Developer Guide

Last update: 23.11.2010 13:56:58

Authors:

Joern Turner <joern.turner@betterform.de>

Lars Windauer <lars.windauer@betterform.de>

Version: 1.0

DeveloperGuide

1 Preamble

The Developer Guide is targetted at developers that want to integrate betterFORM into their own applications and/or write extensions that are not provided by the standard distribution.

It covers everything from building and setup of the different deployment scenarios to customization and the various extensibility features. It will further feature an introduction to the Java API of the Core XForms implementation.

The guide can be read chapter by chapter or used as a reference to be consulted for a specific use case like running betterFORM within a cross context scenario.

2 Introduction

> „Forms are an important part of the Web, and they continue to be the primary means for enabling interactive Web applications. Web applications and electronic commerce solutions have sparked the demand for better Web forms with richer interactions. XForms 1.0 is the response to this demand, and provides a new platform-independent markup language for online interaction between a person (through an XForms Processor) and another, usually remote, agent. XForms are the successor to HTML forms, and benefit from the lessons learned from HTML forms." [XForms 1.0 W3C Candidate Recommendation 12.11.02, 1.1]

HTML Forms once made the Web interactive, allowing users to talk back to the servers they visit. Since then whole industries have built their profits on HTML forms as they have existed since the very beginning of HTML (Hypertext Markup Language). Many technologies have evolved around and build on top of the possibility to interact with users via the Internet.

Despite the immense impact HTML Forms have had on the development of the Web, years of use have shown that they are not suitable for today's applications any more. Missing validation, data-typing and logic-components must be compensated with scripting which itself leads to bad maintainability and increasing development cost. The arrival and growing popularity of AJAX has not improved this situation. Though a huge variety of cross-browser libraries is available there's also a lack of standards about the details of data formats, typing, submission control, events, validation and calculation etc. etc.

XForms is a open standard[1] that addresses all these issues bringing rich interaction, a well-defined processing model and universal data management to improve the situation. It's based on XML and therefore easy to handle, it offers self-describing data structures and is adaptable to any application or environment.

But there are things belonging to a full-flegded application that XForms intentionally leaves out: specifically layout and the concrete control or widget to the used to interact with the user. Implementations have to choose which widget set or UI markup they want to support on a specific platform. As XForms is platform- and device-independent no assumptions about specifica only available in a certain of client can be made. E.g. a voice client is not interested in layout altogether.

Therefore betterFORM on the one hand provides an implementation of the XForms standard but does more by providing a toolkit of controls and layout containers to build complete, dynamic web applications. It further wraps the whole thing in a non-

1 XForms 1.0 is a W3C Recommendation since 29 October 2007. XForms 1.1 is a final recommendation since 20 Oct 2009.

intrusive architecture that's easy to use with any existing application and is especially well-suited to work with applications that use XML as data format.

This guide covers the more advanced topics like building from source, development tools, building your own controls, integration with your application and of course the various possibilities of extending the existing functionality.

2.1 What is XForms?

XForms 1.1 is a W3C recommendation since October 2009 and the successor of HTML forms. It addresses the limitations of HTML forms and adds common functionality found in most form-applications like logic, validation, calculation and data-typing.

Today a diversity of frameworks is used to circumvent the weaknesses of web forms. These are either proprietary and/or commercial which results in vendor lock-in and raises the overall cost-of-ownership. While server-infrastructures like J2EE and .NET have evolved and established industry standards for nearly all requirements, the form-processing part will still be hand-coded over and over again [2].

Once XForms is adopted this picture will change drastically. Forms will be portable from one server to another, serve different clients from a single source, provide rich validation and calculation and allow complex logic without writing a single line of scripting code. Furthermore XForms is maintainable through clear separation of logic, data and presentation which allow a role-based development process.

2.2 What is betterFORM?

betterFORM is a XForms toolkit that allows to quickly build state-of-the-art web applications in a declarative way. It evolved around a XForms Processor implementation in Java (commonly called the 'betterFORM Core'), has an AJAX layer to speak to common browsers and provides a UI toolkit for dynamic, accessible and consistent widgets.

betterFORM fills the aforementioned gaps that XForms leaves out. You can use Dojo layoutcontainers to build web applications that use a desktop-like interface with toolbars, sizeable content panes, pop-up dialogs and many fully accessible controls. Forms can be assembled in a modular way into fully interactive and dynamic user interfaces to build complete, AJAXified web applications.

2 There are some form extensions in some framework but these don't reach far enough when it comes to dependency tracking, datatyping and openness.

In combination with eXistDb (http://exist.sourceforge.net) you get a full stack of XML tools and technologies that enable to use the newly emerging XRX architecture [3] that uses XML from front to back without the need of any object-mapping or conversion. This leads to an elegant and efficient way to quickly build even complex web applications with the help of XForms, REST and XQuery.

3 See http://en.wikibooks.org/wiki/XRX

3 The Architecture

The following sections are targetted at architects and developers who are interested in using betterFORM in their own applications and give some insights about the inner workings.

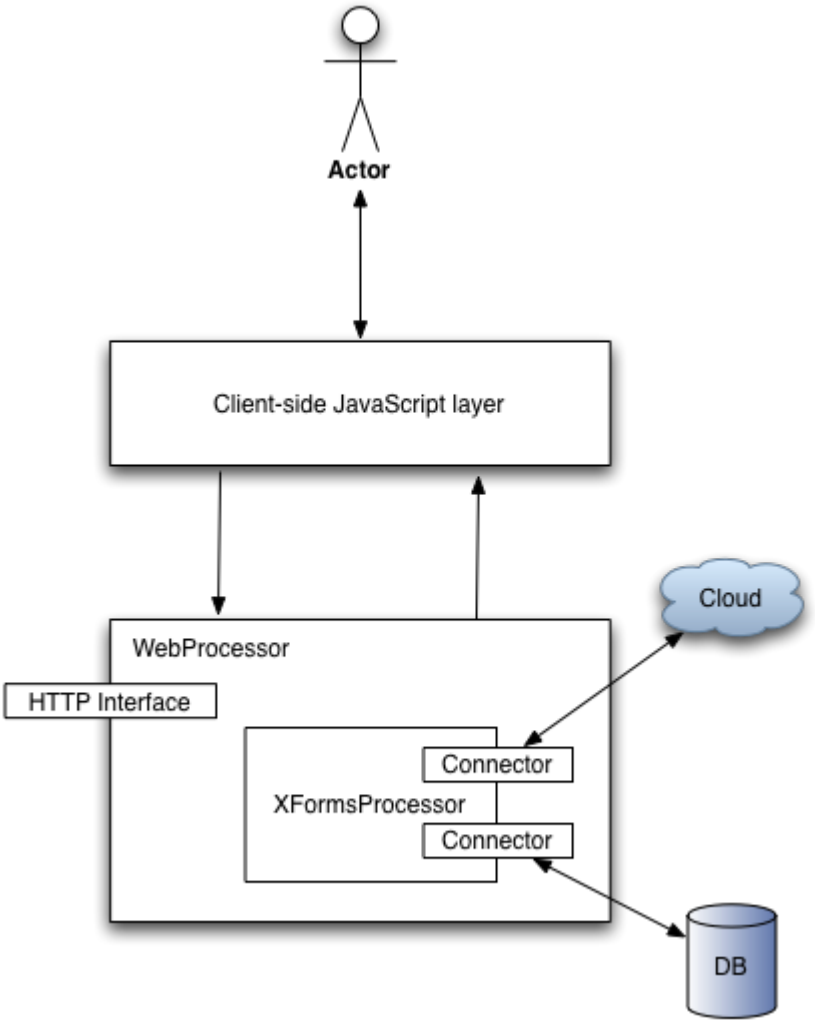
3.1 Applicability

Over the years the codebase has been used in many different industries and applications:

- Aeronautical Information Systems
- Banking
- eHealth
- eBusiness
- Inter-, Intra- and Extranet Applications
- Customer Relationship Management Solutions (CRM)
- Content Management Systems (CMS + ECM)
- eGovernment
- Workflow Applications and BPM
- Riscmanagement
- Knowledge Mangement
- applications that require multi-client support
- as an interface to Webservices in SOA applications
- as a configuration tool for appliances

3.2 System components

This section describes the main building blocks of the betterFORM architecture. This is a bird-eye view of the processor itself and its connections/interactions with the embedding environment.



BetterFORM can be used in different scenarios but the above diagrams focusses on the most one:

the XFormsProcessor lives on the server-side and runs either in a standalone server or as a typical web application (WAR file) on top of any web container. On the client-side Dojo is used to provide the user interface and handle the communication with the server. The XFormsProcessor represents the implementation of XForms 1.1.

It is wrapped (Decorator pattern) by a WebProcessor class that handles the communication with the client via the HTTP protocol.

On the client the Dojo toolkit is used for the rendering of controls, for creating application-like layouts and for keeping state of the user interface in sync with the server-side XForms document at runtime.

3.2.1 The XForms Processor

The core of the betterFORM XForms implementation is represented by the `XFormsProcessor` interface. It is implemented in `XFormsProcessorImpl`. This implementation may be used in any Java application and offers all methods to instanciate and process a XForms document.

With regard to the MVC pattern the `XFormsProcessorImpl` takes the role of the XForms controller and at the same time acts as a facade to the processors' functionality.

Once inited, `XFormsProcessorImpl` holds a reference to a `Container` instance which is the DOM representation of the XForms document and the model in our MVC architecture. All runtime manipulations on the `Container` are carried out by dispatching DOM events from `XFormsProcessorImpl` to the `Container` through the `dispatch()` method of the XFormsProcessor. It will take the id of the target element and the eventtype as arguments and feeds an appropriate event instance into the internal XForms DOM.

All other methods are used to configure the processor before calling init() or to `shutdown()` the processor.

The main integration as a webapplication follows the Decorator pattern and can be found in the `WebProcessor` class. This class encapsulates the specifics of the Servlet API and allows the configuration of the processor from a web environment e.g. the passing of initlal params.

3.2.2 The UIGenerator

The `de.betterform.generator.UIGenerator` is an optional component in the architecture of betterFORM. It's normally found only in server-side integrations though there might be cases where it makes sense on a client too (e.g. when generating print versions of your forms).

It is responsible for transcoding a XHTML/XForms document into XHTML + JavaScript which can be served to browsers that have no builtin XForms support. Currently betterFORM uses `XSLTGenerator` as an implementation of `UIGenerator` to handle the transcoding part. It would be possible to use an alternative template language for this task.

3.3 Processing Model and Lifecycle

The lifecycle of the processor divides in three phases

1. At init the processor will setup the given XForms along with all referenced instancedata and build an internal object representation of the XForms document. This will include a representation of the host document along with all models and the UI as well as one independent DOM for each instance defined by the model(s). Additionally each model will build a dependency graph to track validations and calculations. During init phase the event `xforms-model-construct`, `xforms-model-construct-done` and `xforms-ready` are fired as defined by the XForms spec.
2. After init is done the user interface will be generated and the user starts to interact with the UI to input and manipulate data until all entries are complete and correct. These interactions trigger actions in XForms or set instancedata nodes to new values. The processor will execute all these in conformance to the XForms processing model as defined by the Specification. E.g. some actions require the model to be revalidated to ensure consistency. The processor sends notification events for all changes of the processor state which will be used to update the client.
3. The processing phase ends when the user submits the form or loads a resource with the 'load' action. To be exact this does only happen when a submission is declared with replace="all" or the load action was used with show="replace". Both options use the current display context to show new content returned from the submission or load respectively. The processor will then call `shutdown()` on `XFormsProcessor` to cleanup resources. In the web context all session resources will be clean up additionally.

4 Getting started

This chapter helps getting started with betterFORM development. Please note that this guide assumes some familiarity with technologies like CSS, XPath, XSLT, DOM, JavaScript and of course XML.

This guide is meant for developers that like to extend betterFORM classes, integrate their own JavaScript components, write connectors to read/write data from arbitrary sources or implement their own XPath functions in case the provided feature set is not sufficient.

For questions regarding the authoring of XForms and the usage of betterFORM as a XForms processor please refer to the UserGuide.

4.1 Development Requirements

If you'd like to build the betterFORM sources yourself or make use of the tools betterFORM provides you'll need installed on your system:

- Apache Ant 1.7 or higher
- Java 1.5 or higher

Optional requirements are

- Apache Maven 2.1 or higher installed on your system
- Any Subversion client you prefer to download and update the sources

4.2 Getting the Source

There are various ways to download the betterFORM source from the public Subversion repository. It can be downloaded directly via HTTP or by using a Sub - version client[4] of your choice. The repository is located at

https://betterform.de/svn/betterform

4 http://en.wikipedia.org/wiki/Comparison_of_Subversion_clients

4.2.1 Public Repository Structure

SVN repository organisation:

trunk	The trunk represents the head (latest) version of the betterFORM source code and is used for ongoing development.
tags	If a trunk version is released it will be tagged following with 'R-' following the version number.
branches	Branches are usually copies of the latest trunk version and used for major implementations like new core XForms functionalities, redesign of betterFORM components as well as developing new XForms rendering possibilities next to the web version. • For external contributors not belonging to the core team we open up branches on demand. This way these can work independently and their work is merged back into trunk when stable. If you are interested in having your own branch please contact us via the mailinglist.

4.2.2 Downloading betterFORM over HTTP

Note: Downloading the sources over **HTTP** will give you a **snapshot** of the bettterFORM sources. There is possibility of updating the sources other than downloading them again and to merge any changes you have made by hand.

To **enable automated updating** go on reading with the following chapter Downloading betterFORM by Subversion

To download betterFORM by HTTP point your browser to the URL:

https://betterform.de/betterform/browser

and step into the root of the betterFORM version you want to download.

To download the latest betterFORM version

1. point your browser to https://betterform.de/betterform/browser/trunk

2. click on the Link „Zip Archive" at the very bottom of the page and the actual trunk revision will be packed and provided as Zip file.
3. Unpacking this Zip file will create a directory containing the betterFORM sources

4.2.3 Downloading betterFORM by Subversion

The betterFORM repository is located at https://betterform.de/svn/betterform and can be accessed by any Subversion client.

If you have a command line client installed please type the following at the prompt:

```
svn checkout https://betterform.de/svn/betterform/trunk betterFORM
```

to download the latest betterFORM trunk version. The sources will be download into the directory betterFORM specified as last argument of the svn call.

4.2.4 Source Structure

BetterFORM is organized into several modules which correspond to directories in the root of the source tree:

- **src:** provides general documentation about betterFORM (***main/docs***), various demo, reference and test forms (***main/xforms***) as well as XSLT, JavaScript, Image and CSS resources (***main/resources***) and JavaScript libraries (***main/lib***) used by the web and convex module. Further the official XForms conformance test suite and their matching HTML Selenium tests for automated testing are placed within the directory ***test***.
- **core**: contains the core XForms Processor implementation in Java
- **web**: contains the integration of the betterFORM Core into a web application
- **xApp**: entry point to create projects based on betterFORM. This is explained in detail in chapter 5.2 xApp – using betterFORM in webapps.
- **zaphod**: is a separate module to deploy the betterFORM web application next to your existing one and to use betterFORM as a service within the same servlet engine. See chapter 5.3 Zaphod - betterForm As Service (Cross Context Environment) for more details
- **convex**: still here for historical reasons. Might be revived one day if someone wants an applet running XForms
- **tools**: contain several tools for XForms/betterFORM development

The whole betterFORM directory structure is inspired by Maven. Each module provides a src directory. The typical structure of a module is folder is

src /main	/doc	Manuals, Release notes and other information regarding the project or module.
	/license	License files regarding betterFORM itself and its dependent libraries.
	/java	Java sources
	/lib	Libraries for the specific module. Note that modules like web depend on the core module. Libraries are not added twice but placed within the base module.
	/resources	All kind of resources belonging to the respective module. This might be config files, schemas or any other kind of resource.
	/webapp	All files within the webapp directory are deployed as root folders of the web application.
/test	/java	JUnit tests and/or functional tests for the module are located here. These can be executed automated by Apache Ant, Maven or most common integrated development environments (IDE) like IntelliJ IDEA, Eclipse or NetBeans.
	/resources	Test resources used by the JUnit tests. For instance XForms which are manually loaded into the processor and tested for different purposes.
/target		The target folder is created dynamically during the build process. Resulting Jar files, exploded or packaged web applications as well as test results can be found here depending on the executed development step.

4.3 Building betterFORM

betterFORM uses Apache Ant as its primary build tool and provides all targets needed to support the development cycle as well as building source and binary distributions.

Note:
As an option for the Maven 2 enthusiasts there is also support for Maven 2.x but all dependencies for both Ant and Maven still rely on a single file: `build.properties.xml` which is found in `BETTERFORM_HOME`. This file lists all relevant library dependencies.

There are additional Ant targets in `BETTERFORM_HOME` which generate the correct Maven pom.xml files from `build.properties.xml` by using a XSLT transformation.

All build products are created in a directory named 'target' that is created in the root directory of the respective module e.g. when executing compile for module 'core' the compiled classes will be in `BETTERFORM_HOME`[5]`/core/target`. If the `'target'` directory does not exist it is created.

Each module has its own Apache Ant build.xml file providing at least the following deployment targets:

Ant Target	**Description**
`clean`	delete the 'target' directory of the current module
`prepare`	copy all resources in place needed for compilation of the module.
`compile`	Compile Java classes of the module to 'target/classes'
`package`	Create the result the module is intended to. This might be the core Jar file or the betterFORM web application archive (WAR) file.
`distribute`	Create a distribution version of the current module.
`test`	execute all test of the current module

Note:
This guide will focus on building the 'core' and 'web' modules but as the approach is generic the general hints given will also apply to the other modules.

5 BETTERFORM_HOME means the root directory of your local betterFORM source installation.

4.3.1 building the Core

When building module 'core' a jar file containing the interfaces and implementation of the XFormsProcessor will be created in 'core/target'. It will only contain the 'naked' processor without any supporting classes for web development.

1. cd into the betterFORM 'core' directory
2. execute the following Ant command

```
> ant
```

This will execute the default target `package` and create a betterFORM Core jar file in 'core/target'.

Here's a complete list of the public Ant targets:

Ant target	**Description**
`clean`	cleans the 'target' directory
`prepare`	prepares the 'target' directory + copies resources
`compile`	compiles the Core classes
`distribute-sources`	creates a source archive for releases
`distribute-binary`	creates a binary archive for releases
`distribute`	calls the two targets above in one go
`doc`	create JavaDoc
`package`	creat a jar file from betterFORM Core
`test`	run all unit-tests and generate a report

4.3.2 building the betterFORM Web Application

1. cd into the betterFORM 'web' directory
3. execute the following Ant command

```
> ant
```

This will execute the default target '**package**' and create a betterFORM WAR file at 'web/target'.

Here's a complete list of the Ant targes:

Ant target	**Description**
`clean`	cleans the web 'target' directory

`prepare`	prepares the 'target' directory + copies resources (including betterFORM core)
`compile`	compiles the Web classes to web/target/classes
`deploy-resources`	copies demo / reference / test XForms into the forms directory of the exploded webapp
`deploy-test-resources`	deploy the XForms 1.1 conformance test suite into the forms directory of the deployed web application. The tests can be browsed at http://localhost:8080/jsp/forms.jsp?forms/ XFormsTestSuite1.1E1
`deploy-to-servlet-container`	deploy the exploded webapp including all demo and reference forms to the configured servlet container. The container can be configured in BETTERFORM_HOME/build.properties.xml. Simply adjust the path specified in webappPath to your local servlet container webapp directory.
`distribute`	assembles binary distributions of the web module
`doc`	create JavaDoc
`exploded-webapp`	creates the exploded web application within the 'web/target' directory.
`package`	create a jar file from betterFORM Core

See the betterFORM User Guide on details about running the web application.

4.4 Setup a betterFORM Project

This chapter describes how to setup your favorite build tool or IDE.

> Note:
> This step is optional. To build the betterFORM modules you don't necessarily need an IDE. You can simply use the command-line to execute the relevant Ant build targets.

The easiest way to set up a project is to use any IDE working with Maven. If your favorite IDE does not provide a mechanism to work with Maven the project can still be setup by hand.

4.4.1 Setup betterFORM using Maven

The advantage of setting up the project with Maven is, that all dependent libraries and output pathes will be correctly created by Maven. Before creating a project execute the following Apache Ant target in the betterFORM root directory to install all libraries into the local Maven repository:

```
> ant install-maven-dependencies
```

The next steps vary depending on the IDE:

Intellij IDEA / NetBeans

IntelliJ IDEA 9.x and NetBeans 6.x natively support Maven 2 based projects. Simply choose 'open project' from the file menu and navigate to the Maven project file (pom.xml) placed in the root of your local betterFORM directory. That's it, your project is setup.

Eclipse

Eclipse can't natively handle Maven based projects. But there are several plugins to enable Maven 2 support in Eclipse. See http://maven.apache.org/eclipse-plugin.html for a complete list. If everything is setup follow the instruction of the specific plugin to open the Master Maven 2 project file (pom.xml) found in the betterFORM root directory.

4.4.2 Setup betterFORM manually

As IDEs have varying mechanisms to setup and manage projects we can't provide a detailed description for each of them. But here some general tips: note that all pathes are relative to the betterFORM project root.

- Libraries a respective module depends on can be found in:
  - core/src/main/lib
  - core/src/test/lib
  - web/src/main/lib
  - convex/src/main/lib

- Be aware that the `web` as well as the `convex` module depend on the `core` module.
- The main resources for the web module can be found in
  - web/src/main/webapp — including WEB-INF and JSP directory. The WEB-INF directory contains the webapp deployment desriptor web.xml and all other config files.
  - src/main/resources — JavaScript, XSLT, CSS and other resources used by web and convex module
  - src/main/xforms — Demo, Reference and Test XForms

5 Using betterFORM in applications

There are several ways to use betterFORM in or with your own applications:

a) using just the Core XForms implementation packaged as jar

b) extending the betterFORM webapp

c) deploy a war file side-by-side with your webapp and using betterFORM as a service by forwarding all XForms requests to it

d) use betterFORM in conjunction with eXist. You can use the installer to setup a pre-configured installation that is ready-to-go and allows you to build XRX applications

Regardless of the option you choose you can download a released version or checkout the latest sources from trunk and build the binaries yourself. This is described in the preceding chapters.

5.1 betterFORM Core

For some applications it might be interesting to integrate a complete XForms implementation. This might be especially the case for fat-client applications that have complex dependencies to track in their data or have very dynamic user interfaces.

To use the core XForms implemenation a binary distribution should be used as it also contains the dependent libraries that are needed at runtime. After unpacking you find the betterFORM jar plus a subdirectory named 'lib' on your disk. All jar files need to be added to your classpath to enable the XForms processor.

The file 'betterform-config.xml' contains all configuration related to the betterFORM processor. Normally you don't need to touch this file unless you have very specific requirements. All entries in this file have a short description.

The following example demonstrates one possibility to initialize the processor with a XForms document and to activate a trigger within that document:

```
1 Config.getInstance("file://path/to/betterform-config.xml");
2 XFormsProcessor xfProcessor = new XformsProcessorImpl();
3 xfProcessor.setXForms(
      new URI('file://path/to/forms/XFormsApplication.xhtml'));
4 processor.setBaseURI(new URI('file://path/to/forms'));
5 processor.init();
```

```
6 processor.dispatch("trigger-process-data", DOMEventNames.ACTIVATE);
```

Before initializing the configuration file has to be loaded (line 1). In line 2 an instance of the XFormsProcessor is created. The processor provides several `setXForms` methods to load a XForms document. In this case a form document is loaded via a 'file' URI (line 3). In line 4 the baseURI for the document is set. This is important for all URI resolutions taking place from within the XForm document e.g. resolution of `<xf:instance src="aURI">` or all `submission` and `load` URIs. All links will be resolved relative to this location unless the are absolute.

In line 5 finally the init() method is called that kicks off the processor and starts XForms processing according to the XForms spec. Finally in line 6 we simulate a user clicking a button by dispatching a DOMActivate to a trigger with id 'trigger-process-data'.

5.1.1 Loading a form

There are several ways of loadiing a XForms host document into the processor. You may use one of the following methods:

`setXForms(InputSource source)`	takes a SAX InputSource as input
`setXForms(InputStream stream)`	Uses an InputStream which contains the XForms
`setXForms(URI uri)`	resolves the URI and loads the result as XForms document
`setXForms(Node node)`	Takes a DOM Node which should hold the entire XForms document.

5.1.2 Setting the base URI

An important step of the processor configuration is the setting of a `baseURI` which is used for resolving any URI used within betterFORM and the loaded XForms.

Wether you want to load a form by URI, load instance-data from external sources or execute a 'load' action – in all these cases the processor uses the `baseURI` to resolve relative pathes to absolute ones.

E.g. when running betterFORM Web the `baseURI` is the URI of the requested form and therefore always represented by a http url. All relative references in the form are resolved against this location meaning that a `<xforms:instance`

`src="data/foo.xml"../>` will resolve to a http Url pointing to a subdir 'data' under your forms dir. The `ServletAdapter` sets the `baseURI` during init phase.

The standard procedure to set the base URI is to call:

`setBaseURI(String uriString)` where `uriString` must contain a String that parses into a URI object.

> Note: please pay attention when using dynamically generated XForms which you pass directly as a DOM Node. In this case no baseURI exists and you've to set it manually to take care that resource resolution works.

5.1.3 Attaching event listeners

If you extend some Adapter or build your own you will be interested in registering eventlisteners with `XFormsProcessor`. The processor communicates changes to its internal DOMs by firing DOM Events at defined stages of the processing. An Adapter can register as `Eventlistener` (implementing `org.w3c.dom.events.EventListener`) and get notified of the specific changes taking place. You can register youself as listener for all eventtypes defined in XForms 1.1. Please refer to chapter 4 'Processing Model' of the XForms 1.0 recommendation for details (http://www.w3.org/TR/2003/REC-xforms-20031014/slice4.html).

betterFORM defines some additional non-standard events which make life easier for an Adapter implementor. To see what currently supported you should study the source of de.betterform.`xml.xforms.events.EventFactory` along with the javadocs.

To be able to process the events from the processor your Adapter must:

1. implement `org.w3c.dom.events.EventListener` by providing a `handleEvent` method
2. call `addEventListener` to attach yourself as listener for a specific eventtype when starting processing.
3. Call `removeEventListener` to cleanup when you stop listening (normally during `shutdown`). This is IMPORTANT cause otherwise you'll likely produce leaks.

5.1.4 Initializing the processor

That one is easy. After you've setup the processor simply call:

```
xformsProcessor.init()
```

That's all.

5.2 **xApp** – using betterFORM in webapps

In the betterFORM sourcetree there is a directory called 'xApp'. This is a specialized module for developers that want to build web applications with betterFORM. In this scenario developers typically do not want all the webpages and resources that are not strictly needed for bettterFORM to run.

The xApp module provides exactly that and additionally gives you a separate build file to deploy your project files as part of the resulting web application.

The name of the web application created by the xApp is configurable within the build.properties.xml file located at the betterFORM project root. The output name of the resulting web application can be changed by editing the <xapp/name> node node.

To work with the xApp module you should place your forms in xApp/src/main/webapp/forms directory. You can add custom directories as you want e.g. to add project-specific images or CSS styles. These will be copied to the target before a war file is created.

The betterFORM xApp module provides the following main Ant targets

clean	clean build target directory
deploy-resources	copy all webapp resources to the exploded webapp within the target dir
create-war	creates a not optimized war-file for interims release and development purposes. Depends on deploy-resources
package	builds an optimized Web Application Archive - with compiled JavaScript

Java sources placed within xApp/src/main/java will be compiled and deployed into the classes directory of the created web application. Likewise all resources placed beneath xApp/src/main/webapp will be copied to the target web application into the respective directory.

To overwrite existing betterFORM resources place them into excatly the same structure as they are deployed in the target directory. E.g. to overwrite the betterFORM CSS theme create a file betterform.css in:

xApp/src/main/webapp/resources/styles

Any resource in xApp that matches the name of a original file from the web module and is found in the same directory structure will overwrite the original resource. This is an easy way to make small patches or extensions to the standard betterFORM web application.

5.3 **Zaphod** - betterForm As Service (Cross Context Environment)

This chapter describes how to use betterForm from within a different servlet container context. It allows to store or generate forms in your application context and let betterForm process the form in its own context.

5.3.1 Advantages

- Easy and independently updating the software in the separated contexts.
- Usage of the same software library in different versions. Since both software stacks are loaded from different class loaders the do not interfere each other.
- Creating XForms from within your web application. The created form is processed in the context of betterForm.

5.3.2 Disadvantages

- Complex configuration

5.3.3 Prerequisite

- The setup can only be used in one servlet container. Both contexts have to be deployed on the same servlet container. Distributed systems are not supported at the moment.
- The servlet container must allow to access the context of betterForm. Most servlet container allow this by default. Apache needs to be configured separately. The current distributions of betterForm already contains a context.xml file which allows other contexts to access the context of betterform.
- The distributed filter class muss be installed in the context which accesses betterForm. An Jar file is (not yet) distributed.

- The filter must me configured correctly. The filter is configured in the deployment descriptor of the web application.

5.3.4 Configuration

All the configuration is done in the deployment descriptor. betterForm itself is already configured for this configuration. In the deployment descriptor one has to specify:

- the URL for the filter.
- the context name of betterForm.
- and URL which points back to the applications context

The filter is mapped to an URL pattern. If a request matches the pattern the returned page is send to betterForm and processed as an XForm form. Under the URL one can store forms in files or servlets which generate forms.

The name of the betterForm context is configured through the filter parameter xforms.engine.webcontext. It should contain the name of the context as string (defaults to 'betterform').

In the filter parameter `xforms.engine.resource` one must specifies an URL which points back to the filter. The url has to be appended with an virtual directory. This allows the filter to handle and forward requests which are resources within the betterForm context. This applies to css, scripts and images distributed with betterForm.

Additionally one can configure the servlet which processes forms in betterForm. It is not recommended to change this variable. It is intended for development.

5.3.5 Examples for Apache Tomcat

The following example files should work with Tomcat 5.5.x and Tomcat 6.x.

Context.xml

This file is only needed when betterForm is used together with Tomcat. It should be stored in the directory META-INF under the name context.xml. When first deployed Tomcat copies it to its own directory structure. Changes after the first deployment are not considered.

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

<Context antiJARLocking="true" crossContext="true" path="/betterform"/>
```

Applications WEB.xml

This file shows a minimal setup. The setup renders all forms in the `forms` directory with the help of betterForm into an html page. The setup specifies the context in which betterForm is running. Its important that the path of `xforms.engine.resources` is set to the same directory as the patter of the filter.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <description>..</description>
  <display-name>Sample Application</display-name>
  <filter>
    <filter-name>CrossContextFilter</filter-name>
   <filter-
class>de.betterform.agent.web.filter.CrossContextFilter</filter-class>
    <init-param>
      <param-name>xforms.engine.webcontext</param-name>
      <param-value>betterform</param-value>
    </init-param>
    <init-param>
      <param-name>xforms.engine.servlet</param-name>
      <param-value>/repeater</param-value>
    </init-param>
    <init-param>
      <param-name>xforms.engine.resources</param-name>
      <param-value>/SampleApp/forms/forward</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>CrossContextFilter</filter-name>
    <url-pattern>/forms/*</url-pattern>
  </filter-mapping>
```

```
</web-app>
```

5.3.6 Status

The current status of this configuration is experimental. We have not yet used this setup with any of our clients. This is the first public release. So be warned!

In the current version the code should allow authentication. It does not support the configuration of betterForm with absolute URLs.

The configuration of the filter in the WEB.xml is quite complex and error prone. I would be much better to generate the path to betterForms internal resources.

6 Extending betterFORM

6.1 custom XPath Functions

TBD

6.2 Connectors

Connectors allow to access arbitrary datasources through URIs. At init (except for very simple forms) and at submission time a XForms processor needs to resolve external URIs to load or submit instance data. The `de.betterform.connector` package defines two main interfaces that offers this capability (both extend `de.betterform.connector.Connector`):

- `de.betterform.connector.URIResolver`
  will be triggered for each 'src'-attribute found on a xforms:instance element
- `de.betterform.connector.SubmissionHandler`
  will be triggered by the 'resource'-attribute on element xforms:submission

At init time the processor looks for 'src' attributes on each instance element found in the document. If found the URI is passed to the `ConnectorFactory` which creates the needed `URIResolver`. The type of the resolver is determined from the URI scheme used. E.g. if the URI begins with 'http://...' the `ConnectorFactory` will instanciate a `HTTPURIResolver`, connect to that Url, fetch the data and return them as a DOM-tree.

Likewise at submission time the 'action' attribute of the relevant `xforms:submission` is examined, the instance data get validated, serialized and send along to the wanted URI.

All connector classes are configured in the betterFORM configuration file making it easy to add new custom connectors which use their own scheme. E.g. if you'd like to access a database you might choose to build a `DBResolver` and `DBHandler` pair and use the scheme 'db:' in your URIs.

An example entry might looks like this:

```
<submission-driver scheme="db"
class="de.betterform.xml.xforms.connector.db.DBSubmissionDriver"/>
```

Connectors are the most common gateway between the XForms and 'non XForms world'. The User Guide lists several helpful connectors provided by betterFORM to load / save data e.g. via http(s) / file or xmlrpc protocoll or to process them using XSLT. This section describes how users can extend existing connectors to tailor them to their needs and to write custom connector.

6.2.1 URI Resolver

The following sample illustrates how the betterFORM config is used for the mapping between the URI Scheme and the adequat URI Resolver.

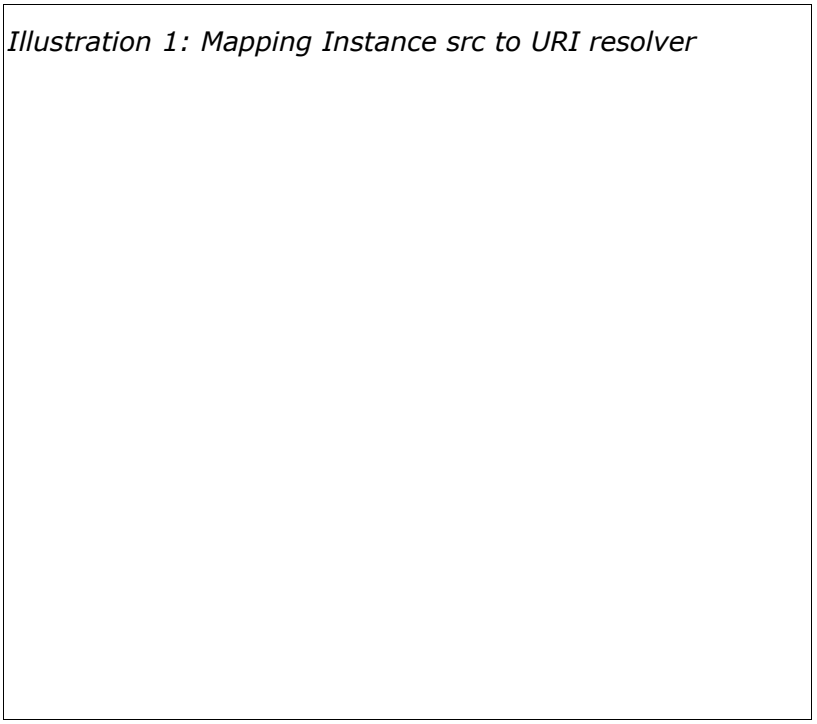


Illustration 1: Mapping Instance src to URI resolver

To write your own URI Resolver you have to

1. write your own URI Resolver Java class which
    1. implements the URIResolver interface and
    2. extends AbstractConnector
2. implement the missing public Object resolve() function. You can gear yourself to existing URIResolver like the File- or HTTPURIResolver. Although the return

type of the resolve() function is Object you have to return a DOM Node as result.

3. Add your new created URI Resolver to the betterform-config.xml
<uri-resolver scheme="my" class="my.CustomURIResolver" />

After everything is set up, you can use CustomURIResolver within your form as shown below:

```
<xf:instance src="my://path/to/where/to/load/data/from/file.xml"/>
```

6.2.2 Submission Handler

Roughly speaking SubmissionHandlers work the same way URI-Resolver do. While the URI-Resolver reflects the instance src attribute, the Submission Hander is choosen by the Submission resource attribute as shown in the following sampe:

```
<xf:submission id="s-xslt-task"
                        resource="xslt:relative/path/to/Stylesheet.xsl
                        method="get" />
```

Analog to the URI Resolver, the betterform-config.xml is used to map the URI scheme 'xslt' to the XSLTSubmissionHandler.

To write your own SubmissionHandler:

1. write your own Submission Handler Java class which
    1. implements the SubmissionHandler interface and
    2. extends AbstractConnector
2. implement the missing function: 'Map submit(Submission submission, Node instance)'. You can gear yourself to existing SbumissionHandlers like the File- or HTTPSubmissionHandler. The result Map to return must contain the result stream or document:

```
...
Map response = new HashMap();
if(inputStream != null){
    response.put(XFormsProcessor.SUBMISSION_RESPONSE_STREAM,inputStr
eam);
}
if(resutDocument != null) {
    response.put(XFormsProcessor.SUBMISSION_RESPONSE_DOCUMENT,
```

```
    resultDocument);
    }
    return resonse;
```

3. Add your new created SubmissionHandler to the betterform-config.xml
   <submission-handler scheme="my" class="my.CustomSubmissionHandler"/>

To use the new created SubmissionHandler simply use the URI Scheme 'my' within the resource attribute of your XForms Submission.

```
<xf:submission id="s-my-submission"
                        resource="my:relative/path/to/Resource.xml
                        method="get" />
```

6.3 Schema Datatypes

Next to the standard Schema and XForms datatypes betterFORM provides the possiblity to embed custom types. This can be inline Schema types as well as Schema types within an external Schema file.

> Note:
> Be aware that only XML Schema Simple types are supported by XForms.

6.3.1 Inline Schema

The sample shows how an inline Schema is defined within the host document and referenced from the XForms Model:

```
<xf:model id="m-schema" schema="#s-schema">
   <xf:instance id="i-listing" xmlns="">
```

```
      <data>
        <login/>
      </data>
   </xf:instance>
   <xf:bind nodeset="login" type="login"/>
</xf:model>
<xs:schema id="s-schema">
   <xs:simpleType name="login">
      <xs:restriction base="xs:string">
         <xs:minLength value="4"/>
         <xs:maxLength value="8"/>
         <xs:pattern value="[A-Za-z0-9]+"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

6.3.2 External Schema

Embinding an external Schema works quite the same as embedding an internal Schema. Only the value of the schema attribute of the XForms model has to be adjusted. Insead of '<xf:model schema="#<internal_schema_id">' the schema is referenced relative or absolute: <xf:model schema="xsd/MySchema.xsd">. That's all, the schema types of MySchema.xsd can now (analog to the internal Schema) be used within the type attribute of XForms bind.

6.4 Useragents

Note:
keep in mind that this approach applies only to betterFORM web application processing. In a client-side setting on a standalone application it might be likely that you want to operate directly on the abstract Xforms markup without the need of any transformation. While the processor is ignorant about the presentation, the UI generation will always depend on the client to be served and has to be handled separately.

After the betterFORM procesor has initialized XForms Markup the resulting UI DOM is fed into the XSLT processor depending on the configured useragents. The Useragent defines which XSL Stylesheets is used to transform the abstract XForms markup the XForms processor creates to the specific target plattform. Currently betterFORM web module supports three different useragents:

Agent	XSLT Stylesheet	Description
dojo	dojo.xsl	The Web 2.0 useragent stylesheets renders HTML where every xforms ui control is represented by a dojotoolkit Dijit. Run with on optimized JavaScript resources which means that whitespace, new-line character, comments and debug output is removed and symbols are replaced with shorter names.
dojodev	dojo-dev.xsl	Same as dojo useragent but runs on not optimized JavaScript resources to enabled debugging.
html	html4.xsl	No Script HTML useragent (currently neglected)

By default the processor will use the dojo useragent configured within the web.xml at `web/src/main/webapp/WEB-INF`. You can switch to another useragent configured within the betterform-coonfig.xml during runtime by adding the URI param `useragent=<useragent.name>` to the form url.

To utilize a custom stylesheet reference it within the value attribute of the dojo useragent in betterform-config.xml. If you don't want to write a complete new useragent but do to fine-tune your HTML rendering or add some scripting use `<xsl:import href="dojo.xsl" />` at the beginning of your stylesheet to import the standard stylesheet. This option is often usefull to integrate with a certain environment e.g. by overwriting the 'body' or 'head' templates.

The mapping between useragent and XSL stylesheets is done within the betterFORM config file:

```
<useragents>
    <useragent name="html" value="html4.xsl"/>
    <useragent name="dojo" value="dojo.xsl"/>
    <useragent name="dojodev" value="dojo-dev.xsl"/>
</useragents>
```

To exchange the stylesheet for the useragent 'dojo' simply reference another stylesheet within the value attribute. All useragent stylesheets are placed at $PROJECT_HOME/src/main/resources/xslt.

Be aware that you exchange the existing useragent 'dojo' this way. The advantage is that nothin more is needed. The easiest way to change the useragent is to import dojo.xsl and to overwrite only the templates that need to change:

MyUseragent.xsl

```
<xsl:stylesheet version="2.0"
                xmlns:xhtml="http://www.w3.org/1999/xhtml"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xf="http://www.w3.org/2002/xforms"
                xmlns:bf="http://betterform.sourceforge.net/xforms"
                exclude-result-prefixes="xhtml xf bf">
    <xsl:import href="dojo.xsl"/>
</xsl:stylesheet>
```

To integrate the stylesheet simply change the dojo useragent config to: <useragent name="dojo" value="**MyUseragent.xsl**"/>.

If you have to optimize the user experience by sophisticated styling and reduction of server-turnarounds the stylesheets are the place to do so.

It would leed to far to describe how to implement a new useragent. But if thats what you want here a few hints where to look and what you have to consider.

1. add the new useragent to the betterform-config.xml file:
   <useragent name="my" value="myAgent.xsl"
2. take a look into WebFactory.java and WebProcessor.java and search for the expression 'useragent.equalsIgnoreCase'. Here all existing useragents are mapped to a XForms Processor implementation. Depending on your usecase you might want to use the FluxProcessor as well or write you own XForms Processor. Take a look into the Flux- and PlainHtmlProcessor to see how to write you own processor.

6.4.1 Dojo Useragent Stylesheets

TBD

6.4.2 DojoDev Useragent Stylesheets

TBD

6.4.3 PlainHTML Usermode Stylesheets

Note:this information applies to the html4.xsl, html-form-controls.xsl, ui.xsl stylesheets dated from 14.11.2005. These stylesheets are used in 'betterFORM Web' and Convex. They may serve as a blueprint for developing your own customized set.

betterFORM comes with three standard stylesheets that transform XHTML documents with XForms markup into HTML. The stylesheets are acting as a unit and handle both a non-scripted as well as a scripted mode. In scripted mode Javascript handlers are attached to the generated controls. These handlers are used to call functions of the XForms processor in the applet and AJAX integrations of betterFORM when a user makes changes through the UI.

6.4.4 html4.xsl

is the main stylesheet and used as an entry to point to the generation. It includes html-form-controls.xsl and ui.xsl and is responsible for building the complete output document structure (the host document) with `<html>`, `<head>` and `<body>`. CSS stylesheet or script references from the input document will be copied to the output. The generic xforms.css and eventual script libraries that are needed for AJAX processing are linked in.

While the model section of XForms is not copied to the output document, the stylesheet provides templates to match all XForms UI control markup including common childs (label, alert, hint, help). For every XForms control found the respective named template from html-form-controls is called. There one named

template for every XForms control. Foreign (not XForms) namespaced elements are copied to the output. XHTML1 namespaced elements are copied to plain HTML4.01.

Current standard output from this transform is HTML4.01 transitional. It is intended to provide additional support for XHTML1 in the future.

6.4.5 html-form-controls.xsl

transcodes XForms UI controls such as input, select1 or textarea into HTML controls that can be handled by any browser. These templates are all named and are used from html4.xsl to build complete HTML documents that match the XForms input.

As these templates are quite generic there should be not much need to patch or overwrite them unless you want to add scripting or change the overall behaviour. Though it's a good place to put new custom controls.

6.4.6 ui.xsl

handles the XForms UI constructs 'group', 'repeat' and 'switch'. Different appearances are offered for all of them and give the form author a convenient way of applying a generic layout such as a table or tabsheet.

Currently betterFORM uses the HTML 'fieldset' and 'legend' elements to build groups of controls. This allows for 'lighter' markup than use of tables and offers a lot of freedom to style the form with CSS. The file 'xforms.css' that comes with betterFORM shows examples of applying CSS to the generated page.

6.5 Javascript Components

This section describes how to map an xforms:input control to a custom JavaScript control (called dijit in Dojo slang). This will be illustrated by the the example of the DateTime dijit rendered for any xforms input control bound to a value typed as dateTime.

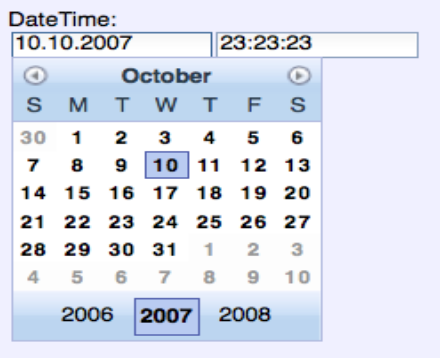


Illustration 2: An input bound to a dateTime value

```
…
<body>
 <xf:model>
   <xf:instance xmlns="">
     <dateTime>2007-10-10T23:23:23Z</dateTime>
   </xf:instance>
   <xf:bind nodeset="/dateTime" type="dateTime"/>
 <xf:model>
 …
 <xf:group>
   <xf:input ref="/dateTime">
     <xf:label>DateTime:</xf:label>
   </xf:input>
 …
 </xf:group>
</body>
```

Before creating a custom JavaScript control some basics are needed. Any XForms UI Control consists of the control itself, carrying the current model item property state (readonly / readwrite, enabled / disabled..) as CSS classes and its label and value.

Optionally controls can have a xforms:alert, xforms:hint and / or xforms:help child.

The following sample shows some typical DateTime Control HTML markup generated by the dojo.xsl stylesheet

```
<span id="dateTimeDijit"  dojoType="betterform.ui.Control"
```

```
      class="xfControl xfInput xsdDateTime xfEnabled xfReadWrite
xfOptional xfValid
             xfIncremental">
    <label for="dateTimeDijit-value"
           id="dateTimeDijit-label"
           class="xfLabel">DateTime:</label>
    <div id="dateTimeDijit-value" class="xfValue" dataType="dateTime"
         controlType="input" appearance="" name="d_dateTimeDijit"
         incremental="true" tabindex="0" accessKey="none"
         schemaValue="2007-10-10T23:23:23.123Z">Oct 10, 2007 11:23:23
PM</div>
</span>
```

For each XForms UI Control a betterform.ui.Control dijit is created to handle the common XForms logic. While the label is not represented as dijit the value is. Each betterform.ui.Control is responsible to create its value via the UIElementFactory. Here the dateTimeDijit-value is passed to the factory to create a new DateTime dijit.

But before a new dijit can be created it has to be implemented.

The first step to write a custom JavaScript control is to create the JavaScript class representing the dojo dijit. The DateTime dijit used as example can be found beneath the script folder ($PROJECT_HOME/src/main/resources/scripts) at betterform.ui.input.DateTime.js.

DateTime.js raw body:

```
dojo.provide("betterform.ui.input.DateTime");
dojo.require("betterform.ui.ControlValue");
dojo.require("dijit.form.DateTextBox");     // displays the Date part
dojo.require("dijit.form.TimeTextBox");     // displays the Time part
dojo.declare(
        "betterform.ui.input.DateTime",  // defines DateTime dijit
         betterform.ui.ControlValue,        // which extends
ControlValue
  {
    ...
  }
);
```

The UI markup for the dijit is placed wihtin an external file DateTime.html:

```
<div class="xfDateTimeControl">
  <input type="text" dojoAttachPoint="dateFacet" value=""
           class="xfValue xfDateTextBox"
           dojoType="dijit.form.DateTextBox"/>
  <input type="text" dojoAttachPoint="timeFacet" value=""
             class="xfValue xfTimeTextBox"
             dojoType="dijit.form.TimeTextBox"
             constraints="{ timePattern:'HH:mm:ss'}"/>
</div>
```

This template is referenced within DateTime.js via the templatePath property:

```
templatePath: dojo.moduleUrl("betterform", "ui/templates/DateTime.html")
```

To initially create the DateTime dijit the dojo functions postMixInProperties and postCreate are overwritten and extended. Here only the postMixInProperties function is shown because it calls the this.applyProperties function of the superclass ControlValue. The function **this.applyProperties** must be called by any object instantiation of ControlValues to store a reference to its parent Control and set up some properties.

```
    postMixInProperties:function() {
        this.inherited(arguments); // super call in JavaScript
        this.applyProperties(dijit.byId(this.xfControlId),
this.srcNodeRef);
    },
```

The second function any ControlValue must call is **this.setControlValue()** if a value change takes place the server should be notified about. The function retrieves the value from the ControlValue by calling **this.getControlValue()** and passes it to the server.

This brings us to the two functions any custom ControlValue dijit has to implement:

getControlValue:function();	Returns the current value of the dijit. If the value is localizable and localization is enabled it will be localized by the server but any other transformation of the value has to done within the dijit itself. E.g concatenate the date and time part of the DateTime dijit to a valid dateTime value.

_handleSetControlValue:function(value);	Update the dijit with the value received from the server. In case of the DateTime dijit the value is splited into its date and time part and which are used to update the associated date and time components of the control.

Describing the inner logic of the DateTime dijit would go beyond the scope of this document. Readers who want to go deeper into this should take a look at the DateTime dijit source code, play around with it and compare it to the betterform.ui.input.TextField dijit to get a feeling how ControlValue dijits work. For sure questions on the mailinglist are welcome too.

After finishing the DateTime.js implementation its time to wire the it to an XForms input control bound to a „dateTime" data type. This takes places within the UIElementFactory:

```
var controlType = dojo.attr(sourceNode, "controlType");
var dataType = dojo.attr(sourceNode, "dataType");
...
switch (controlType) {
  ...
  case „input":
    ...
    switch(dataType) {
      case "datetime":
        // use not localized value
        var xfValue = dojo.attr(sourceNode, "schemaValue");
        // load DateTime.js via Dojo module loader
        dojo.require("betterform.ui.input.DateTime");
        // create new DateTime object bound to sourceNode
        newWidget = new betterform.ui.input.DateTime({
                                        name:controlId + "-value",
                                        xfControlId:controlId
                                        value:xfValue,
                                        constraints:{
                                             datePattern:'dd.MM.yyyy',
                                             timePattern:'HH:mm:ss'
                                         },
                                  }, sourceNode);
        break;
      }
      ...
    }
   ...
```

The sourceNode is the DOM representation of the XForms Input bound to DateTime Markup listed two pages above.

Now everything is set up to create and render the DateTime dijit for any XForms Input that is bound to a dateTime typed value.

7 Configuration

All configuration parameters are found in a file named 'default.xml' which is located in package de.betterform.xml.xforms.config in the classpath. If you don't like this policy you may put the configfile in a different location and reference it from web.xml as follows:

```
<context-param>
      <param-name>betterform-config.xml</param-name>
      <param-value>WEB-INF/betterform-config.xml</param-value>
      <description>location of config-file</description>
</context-param>
```

The given location has to be relative to the root of the webcontext.

7.0.1 betterFORM config

The configuration file consists of several sections which are grouped by topic. These are:

- `<properties>`
  this section contains (you guess it) <property> elements which are used to store simple key/value pairs involved in betterFORM configuration. For documentation of these properties please see the comment in default.xml.
- `<error-messages>`
  used to configure the error-messages of the processor itself. (NOT the XForms messages).
- `<stylesheets>`
  while its always possible to set the stylesheet at runtime, this section defines the standard stylesheet to be used. For HTML the entry
  `<stylesheet name="html-default" value="html4.xsl"/>`
  defines 'html4.xsl' as the standard stylesheet for HTML clients.
- `<connectors>`
  in this section the various `Connector` implementations that betterFORM uses for URI resolution can be configured. Every `Connector` defines a scheme attribute which defines the URI scheme which identifies a `Connector` and the fully qualified classname of the `Connector` implementation. The `ConnectorFactory` class will use these entries at runtime to instanciate the configured implementation.
- `<extension-functions>`
  this markup is proposed to configure XPath extension functions that are not part of the XForms standard. This feature has not yet been enabled fully.

- `<actions>`
  similar to `Connector` this sections defines the handler classes for a specific action. This allows to extend the processor and plug-in custom actions.
  Note: dynamic instanciation of actions has been disabled during a refactoring. This feature will be re-integrated.