The HTTP response object |
| $ses | The HTTP session object |
| $app | The servlet context |
| $root | The root element of the stxx-created XML document |
| $msg.getMessage($string) | Retrieves the localized message from the appropriate application resources |
| $relativePath | The absolute path to the web application directory |
| $escape.getText($string) | This context object will convert HTML Entities in the $string that is passed into it and it will return the converted String. This is good for dealing with CDATA. The entities that are converted are: " -> &quot; \| < -> &lt; \| > -> &gt; \| & - > &amp; |

**Table 1: Context Objects**

All node lists returned from Anakia (extended JDOM) objects through $element.selectNodes, $element.content, and $element.children have two special features:

- they support the selectNodes method just as a single element does that applies an XPath

expression to all nodes in the list, and
- when inserted into template output by simply specifying $list, they produce the XML fragment consisting of all nodes they contain. This eliminates much of the #foreach code in templates.

For more information, visit the Anakia page.

#### 3.4.4.2. Features

- The features of Velocity
- Easier to pick up than XSL
- Supposedly faster than XSL transformations

#### 3.4.4.3. Configuration

**Global**

| Property | Value | Status |
|---|---|---|
| mimeType | The mime type of the transformed content | Required |
| templateEncoding | The conventional encoding specification for the Velocity templates as supported by your JVM, for example "UTF-8" or "ISO-8859-1". | Optional |
| outputEncoding | The conventional encoding specification for the output as supported by your JVM, for example "UTF-8" or "ISO-8859-1". | Optional |

**Table 1: Global stxx.properties properties**

**Transform-specific**

These parameters are specified in stxx 1.2+ pipeline configuration format. See Configuring with Struts 1.1 for more information.

| Parameter | Value | Status |
|---|---|---|
| path | The path to the Velocity template file | Required |
| debug | `true | false` (default) | Optional |

**Table 1: Transform-specific parameters**

**Examples**

Global configuration:

```
stxx.transformer.velocity.class=com.oroad.stxx.transform.VelocityTransformer
stxx.transformer.velocity.mimeType=text/html
stxx.transformer.velocity.encoding=ISO-8859-1
```

Transform-specific configuration:

```
<pipeline match="velocity/*.dox">
 <transform type="velocity">
    <param name="path" value="/{1}.vm" />
 </transform>
</pipeline>
```

### 3.4.5. DOMWriteTransformer

#### 3.4.5.1. Overview

This transformer takes the stxx-created XML, converts it into a DOM tree, and stores it in a request, session, or application scope for use later. The request is then forwarded to "path" value. Most likely, the primary use of this transformer will be to create XML to be accessed by a JSP page. By storing the XML as a DOM node, JSTL's XML tags will be able to access the information easily with XPath.

#### 3.4.5.2. Features

- DOM can be stored in the request, session, or application scopes
- Uses request dispatcher for further request handling
- Works great with JSTL's XML tags

#### 3.4.5.3. Configuration

**Global**

| Property | Value | Status |
|---|---|---|
| attributeName | The name of the attribute the DOM will be stored under | Optional (defaults to "stxxXML") |
| scope | `application \| session \| request` (default) | Optional |

**Table 1: Global stxx.properties properties**

**Transform-specific**

These parameters are specified in stxx 1.2+ pipeline configuration format. See Configuring with Struts 1.1 for more information.

| Parameter | Value | Status |
|---|---|---|
| path | The path to forward to after the DOM is saved | Required |
| attributeName | The name of the attribute the DOM will be stored under | Optional |
| scope | `application \| session \| request` | Optional |
| attachRequestParameters | `false \| true` (default) | Optional |
| attachRequestAttributes | `false \| true` (default) | Optional |
| attachForm | `false \| true` (default) | Optional |
| attachResources | `false \| true` (default) | Optional |
| attachErrors | `false \| true` (default) | Optional |
| attachMessages | `false \| true` (default) | |

**Table 1: Transform-specific parameters**

**Examples**

Global configuration:

```
stxx.transformer.dom.class=com.oroad.stxx.transform.DOMWriteTransformer
stxx.transformer.dom.attributeName=stxxXML
stxx.transformer.dom.scope=request
```

Transform-specific configuration:

```
<pipeline match="jsp/*.dox">
 <transform type="dom">
    <param name="path"     value="/{1}.jsp" />
 </transform>
</pipeline>
```

## 3.5. Advanced Features

### 3.5.1. Advanced Features

#### 3.5.1.1. Overview

These are advanced features, configurations, and customizations of stxx. It is entirely possible to use stxx and never bother with these settings, however for the advanced user, they are available.

### 3.5.2. Selectors

#### 3.5.2.1. Overview

For each action mapping or pipeline, you can define multiple transforms and let stxx choose which ones to use based on a selection criteria. The selection criteria could be a browser's type, request parameter value, a session value, or even time of day. To configure which selector stxx will use, set the `stxx.transformSelector.class` property to the implementing class.

Stxx comes with a couple of different implementations. Each can have its own configuration properties in stxx.properties.

#### 3.5.2.2. com.oroad.stxx.util.UserAgentSelector

This selector chooses stylesheets based on matching the client browser's type. You can define a comma-delimited list of strings used to match each browser type in the property `stxx.userAgent.BROWSER_TYPE`. For example:

```
stxx.userAgent.MSIE="MSIE 6.0","MSIE 5.0"
```

defines the browser type "MSIE" to match any user agent that has the string "MSIE 6.0" or "MSIE 5.0".

#### 3.5.2.3. com.oroad.stxx.util.UserAgentRegexpSelector

This selector is like the previous selector as it chooses stylesheets based on matching the client browser's type, but instead of simply matching strings, it uses Perl 5-compatible regular expressions. Don't forget to use "\\" instead of a single "\" due to "\" having special meaning to properties files. You can define a regular expression that will be used to match each browser type in the property `stxx.userAgent.BROWSER_TYPE`. For example:

```
stxx.userAgent.MSIE=.*MSIE [56]\\.[05].*
```

defines the browser type "MSIE" to match a user agent like "MSIE 6.0", "MSIE 5.0", or "MSIE 5.5".

#### 3.5.2.4. com.oroad.stxx.util.RequestAttributeSelector

This selector uses the value of a request attribute to select stylesheets. One way to use this selector would be to create a Servlet filter that could try to determine which "theme" is set for the user from the session, but if no theme can be found, use the browser type to try to guess the most appropriate theme. To configure which request attribute it will use, set the `stxx.transformSelector.key`.

### 3.5.3. Web Services Support

#### 3.5.3.1. Overview

In addition to exposing an HTML interface for humans to interact with the data a stxx application provides, it also might be desirable to take those same actions and expose their information as pure data without any formatting for other applications to use. The exposing of pure XML data over the web is loosely known as "web services".

#### 3.5.3.2. REST-style Web Services

While there are several different architectural styles for constructing web services, stxx supports a style known as REST. Other good sources of information are here and here. Taken from the latter, here is a short unofficial list of what are REST best practices:

1. Every public resource should have a URI.
2. A URI should expose no detail of its implementation; in Costello's terms, they should be "logical" rather than "physical".
3. Since resources are objects rather than processes, URI parts should be named for nouns rather than verbs.
4. Responses to HTTP GET should be free of side effects.
5. The representation of a resource should contain links to other resources.
6. Eliminate "query strings" as often as possible.

Unfortunately, Struts is very much verb oriented (violating number 3) and requires each action mapping URL to be explictly named (violating number 5). To better support REST-syle web services, stxx incorporated code from the Wildcard-Matched Actions project to allow action mappings to use wildcards.

To demonstrate how wildcard-matched actions work, say, for example, a web application needed to display an item. Struts would require the action mapping to look something like this:

```
<action path="/viewItem"
```

```
        type="foo.ViewItemAction">
    <forward name="success" path="xml/view-item.dox" />
</action>
```

With wildcards:

```
<action path="/item/*"
        type="foo.ViewItemAction">
   parameter="{1}"
    <forward name="success" path="xml/view-item.dox" />
</action>
```

With the former, to request an item, the URL would look like this: `http://server/webapp/viewItem.do?key=1`. With wildcards, it would look like this: `http://server/webapp/item/1.do`. While it isn't perfect (the ".do" suffix isn't desirable), it is much closer to REST-style best practices.

### 3.5.4. XML Building

#### 3.5.4.1. Overview

As a web application gets larger and more complex, the XML that stxx creates gets larger and larger. In order to optimize transformation performance, stxx allows you to configure what information will be attached to the XML in several ways: globally in `stxx.properties`, per transform (Struts 1.1+), and per Action. If information is configured to no be included in any of these places, the information will not be attached to the XML.

Stxx comes with a couple of different implementations. Each can have its own configuration properties in stxx.properties.

#### 3.5.4.2. Globally

Detailed global configuration can be found in the stxx.properties page.

#### 3.5.4.3. Per Transform (Struts 1.1+)

If you are using stxx with Struts 1.1+ and are using the pipeline style of configuring transforms, you can specify in the transform definition what information should and should not be attached for any Transformer that uses stxx to build the XML (which is every Transformer included with stxx). The following transform parameters can be defined for a transform:

| Parameter | Type of information |
|---|---|

| | |
|---|---|
| attachRequestParameters | Request parameters |
| attachRequestAttributes | Request attributes |
| attachResources | Application resources for the current locale |
| attachForm | The current action form for this Action |
| attachErrors | The action errors for the current action form |
| attachMessages | Action messages |

**Table 1: Transform parameters**

#### 3.5.4.4. Per Action

Within your action code, you can call stxx methods that will tell stxx to not attach information for the duration of the request. These methods can either be called locally from an Action that extends the stxx Action, or by using the StxxHelper class.

| Method name | Type of information |
|---|---|
| attachRequestParametersXML | Request parameters |
| attachRequestAttributesXML | Request attributes |
| attachResourcesXML | Application resources for the current locale |
| attachFormXML | The current action form for this Action |
| attachErrorsXML | The action errors for the current action form |
| attachMessagesXML | Action messages |

**Table 1: XML Building Methods**

### 3.5.5. stxx.properties Configuration

#### 3.5.5.1. General Properties

The stxx.properties file is where all of the stxx specific configuration properties are kept. There are a number of general properties that can be assigned in this file, they are:

| Property | Value |
|---|---|
| `stxx.allowURLDebug` | `true \| false` |

This determines whether "debug=true" can be appended to a query string and debug information will be sent back (if the transformer supports it).

```
http://localhost:8080/stxx/index.do?debug=true
```

Would return the XML data without transforming it. While this is useful for development, it is probably better to disable this for production use.

| Property | Value |
|---|---|
| `stxx.writeXMLDebug` | `true \| false` |

When debugging information is requested either by use if the previous method or setting the debug property for your transform, this value determines whether that same information will be written to disk. Unless you need the debugging files, turn this off to prevent your disk being filled up with debug files during development/testing.

| Property | Value |
|---|---|
| `stxx.debugXMLPath` | Path to log file location |

By default, the debugging files mentioned above are written to the /WEB-INF directory. If you wish for them to be written elsewhere, uncomment this property and set it to an absolute path of a directory on your server. If you are using a version of Windows, remember to escape your backslashes, for example ex: `c:\\Tomcat4\\logs`

| Property | Value |
|---|---|
| `stxx.alwaysUseChainedRoot` | `true` (default) \| `false` |

By default, when action chaining has NOT resulted in multiple XML documents, the document created in a Action is modified by stxx to add other elements like request attributes. By setting this property to true, stxx will always use the chained root node as the root node of the document and add the action-created document to it.

| Property | Value |
|---|---|
| `stxx.chainedRootNode` | The name of the chained root node |

When action chaining has resulted in multiple XML documents or the `stxx.alwaysUseChainedRoot` property is true, stxx places their root nodes under one new root node. This property determines the name of that node.

| Property | Value |
|---|---|
| `stxx.useCSVParameterFormat` | `true \| false` (default) |

In previous versions of stxx, multiple values for a request parameter were serialized as comma-seperated values, however, the current behavior is separate "value" elements for each value. To revert to the previous CSV behavior, set this property to true.

#### 3.5.5.2. Transformers

| Property | Value |
|---|---|
| `stxx.transformer.TYPE.class` | The name of the transformer class |
| `stxx.transformer.TYPE.cache.class` | The name of the cache class |
| `stxx.transformer.TYPE.cache.name` | The unique name of the cache |
| `stxx.transformer.TYPE.cache.size` | The maxiumum number of elements allowed in the cache |
| `stxx.transformer.TYPE.cache.expiryTime` | Expiration time of the elements in the cache in seconds |
| `stxx.transformer.TYPE.CUSTOM` | Custom properties specific to the transformer |

The preceeding properties deal with what types of transformation stxx will recognize. It also allows you to customize the cache that is available to the Transformer. Only the first property is required for a transformer type. If you do not specify the next three properties, values of properties starting with `stxx.default` will be used. You can further customize each transformer by including additional properties unique to the transformer class by using the CUSTOM properties that will be read in by the transforming class

This Transformer, CachedXSLTransformer, uses a custom property named "mimeType" that is used to set the mime type of the transformation. This is how it is implemented in the `stxx.properties` file:

```
stxx.transformer.xml.class = com.oroad.stxx.transform.CachedXSLTransformer
stxx.transformer.xml.mimeType = text/xml
```

#### 3.5.5.3. Transform Selection

Each transform element is given a name. You can specify how stxx will use that name to choose the correct transform. The default transform selector is the UserAgentSelector which will try to match the browser type to choose the appropriate stylesheet customized for the browser type.

| Property | Value |
|---|---|
| `stxx.transformSelector.class` | The chosen Selector class |

#### 3.5.5.4. Serialization

To serialize a request parameter, request attribute, action error, or message resource, stxx calls an implementation of the serializer interface for the appropriate XML format. For example, the CachedXSLTransformer uses an implementation of the SAXSerializer to serialize information. Other transfomers could use other types of XML formats that might have their own Serializer interfaces.

The default serializer uses a custom serializer that recognizes JavaBeans, JDOM, DOM, arrays, collections, and primitive wrappers. To have more control over how an object is serialized, extend these classes and override the methods of your choice. For example, if you wanted to use Castor XML to serialize request attributes, override the serializeRequestAttribute method.

| Property | Value |
|---|---|
| `stxx.serialize.sax.class` | The class that serializes information as SAX events. Used by all built-in stxx transformers. |
| `stxx.serialize.FORMAT_TYPE.OPTION_NAME` | A custom serializer property where FORMAT_TYPE is the XML format type and OPTION_NAME is the name of the custom option. |

### 3.5.5.5. XML Building Rules

You can configure what information will be attached to the XML in which situations.

| Property | Value |
|---|---|
| `stxx.attach.rulesClass` | The implementation of BuilderRules that controls what XML is attached in what circumstances. The default class uses the following properties. |
| `stxx.attach.resources.include` | Matches which requests to attach resource XML |
| `stxx.attach.resources.exclude` | Matches which requests not to attach resource XML |
| `stxx.attach.requestParameters.include` | Matches which requests to attach request parameters XML |
| `stxx.attach.requestParameters.exclude` | Matches which requests not to attach request parameters XML |
| `stxx.attach.requestParameters.ignore` | Matches which request parameters to ignore |
| `stxx.attach.requestAttributes.include` | Matches which requests to attach request attributes XML |
| `stxx.attach.requestAttributes.exclude` | Matches which requests not to attach request |

| | |
|---|---|
| | attributes XML |
| `stxx.attach.requestAttributes.ignore` | Matches which request attributes to ignore |
| `stxx.attach.requestParameters.exclude` | Matches which requests not to attach request parameters XML |
| `stxx.attach.errors.include` | Matches which requests to attach action errors XML |
| `stxx.attach.errors.exclude` | Matches which requests not to attach action errors XML |
| `stxx.attach.form.include` | Matches which requests to attach action form XML |
| `stxx.attach.form.exclude` | Matches which requests not to attach action form XML |
| `stxx.attach.messages.include` | Matches which requests to attach action messages XML |
| `stxx.attach.messages.exclude` | Matches which requests not to attach action messages XML |

XML attachment configurations allow you to have more control over what extra information is serialized and added to your XML document before transformation. For all information types, there will be two lines of configuration - include, and exclude. For the request attributes and parameters, there will be the additional "ignore" configuration line.

In order for information to be attached as XML, the absolute path of the requested URL must match the regular expression defined for the "include" and not the "exclude". Additionally, in the case of request attributes and parameters, the name of the parameter/attribute must not match the regular expression defined for "ignore".

The regular expressions are compatible with Perl5, for example:

```
stxx.attach.requestParameters.include=.*/public/.*
stxx.attach.requestParameters.exclude=.*/public/other/.*
stxx.attach.requestParameters.ignore=ID_.*
```

In this case, the transformation for the URL "/stxx/public/index.do" will have all request parameters that start with "ID_" attached to the XML but the URL "/stxx/public/other/index.do" will not. In forming your regular expressions, please note to use a "\", you must type "\\" due to the properties format. Therefore to match a space, type "\\S".

## 4. References