User Guide

Last update: 10.03.2010 16:58:42
Author: Joern Turner <joern.turner@betterform.de>
Lars Windauer <lars.windauer@betterform.de>
Version: 1.0

Content

1 Introduction

Welcome to the betterFORM User Guide. This document tries to give an overview of the betterFORM toolkit and helps you get started as an XForms author.

betterFORM is based on the W3C XForms 1.1 standard which is aligned with all of the other standards that make up the web (most prominently (X)HTML). XForms defines itself as „the next generation of forms technology for the world wide web“ [1] and is supported and developed by major companies.

betterFORM is a full implementation of XForms 1.1 plus additional functionalities to cover the last mile to a full web application toolkit. While XForms has a powerful abstract Model-View-Controller architecture to build complex logic in forms it is by intend ignorant of layout. betterFORM incorporates and extends the Dojo [2] toolkit to offer application-like layout capabilities (layout containers) and a rich set of controls (Dijits). As Dojo also supports declarative programming it ideally integrates with the XForms approach.

In the technological sense of the term „Web 2.0“ is synonymous for highly interactive, browser-based applications that are supported by AJAX. AJAX resolves to „Asynchronous JavaScript and XML“. A complicated name for a simple thing – with AJAX a browser can make HTTP requests in the background and handle the response to update the page. The language that makes this happen is JavaScript and the data sent are sometimes XML[3]. This rather simple technology was suddenly (re)discovered and leads to a boom of applications especially in social networking applications. Instead of navigating from page to page the „single page application“ was born that dynamically exchanges the contents of a single document until its purpose is done.

Typically Web 2.0 apps involve a lot of hand-crafted JavaScript code, require good knowledge of JavaScript, AJAX, DOM and others technologies and force you to test a lot to assert a good quality. Certain features like validations, calculations are coded over and over again to meet the specific needs of the application. Finally you have glue all this together with events and handle the states of your user interface with custom code. With XForms and betterFORM you build on a higher ground. Forget about the details of the browser technologies and how to build good-looking, dynamic webapps. Just use the declarative power of a markup language to express your logic, handle complex validations, speak to all kind of external datasources etc. betterFORM shields all the tricky details of state keeping, life-cycle management and

1 http://www.w3.org/MarkUp/Forms

2 Dojo – Unbeatable JavaScript Tools (http://dojotoolkit.org/)

3 JSON is another popular data format in this context which is likely supported by future versions of XForms.

data consistency from you and lets you concentrate on the purpose of your application.

This User Guide[4] mainly targets at people trying to quickly build user interfaces or applications without worrying much about the technical details behind. Those interested in the details under the hood should read the betterFORM Developer Guide.

4 This guide does NOT cover the various other options of using betterFORM e.g on client-side, with a fat client or with mobile browsers.

1.1 Who should read this guide?

You should read this guide if you

- are looking for a conformant XForms 1.1 implementation
- are planning to develop a web application which involves the use of XML data, web services or business processes
- already have started to develop (XML based) web applications
- are looking for a rough overview of the betterFORM toolkit and its high-level offerings
- want to write, deploy and run XHTML/XForms documents
- are seeking to understand more about page layout and styling for XForms
- want to understand the pieces betterFORM is made of and what you'll find on your disk

1.2 What is not covered?

This guide is no XForms tutorial and assumes at least a basic understanding of W3C XForms. There are many sources on the Web about XForms and perhaps the best starting point for learning XForms is the Wiki book. The link can be found at the end of this document. This guide won't explain any of the powerful XForms features neither does it cover XForms authoring as a whole.

The Dojo Toolkit is not covered at all. Though betterFORM heavily relies on Dojo features you won't need any knowledge about it to write even advanced XForms. if you like to learn more about Dojo please refer to the link section.

2 Why use betterFORM?

There are probably hundreds of ways today to build a webapp. Why should you consider betterFORM for the job?

- ***Based upon open standards:*** betterFORM is addicted to open standards like XML, XForms, XPath, XQuery, CSS, JavaScript, Java and XSLT. betterFORM itself provides a complete and open implementation of the XForms 1.0 + 1.1 standards.
- ***Quality:*** The betterFORM project was designed with modularity and extensibility in mind. The code has undergone many years of development, testing and improvement and has been successfully used in dozens of projects in the fields of banking, eCommerce, eHealth, aeronautical information Management, science, business process management (bpm), payment services and enterprise content management (ecm).

  About 580 unit tests assure the proper operation of the core processor and make sure that the code stays alive. In addition betterFORM passes 98%[5] of the official XForms 1.1 conformance tests (results are frequently updated on our homepage). Dojo, which is used extensively by betterFORM, is a high-quality widget and layout system for really good-looking and accessible front-ends.
- ***Minimal-intrusive architecture:*** betterFORM comes as a Servlet filter filtering requests to certain URLs on your server. Pages that contain XForms markup will be handled by betterFORM automatically. This will be completely transparent to your application just by editing your web.xml and without any coding involved. As an alternative deployment betterFORM can be installed as a separate WAR file alongside your webapp. XForms pages will then be forwarded to betterFORM for processing. Both applications will be completely separated from each other allowing selective upgrading of each of the webapps.
- ***Zero-install:*** XForms is a powerful new standard but not natively supported by current browsers on the market. Server-side XForms can deliver cross-browser compatible user interfaces without the need of any kind of installation or plugin.
- ***Declarative:*** XForms is a declarative language which makes it powerful for Rapid Application Development. Main advantage is that XForms is designed for a purpose (what it should do) and not saying HOW it's done. Besides the controls you also get a declarative event, action and submission model so all aspects of an application can be covered from I/O to presentation without writing a single line of imperative code. Not stopping there you can still extend your pages with custom script and use the rich features of the Dojo toolkit.

  Dojo itself can be used in a declarative manner so mixing and matching

5 Tested on reference platform Mac OSX, Firefox 3.0. Chrome, IE 7 and Safari results are published on our homepage.

XForms and Dojo is a natural fit. Especially in layout Dojo can help with TabContainers, BorderLayout, ContentPanes and others without interfering with XForms in any way.

- ***Data typing, validation and calculation:*** XForms provides strong datatyping for your app and takes care that everything is valid and consistent in your data. Extension functions allow to bind even the most complex validators or calculators and give you the choice between different languages to implement them.

  The dependency engine is an outstanding feature of XForms to track the most complex relationships between data nodes and to automatically keep everything in sync. At the same time the algorithm used is optimized to calculate only those nodes that really needed at a given time. Don't worry about the execution sequence to get the right results. Just markup your constraints and calculations and the dependency engine will do the rest.
- ***Ideally suited for REST[6] and XRX[7]:*** Integration with RESTful services or architectures is a matter of minutes. In combination with XQuery even complex data-management tasks can be handled. When your data is already XML you can hardly get a faster development process.
- ***Open Souce:*** betterFORM is licensed under the BSD and Apache 2 licenses which are very loose and business-friendly. An open standard and a free license avoid vendor lock-in and protect your investments.

2.1 When to use it?

- You heavily use XML in your backend and need to quickly generate user interfaces for data mainteinance
- You're not a JavaScript or AJAX expert but like Web 2.0-style applications
- You never want to worry about data validation any more in your backend
- You're building loosely coupled architectures with REST or web services
- You have CMS, ECM, BPM or Workflow involved with your app
- You're looking for open (W3C) standards in the area of user interfaces
- You have large amounts of data to be maintained by humans
- Accessibility, internationalization and localization are important for you
- plus all the reasons listed under „Why use betterFORM?"

6 Representational State Transfer (REST): http://enwikipedia.org/wiki/Representational_State_Transfer

7 Xforms – Rest – Xquery (XRX): http://en.wikipedia.org/wiki/XRX_(web_application_architecture)

2.2 When to not use it?

- You quickly need a contact form for your personal homepage to let visitors leave a message to you. For trivial use cases XForms may be a bit oversized.
- If you're experiencing anaphylactic reactions when getting in contact with XML you should stay away

3 Getting started

Before you get started you should have read the introduction. This chapter will guide you through all steps needed to execute betterFORM and run your XForms pages.

3.1 Preparations/Requirements

To run betterFORM you'll need the following:

- JDK/JRE 1.5 or higher
- A Servlet 2.3 compatible web container like Tomcat or Jetty

For best viewing experience we recommed one of the following browsers (minimal browser version in brackets):

- Chrome (4.x)
- Internet Explorer (7.x)
- Firefox (3.x)
- Safari (4.x)

Chrome and Safari are preferable because they render much faster than Firefox or Internet Explorer. Further both provide good debugging tools. Nevertheless we recommend Firefox with the great Firebug[8] plugin for debugging, CSS editing and DOM introspection. betterFORM logs a lot of interesting events to the Firebug console[9]. If you like to see what goes on behind the scences give Firebug a try.

3.2 Installation

betterFORM comes with a izPack based installation wizard that guides you through the steps of installation and allows to choose between some common scenarios:

- you can install the plain webapp in an existing web container like Tomcat, Jetty, JBoss, Glassfish, Websphere and others
- you can install betterFORM along with Jetty as the web container
- you can additionally install the eXist XML database and use it in conjunction with betterFORM to build XRX applications

8 Firebug – Firefox Debugging Tool - http://getfirebug.com/

9 Console logging will also work for Chrome and Safari

- you can optionally install the demo, reference and test forms of betterFORM as a source for learning XForms
- you can optionally install the complete XForms 1.1 Test Suite and run them in betterFORM.

We'd like to encourage people to use the installation wizard. To reduce download size betterFORM uses a webinstaller and you only need to download the packages you have actually chosen.

To use the web installer (you need a running java installation):

1. Download the latest betterform-installer.jar
2. Doubleclick the jar-file once you have it on disk
3. Follow the instructions on screen to install the packs you like

3.2.1 Installation of WAR file

The web installer will install betterFORM as an exploded webapp either in your existing web container or in Jetty. If you don't like this or prefer to work with a plain WAR file follow these steps:

1. Download the latest betterFORM WAR file from Sourceforge http://sourceforge.net/projects/betterform/files/
2. Rename the downloaded file to betterform or any other name you like. The .war file extension must be kept.
3. Move the renamed WAR file to the 'webapp' directory of your servlet-container
4. Start the container – that's it

After startup the servlet-container should unpack the betterFORM WAR file and you may access the betterFORM forms by browsing to http://localhost:8080/betterform (substitute 'betterform' with the name you've choosen for your WAR file if applicable). If you managed to start the servlet container and browse for the above URL you should see the homepage of betterFORM. To run your first XForm click the 'browse-forms'-link. This opens a window showing a directory structure containing several XForms documents (file extension is .xhtml).

3.3 Test your installation

To test your installation go to: http://localhost:8080/betterform/forms/status.xhtml .

If you see a rendered page with some details about the configuration everything is up and running.

3.4 What you got installed

Here's a short description of the contents of the WAR file after your web container has been started. Normally the WAR file is expanded (unpacked) automatically by the web container. Some containers allow to run unexpanded WAR files. In this case you have to use an unzip utility to see the structure described below:

Directoy/File	***Description***
forms/	This is the root of the forms coming bundled with betterFORM. There is a subdirectory structure which groups the documents in categories. You may simply create your own directories below the forms directory to keep them seperately [10].
index.html	betterFORM homepage
resources/	In these directories betterFORM stores all the things it needs for processing. You'll normally do not need to touch these files unless you're a advanced user.
resources/images/	As the name says – some images we're using as icons.
resources/jsp/	Pages to support the demo webapp. The forms browsing page, an error page and a debug page that can be used for testing.
resources/scripts/	The betterFORM JavaScript sources including a complete Dojo distribution.
resources/styles/	Here the default XForms and betterFORM-specific CSS style sheets are found. This may be a convenient place to put your own custom CSS style sheets.
resources/xsd	A directory for putting XML Schema files that are used in the forms.
resources/xslt/	betterFORM uses XSLT to generate the Client UI. In this directory all involved style sheets are found.
WEB-INF/	

10 This is at least good enough for a starter as long as you've not integrated betterFORM into your webapp anyway or load from a different location.

Directoy/File	***Description***
WEB-INF/betterform-config.xml	The configuration file for the XForms engine.
WEB-INF/betterform.profile.js	A Dojo profile for building a compiled[11] version of the betterFORM JavaScript.
WEB-INF/dwr.xml	Configuration file for DWR 2. betterFORM uses Direct Web Remoting (DWR) for the AJAX layer. The end user does not need to touch this file.
WEB-INF/dwr20.dtd	DTD for DWR
WEB-INF/log4j.dtd	DTD for log4j
WEB-INF/log4j.xml	Configuration file for log4j (logging).
WEB-INF/web.xml	Contains the configuration of the ServletFilter for betterFORM

3.5 Deploy pages

To deploy and run your own XForms pages please follow these steps:

1. locate the expanded betterFORM WAR file structure on your disk e.g. '`/tomcat/webapps/betterform`'
2. step into directory 'forms'
3. create a subdirectory for your pages
4. copy your xforms pages to this directory
5. reload /betterform/resources/jsp/forms.jsp in your browser to see the added files

If you need additional resources (images, styles etc.) you may put them anywhere within the betterform directory.

3.6 Use the source Luke!

After you've installed betterFORM you may want to have a look at the sample forms. We constantly try to maintain our demo pages with each new release and show some working XForms functionality.

There is a folder 'reference' which comes with the distribution and contains some forms that show the set of available controls and containers along with markup and explanations. This directory will be constantly updated and new forms will be added

11 JavaScript compilation dramatically improves performance on the client and should always be used in production environments.

as the functionality of betterFORM extends. These reference forms are considered a live documentation which complements this guide.

Another good source for studying are the test forms which will show you most of the features a specific control or container has and how they behave.

By studying them in detail you can figure out many XForms features as well as discover some styling tricks for XForms development.

The XForms Working Group[12] (WG) has published an extensive conformance test suite which covers all of the features of the specification. By looking at the tests you can learn about the syntax and the functionalities of the standard. The test suite is part of the betterFORM distribution and each test can be run directly from our form browser.

For the curious it's recommended to run betterFORM pages within Firefox and latest Firebug plugin installed. This will show you a lot what's going on behind the scenes.

12 http://www.w3.org/MarkUp/Forms

4 Building smart interfaces with betterFORM

XForms offers 'as simple as can be but no simpler' solutions to common problems but can do much more than a plain GUI toolkit. It covers everything from simple forms (e.g. your companies smart contact form) to full-fledged applications with complex logic requirements.

As it's hard to nail down in a few sentences what XForms can do it is recommended to look around in the net and let yourself being inspired by the tremendous range of scenarios it is used for.

As already mentioned this document is not a XForms tutorial. Instead this chapter will show how to use betterFORM as a page author. What naturally leads to the next section.

4.1 What's page authoring?

First of all page authoring in essence means to use the declarative approach of XForms to build xhtml pages containing logic, calculations, validations, event-handlers, actions and more. You can even write complete applications with this approach.

Everything you'll write are tags, attributes and CSS rules. You'll not need to write any JavaScript or Java to make things happen. You don't even need to understand how all the AJAX, XForms magic is done behind the scenes. But you should have at least a basic understanding of XHTML, CSS and XForms. As you go you'll probably like to learn about Dojo e.g. for page layout or animations.

> Note:
> Though absolutely no JavaScript needs to be written it's still possible to do so. The experienced user can tweak even the last bits of the UI and interact with the XForms processor via an API. These topics are covered in the developers guide.

Page authoring means to do at least some of the following things:

- creating XHTML/XForms documents
- use XForms models to work with XML data
- use XForms 'appearance' attribute to select between various rendering options
- use Dojo layout containers for page layout

- adapt the rich set of pre-existing CSS-classes to your needs
- add custom CSS matchers to fine-tune styling
- use XForms actions and events to add logic and behavior
- use XForms submissions to load or send data
- add validations and data types to certain data nodes
- add calculations
- provide help, hint and alert texts

You'll not have to worry about:

- when to display validation errors
- data validity and consistency. The processor will keep track on all dependencies and revalidates everything when needed
- keeping client and server in sync
- wrong or invalid data received by your backend
- dynamic behavior
- cross-browser compatibility

4.2 XForms basics

While not being a XForms tutorial we nevertheless should clarify some basic XForms concepts and terms to avoid confusion and help understanding the further explanations.

4.2.1 Concrete model and abstract UI

One often underemphasized aspect of XForms is that it's 'concrete about the model and abstract for the UI':

You may define a fine-grained model of your data along with validations, calculations and types (usually XML Schema SimpleTypes). You may even use your own XSD-types or custom XPath functions and control every detail of data consistency as well as loading and saving of data via submissions.

For the UI it's different: the XForms UI is abstract. XForms emphazises device-independence by using tag names for controls that avoid any association with visual interfaces such as 'checkbox' or 'border'. XForms are designed for a specific purpose and not for a specific client or device. This allows the same form to be used with a browser, a mobile phone or even a voice application by transcoding the logical XForms UI elements to their appropriate equivalents in the target language (device

language). When authoring XForms you should keep this in mind if you're interested in supporting multiple clients.

Nevertheless most users will deal with XHTML as host documents in the overwhelming majority of cases. So what's the advantage of being abstract?

First it reminds you that the rendering and concrete appearance of the control may vary from client to client but will always fulfill the same logical purpose. On the other hand it allows to automatically choose the most appropriate representation for a given case and shows the same instance nodes with different controls. Through useragent detection an implementation can choose the most appropriate profile for serving that specific client and show optimized controls and structures for it.

4.2.2 Host language

The language of specifications sometimes becomes a bit arcane and hard to understand for the average developer or page author. The good news is that you don't need to read the full specification to start using XForms. There are many good tutorials out there that are much easier to consume. Nevertheless it's good to understand some basics upfront.

> XForms is NOT a standalone markup language!

Instead XForms is an embedded language meant to be used inside other XML markup languages. Of course the most prominent one is XHTML but there's nothing preventing you from embedding it into VoiceML, SVG or any other XML markup language.

The language you're are using to embed your XForms markup is called **'host language'** and a concrete document (e.g. a XHTML page) is called a **'host document'**. XForms itself makes no assumptions about the host language besides being a well-formed XML language but relies on it e.g. for laying out pages.

In the context of this document we'll concentrate on embedding XForms in XHTML.

4.2.3 A XForms skeleton

To embed XForms into XHTML you have to watch a few things:

- put the XForms namespace on the root element of your document. If you're unfamiliar with namespaces – don't worry. Once you have accepted they need to be there you can quickly forget about them again. If not sure what to put there just copy 'n paste from one of the sample forms. The snippet below defines namespaces for XHTML, XForms and

further the namespace of XML events is needed as soon as you're defining event-handlers in your document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<html xmlns="http://www.w3.org/1999/xhtml"

      xmlns:xforms="http://www.w3.org/2002/xforms"

      xmlns:ev="http://www.w3.org/2001/xml-events"

...
```

- add a <xforms:model> tag to the head or body section of your XHTML document (or 'xf' if you have choosen that for your prefix). Placing the <xforms:model> tag within the head is a mere convention that has been evolved over time. It is totally up to you where you put the model element[13].

  The model itself might have one or more of the following children:
  instance – element wrapping or referencing the XML data of the form
  bind – to add constraints, states and datatypes to nodes
  submission – defines loading, processing and saving operations (interaction with some data source)
  action – listen on events during model-initialization

- add some XForms UI to the body of your document In this example a <xforms:group>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xforms="http://www.w3.org/2002/xforms">
    <head>
        <title>hello</title>
        <xforms:model id="hello-model">
      (...)
        </xforms:model>
    </head>
    <body>
     <xforms:group>
      (...)
      </xforms:group>
    </body>
</html>
```

- bind some of your controls to the model data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xforms="http://www.w3.org/2002/xforms">
```

13 More and more implementations start to put the model into the body of the document. This is especially true for the client-side JavaScript-based implementations. This is perfectly legal and just a matter of taste.

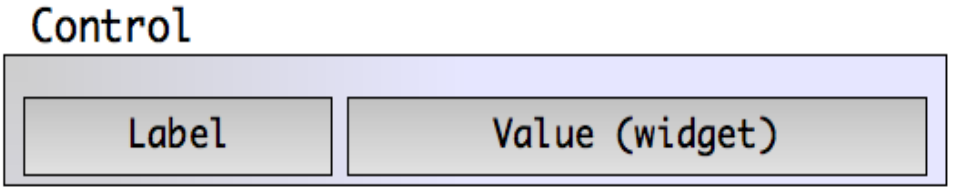
```
    <head>
        <title>hello</title>
        <xforms:model id="hello-model">
           <xf:instance>
              <foo xmlns="">bar</foo>
           </xf:instance>
        </xforms:model>
    </head>
    <body>
     <xforms:group>
        <xforms:input ref="/foo">
          <xforms:label>Foo</xforms:label>
        </xforms:input>
     </xforms:group>
    </body>
</html>
```

betterFORM works with XHTML 1 documents. XHTML 2 or plain HTML documents won't work as host documents. If you'd like to support different namespaces you'll have to change the XSLT stylesheets of betterFORM. Information about changing the XSLT renderers or implementing your own will be found in the betterFORM Developer Guide.

You may now add `bind` elements to the model to constrain data in your instance(s) and UI controls to the `group` in the body of the document.

4.2.4 Anatomy of a XForms control

In XForms the structure of a control is constituted by a label and an 'interaction' widget:



Sample markup:

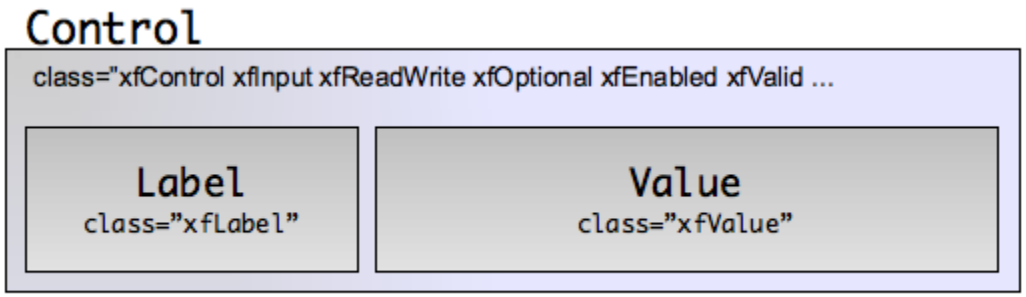
```
<xforms:input ref="foo">
     <xforms:label>bar</xforms:label>
</xforms:input>
```

The label element above is mandatory for each XForms control except output [14] though most processors do not complain if you leave it out. The reasoning behind is that the Working Group wanted to make sure that the accessibility of a form is always given.

The 'Value' part of the control is the actual widget – that area that allows the user to give some input or interact with the system.

It is important to keep this structure in mind when you like to customize the rendering of controls via the automatically assigned CSS classes (see below).

At runtime you'll see that the control wears a lot of information represented as CSS classes (Tip: you can use the Firebug extension for Firefox to take a look at the CSS classes). For the example above you'll get the following for the input:



Each XForms control will have a `xfControl` class assigned. The type of control is given through the second class (here `xfInput`). The following classes are reflecting the current state of the Model Item Properties (MIP) [15] for the instance node bound to the control. In this case the bound node of the input is readwrite, optional, enabled (relevant) and valid. There may be further classes for the XSD datatype, the `incremental` attribute etc.

For details about which classes will be assigned to each kind of control or container please refer to our CSS reference which can be found under http://www.betterform.de/betterform/CSSReference.pdf

4.3 Some guidelines to writing forms

A few simple guidelines when starting to write your first XForms:

- Avoid deep nesting of structures in the UI as it makes layout and styling via CSS harder and more error-prone.
- Start with simple forms
  Start experimenting with simple forms and make them run before advancing

14 This element might be used inline with other content of the page and therefore here the label is optional.

15 see glossary entry about MIPs

to the more tricky parts like complex bindings, constraints and events. It's always simpler to move on if you have a running base.

- Understand the basics
  Experiment with the main building blocks of XForms and try to get an understanding for the processing model of XForms. Use the developer tools of your browser to see which events will be fired at certain times of the lifecycle or in response to your interactions[16].
- Avoid common pitfalls
  When something goes wrong with your form check the most common points for failures are:
    - a needed namespace is not declared
    - an instance or its root node is lacking the needed empty namespace ('xmlns=“”')
    - bindings are using XPathes which do not exist in the instance
    - an instance that cannot be found or loaded
    - incorrect idrefs (pointing to an unknown element)

4.4 Page layout

XForms beginners are often wondering why XForms doesn't provide any mechanism for page layout. The reason is the already mentioned abstract nature of XForms – to be truly device-independent no assumptions about the device's capabilities can be made. Not even if there is something like 'layout'[17]. Thus XForms leaves the responsibility for a nice and ergonomic user experience to the host language.

The vendors of XForms implementations have found different ways to deal with this but over the years common patterns have evolved. Most implementers use CSS to style their XHTML/XForms documents and have setup their own CSS system of classes that allow to influence the rendering of forms. The specification defines some styles and pseudo classes for XForms but these are defined in CSS 3 which is not widely supported in browsers yet. Thus the implementers had to invent their own custom CSS classes to bridge that gap. This limits the portability of XHTML/XForms documents between implementations with regard to the look and feel. Nevertheless compliant implementations should correctly interpret the purpose of a form even if it looks differently.

The following sections outline some common patterns for layout and styling The patterns do not have to be used mutual exclusive. Instead it will be common to use

16 For this to work debug mode must be configured properly. Please refer to the appropriate Section in this document.

17 Like in the often-stressed example of a XForms-based voice application. Though rarely seens on this planet it's still a nice vision for the future.

mixtures of them. They are described as separate patterns here to point out the differences and consequences of each approach.

4.4.1 Direct layout

This pattern applies whenever a Form is layouted by markup contained in the host document e.g. through placing XForms controls into HTML table tags. This way the layout is hard-coded in the host language. This allows to use all features of the host-language but limits the device-independence of the form because not every client will understand this specific language or is limited in its interpretation.

For highly customized and complex pages this approach leads to a stable and reliable layout that works in many different browsers. It might also be useful when the author has limited CSS knowledge or the page is embedded into other pages that use complex CSS themselves. In this case CSS matchers may interfere and break some of your styling rules.

The drawback of this pattern is that the page will become much harder to modify when layout requirements change or new controls have to be added.

4.4.2 CSS layout

CSS layout is signified by a XHTML document container that uses minimal markup to embed the XForms. Most notably there will be a header and body section and probably a couple of HTML div elements for establishing the rough logical structure of the page. For maximum flexibility and changeability it should be avoided to heavily mix XHTML and XForms markup. Instead sections of XHTML and XForms should alternate and deep nesting of markup should be avoided.

CSS layout is especially well suited to be mixed with automatic layout via the XForms `appearance` attribute which is described in greater detail in section 4.6.

This pattern is recommended for production environments where device-independence and production efficiency is required and many forms have to be maintained. It allows single-source authoring for multiple clients and separates content from layout.

Though leading to cleaner and simpler markup this approach may require a bit more of planning than direct layout but the payoff is a much increased flexibility when it comes to changes of the overall look and feel.

4.4.3 Dojo layout containers

Dojo comes with a set of layout containers that can significantly ease the task of building dynamic layouts that work across different screen resolutions and browsers. You just have to write some simple HTML divs and assign the wanted layout declaratively. The implementation will take care of the rest and make sure that the layout adapts as wanted when the browser window is resized.

Using these readymade containers frees you from developing your own hand-rolled CSS rules and tests those against all the target browsers you'd like to support.

As with CSS layout careful planning avoids running into trouble so you should start with a simple layout and work forward into the details. For documentation on the several types of available layout containers please refer to the Dojo documentation found at http://docs.dojocampus.org/dijit/index .

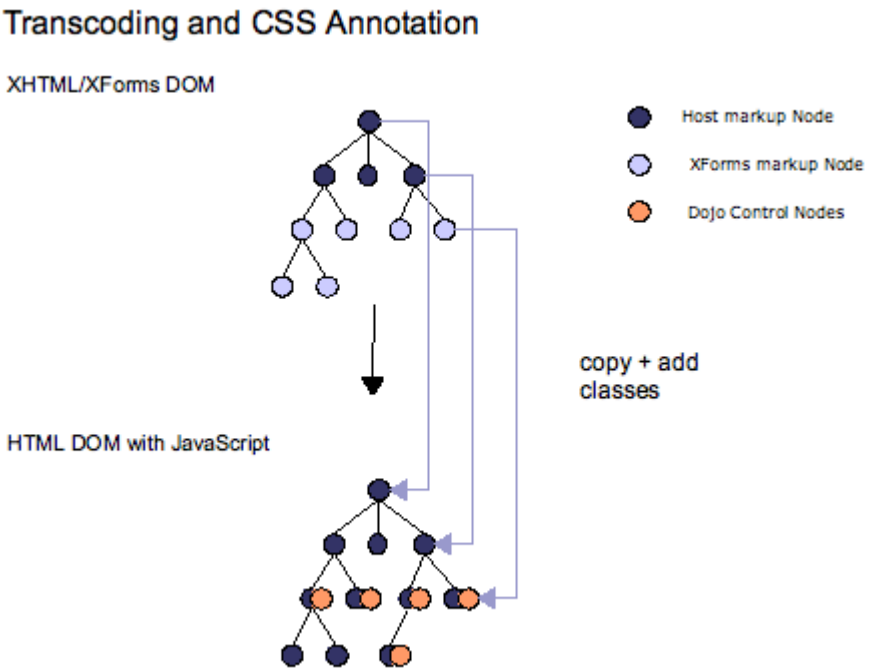
4.5 Styling controls

The following paragraphs explain how to use CSS stylesheets in order to add custom styles or modify the look of controls and their states.

A quick glimpse behind the scenes helps to understand what betterFORM does to support layout and styling for all the authoring patterns mentioned above.

4.5.1 XForms Transcoding

The graphic below shows how betterFORM transforms the input host document for rendering in a browser:



To turn the abstract XForms UI markup into something the browser can interpret betterFORM transcodes the page to HTML, CSS + JavaScript. During this process every XForms control or XForms container will be turned into some corresponding HTML (usually simple or elements) along with some attributes and CSS classes. XForms controls turn into Dojo widgets ('Dijits' in Dojo speak) which represent the original XForms control.

As XForms controls wear state properties (MIPs) like readonly, relevant, enabled or valid which are not present in standard HTML these semantics have to be injected somehow into the resulting page.

While transforming betterFORM assigns appropriate CSS classes for all relevant XForms state properties to the respective control or container. Page authors can use these to customize the rendering of certain states or layout structures. This process is referred to as 'CSS annotation'.

You can find a complete betterFORM CSS reference in a seperate pdf at:
http://www.betterform.de/betterform/CSSReference.pdf

4.5.2 Styling individual controls

The above approach is fine if you want to globally change the rendering for controls and XForms containers[18]. It makes reuse of layout and styling information for multiple forms easy and should always be the preferred way.

18 XForms containers are `group`, `repeat` and `switch`

Nevertheless situations may occur where you need direct access to a certain control or group element for styling purposes.

You have three options to get the same result:

- Add a custom CSS class to your XForms markup.
  This allows to reuse the same class for a set of similar controls that are probably spread within your form or forms. E.g. if you have a small integer which uses a xforms:input for editing you probably don't want it to be 200 pixels wide. Assigning a 'mySmallInteger' class will solve your problem for all occurrences.
- Use the style Attribute to directly assign some styles to a single control.
- Assign an id to the XForms control and use it with a CSS id matcher.

Here's an example to illustrate all three options:

```
<xforms:input ref="foo"
              id="myId"
              style="color:red;"
              class="myCustomClass">
     <xforms:label>bar</xforms:label>
     <xforms:alert>This value is not valid for this field</xforms:alert>
     (...)
</xforms:input>
```

During transform into HTML betterFORM will apply the original id of your input to the resulting structure in the output. Further all XForms common childs [19] will get an id dependent on this. The custom CSS class will be passed to the output and can be used in your CSS.

Here's the **resulting output** for the CSS class defined above

```
<div id="myId" class="myCustomClass xfControl xfInput(..)" style="(..)">
     <label id="myId-label" for="myId-value">bar</label>
     <input id="myId-value" ...>
     <span id="myId-alert"> This value is not valid for this
field</span>
</div>
```

The wrapper div that represents the XForms control gets the original id of the XForms control. The respective child elements get the orginal id plus a suffix that matches the name of the respective element.

This provides several ways to write your own CSS matchers [20]:

19 aert, hint, help and label Elements

20 Only some possible matchers are shown here. Other variants are possible to match the same element(s) in the output document.

```
// *** using custom CSS class
.myCustomClass{(...)}

// matching common children with custom CSS class
.myCustomClass .xfAlert{(...)}

// *** using with id matchers

#myId{
       font-weight:bold;
}
#myId .xfLabel{
       //put your label styles here
}
#myId .xfAlert{
       //put your control styles here
}
// or

#myId-alert{background:red;}


// matching the widget of a control
#myId .xfValue{background:#dddddd;}
```

4.6 The mighty `appearance` Attribute

The XForms standard defines the `appearance` attribute to allow XForms authors to pass a hint to the XForms processor to select one of potential several different renderings of the given control. So for instance a `xf:select1` might be rendered as a group of radio buttons or as a combo box depending on the choosen appearance.

There are three predefined appearance values (`minimal | compact | full`) mentioned in the Xforms Spec plus the option to add your own ones with qualified name e.g.

```
<xf:switch appearance="dijit:tabContainer" ...
```

'dijit:tabContainer' is a custom extension which is only available in betterFORM and will be rendered as a Dojo TabContainer.

A XForms processor is free to choose the most appropriate rendering though the Spec gives some hints how to interpret the values of the `appearance` attribute for certain controls. The reference forms found in the betterFORM distribution show which values for `appearance` are available for a given control and how the final rendered results looks like.

In betterFORM the appearance is not only supported for controls but is also interpreted for `group`, `repeat` and `switch`. This is a very quick way to choose between some basic layout options. Again these layout alternatives can be studied best in our reference forms that are found in 'forms/reference' in the distribution.

5 Handling data

In the old days of HTML forms there were only limited possibilities to load and submit data. They could be hard-coded into the page as values of controls and sent as a http request to a server with GET and POST.

XForms on the other hand has a rich set of features to load data from arbitrary sources, use different protocols, methods and encodings, provide response handling etc. etc.

This section will touch the basics of loading and saving of data and introduce some betterFORM extensions in this area.

5.1 Inline data

The simplest way to work with some XML data is to write them inline within your form markup:

```
<xf:model>
     <xf:instance id="myInstance">
          <data xmlns="">
               <address>
                    <city>Berlin</city>
                    <street>Hohenzollerndamm</street>
                    ....
```

When your form comes up and has controls binding to the above data they will display 'Berlin' and 'Hohenzollerndamm'. Instances can be used to preset some data before showing the rendered form.

Be aware of one important concept of XForms: once the instance is loaded a separate document will be created for it which clones the inline data. All data manipulations are happening on this external document. You'll never see the inline instance change at runtime. This is true for all instances independent of how they were loaded.

5.2 The baseURI

One important concept for the resolution of data is the baseURI. Every XHTML/XForms document will have a baseURI. This is the place from where the document was loaded into the XForms processor. If the document happens to be on a server in the internet (which will be most often the case), the baseURI will be an

absolute http URL pointing to the document itself. If your processor is capable of loading XForms from the file system it will be a file:/ URL.

In many web-apps documents are created dynamically by a template language. In this case they don't really exist under a certain URL (instead they will have the URL of the templating engine) and it may be necessary to assign an explicit baseURI. This can be done with the xml:base attribute.

Example:

```
<xhtml:html xml:base="http://anotherHost/foo" ...
```

5.3 Loading data

The simplest way to load some data into a form is the use of the `src` and `resource` attributes of Element `instance`. Here's an example:

```
<xf:instance src="data.xml">
      ...
```

Once the XForms processor initializes the instances the `src` Attribute will be evaluated, the link will be resolved and the resulting response will be parsed as XML. If something fails a `xforms-link-exception` is dispatched.

Loading data this way is rather limited as there's no way to customize the request. It will be a plain GET request to fetch the referenced resource. There is no way to send parameters, headers or use a different request method.

This will be fine as long as you have plain, more or less static resources such as language files, static reference lists or helper instances. If you have more fine-grained requirements you should use the XForms `submission` module which gives you all to interact with complex webservices, do authentication, handle exceptions or other events and give messages to the user. The full feature-set is too rich to be explained here in detail but some simple examples should get you started.

5.3.1 Loading data with submission

```
<xf:instance id="myData">
      <data xmlns="">
            <foo/>
      </data>
</xf:instance>
<xf:submission
      id="mySubmit"
```

```
      method="GET"
      replace="instance"
      resource="http://myHost/myTarget" ...>
  <xf:action ev:event="xforms-submit">
      <xf:message level="ephemeral">
          about to send your data...
  </xf:message>
  <xf:action ev:event="xforms-submit-done">
      <xf:message>success!</xf:message>
  </xf:action>
  <xf:action ev:event="xforms-submit-error">
      <xf:message>oops, an error occurred</xf:message>
  </xf:action>
</xf:submission>
```

The above examples shows the loading of data by using a `submission` with method GET to load the wanted data from 'http://myHost/myTarget'. The response will be parsed as XML and used as instance data. This is expressed by the `replace` Attribute with value 'instance'.

A XForms processor will send specific events at the different stages of `submission`. First a `xforms-submit` will be dispatched before the submission actually starts. Dependent of the success or failure of the request the event `xforms-submit-done` or `xforms-submit-error` will follow. These events can be used to inform the user, deal with exceptional cases or show details about the problem.

When data have to be loaded at startup it's common to attach the `send` action to a `xforms-ready` event to fire the submission „mySubmit" automatically once everything is initialized.

```
<xf:model>
...
  <xf:action ev:event="xforms-ready"
      <xf:send submission="mySubmit"/>
  </xf:action>
...
</xf:model>
```

Hint:
Although this example focusses on the loading aspect all XForms submissions also send data even for method GET. In this case the existing instance data will be send urlencoded on the GET string. If you want to suppress that and send a plain GET use **`serialization="none"`** on your submission element.

5.4 Sending data

Sending of data works the same as in the example above. Besides using GET you can use POST, PUT and DELETE and have a rich set of switches to tailor your request, define the target and the handling of the response. Please refer to Chapter 11 of the XForms 1.1 Final Recommendation for a full overview of the Attributes and child Elements.

5.5 Using parameters

In the early days of XForms all `submission action` targets and `instance src` Attributes were only static strings. But of course this limited their usefulness a lot. In almost any application you need a way to pass params on the request string. This was hard in these days and the implementers had to make inventions.

One natural fit for use in XForms is the adaption of the so-called Attribute Value Templates (AVT) from XSLT. For any attribute the contents between curly brackets is interpreted and evaluated as a XPath expression.

betterFORM supports the AVT approach in very limited areas as on `submission resource` or `action` Attributes, on the `instance src` Attribute and for `load resource` Attribute. An enhancement of the support is planned for future versions.

As part of an AVT a variable expression or XPath expression can be used. Variables start with a '$' followed by the name of the variable. An expression not starting with a '$' is interpreted and evaluated as an XPath.

This table shows the support for Variables and XPath expressions for different Elements:

AVT support	Variable	XPath
Instance src	Yes	No
Submission resource (action)	Yes	Yes
Load resource	Yes	Yes

5.5.1 Using variables

As mentioned variables always start with a '$' sign. There are several predefined variables, some HTTP header-related ones as well as a variable for every param your document was called with. An example will illustrate this.

Assume your document was called with the URL:

http://myhost.com/mydoc.xhtml?filename =foo.xml&bar=baz

betterFORM will parse all params and store them. You may then access the values of these variables:

```
<xf:instance src="{$filename}?query={$bar}">
...
```

Variables can be accessed in two ways:

1. starting with a '$' sign as prefix in AVTs on the above mentioned XForms Elements

```
<xf:submission resource="http://myHost/{$foo} ...
```

2. by using the appContext() XPath function. This will take a variable name as string argument and return the respective value. This function can be used anywhere in XForms where a XPath expression is allowed.

```
<!-- this output will display the webapp context string -->
<html
xmlns:bffn="java:de.betterform.xml.xforms.xpath.BetterFormXPathFunctions
">
...
     <xf:output value="bffn:appContext('contextroot')" />
...
```

To use the appContext() function you have to declare the namespace shown in the example above.

5.5.2 Available Variables

Predefined Variables:

betterform.baseURI	The absolute baseURI of the document.
contextroot	Returns the web-app context name e.g. '/betterform'.
betterform.locale	A lowercase two-letter ISO-639 language code denoting the selected locale.

httpSessionId	Returns the HTTP sessionId for the current user. Please note that the name of this param is dependent on the web container implementation. 'httpSessionId' is the key that is used by Tomcat.
queryString	The query string passed on current URL.
requestURI	Absolute HTTP URL for the current web-app e.g. http://myHost/betterform
requestURL	The absolute HTTP URL of the form. This will have the same value as $baseURI unless the baseURI has not been modified by the xml:base Attribute.
webapp.realpath	The absolute file URL of the webapp context.

Runtime variables:

Any URL param	Besides the whole querystring as available from the 'queryString' variable each param passed to the form will be present as standalone variable. E.g. if a ?foo=bar param is passed you can access the variable $foo or bffn:appContext('foo') to get the value 'bar'.

HTTP-headers	Any HTTP header will be stored as a variable. To e.g. access the user-agent use the key 'http-headers/user-agent'. It depends on the environment which headers will be sent by a given browser. Most of the time following keys should be present: - http-headers/host - http-headers/user-agent - http-headers/accept - http-headers/accept-encoding - http-headers/accept-language - http-headers/accept-encoding - http-headers/accept-charset - http-headers/keep-alive - http-headers/cookie Other headers may be present but you shouldn't rely on them unless you're sure they are sent.

5.5.3 Using XPathes in AVTs

Especially if you want to route your requests dynamically dependent on some data in your instance(s) it is necessary to use XPathes in your submissions or load actions.

Example:

```
<xf:instance>
      <data xmlns="">
            <newURL>http://www.google.com</newURL>
      </data>
</xf:instance>

<xf:submission resource="{instance()/newURL}" ...
```

In this example the AVT inside of the `submission resource` Attribute will be evaluated before the `submission` actually takes place thereby leading to a changed location depending on the value currently set in the instance [21].

21 The same functionality is available through the resouce child Element in XForms 1.1 so this approach is not needed any more. Nevertheless we have not deprecated the AVT support and it is a matter of taste which one to use.

5.6 Noteworthy helpers

This section lists some helpful Connectors you can use in your forms. Please be aware that these are extensions that are not found in other implementations.

5.6.1 Echo

As the name says: The `EchoSubmissionHandler` just echoes the XML back that was sent to it. This may be useful for debugging.

This `SubmissionHandler` is by config registered to the URI scheme 'echo:'. As a full URI must have a non scheme-specific part you have to put an arbitrary string behind the echo though it's not processed in any way:

```
<xf:submission resource="echo:void" .. />
```

5.6.2 File

File is a standard protocol available in XForms. betterFORM extends the functionality by providing a directory listing. When the file you are referring to is a directory a directoy listing for that directory will be created in XML. This comes in hand when you need to browse a server-side filesystem.

The directory listing capability is present in the FileURIResolver and the `FileSubmissionHandler` meaning you can use it in `instance src` and `submission resource`.

> Please be aware of the security concerns that might arise from using this. Be cautious to allow browsing since the user might step through your whole filesystem.

5.6.3 SMTP

The SMTP submission driver serializes and submits instance data over SMTP (internet mail).

Currently, the driver only supports the `post submission` method and the `replace` mode none. The driver requires the additional information about the SMTP

server to use, the mail subject, and the sender. This information has to be passed in the query part of the submission's action URI. If you want the driver to authenticate a user with the SMTP server, just provide a `username` and a `password`.

Be careful when writing the submission's action URI: First, the contents of the query part has to be URL-encoded, then you have to replace all &'s with their corresponding XML entity & in order to keep the XML well-formed.

Example:

```
<xf:submission id='smtp'
      resource="mailto:nn@nowhere.no?
      server=smtp.nowhere.no&amp;amp;sender=xforms@nowhere.no&amp;amp;
      subject=instance%20data"
```

Another example with authentication:

```
<xf:submission id='smtp-auth'
      resource="mailto:nn@nowhere.no?server=smtp.nowhere.no&amp;amp;
      sender=xforms@nowhere.no&amp;amp;subject=instance
      %20data&amp;amp;username=xforms&amp;amp;password=shhh"
```

Support for other mail header fields like cc may be added later.

5.6.4 XMLRPC

You can connect to XMLRPC servers by using XMLRPCURIResolver and XMLRPCSubmissionHandler. These are both configured with URI scheme 'xmlrpc:'.

The `XMLRPCURIResolver` resolves `xmlrpc` URIs. It treats the denoted `xmlrpc` resource as a XML-RPC function, with parameters passed as a query string to the URI. The URIs look like this:

```
xmlrpc://server:port/path/[xmlrpcQuery]?[xmlrpcParameters]#frag
```

The function is called with the parameters in a hash table, and an XML Document is returned.

If the specified URI contains a fragment part, the specified element is looked up via `getElementById`. Thus, the parsed response must have an internal DTD subset specifying all ID attribute. Otherwise the element will not be found.

The XMLRPCSubmissionHandler parses the string given by the `resource` Attribute, extracts function and parameters from it and executes the remote function.

5.6.5 XSLT

The `XSLTSubmissionHandler` applies an XSLT transformation to the submitted instance and is useful when you want to apply some sorting or filtering of the instances.

For a base URI of e.g. `file:///Users/JohnDoe/Forms/bin` the submission URI `xslt:../xsl/my.xslt?foo=bar` will result in the following processing steps:

- After URI resolution the XSLT stylesheet is loaded from
- file:///Users/JohnDoe/Forms/xsl/my.xslt .
- A Transformer is created for that stylesheet and the uri param foo=bar is passed to the stylesheet as `xsl:param` 'foo' and its value 'bar'.
- The submitted instance is transformed using that Transformer.
- The transformation result is returned as a stream in the submission response.

6 Modularizing forms

Sooner or later when forms start to grow more complex it is time to think about how to avoid redundancies and keep the documents maintainable. If you push loads of data into a 5000 lines XForms document with potentially hundreds of controls you shouldn't complain about bad performance. Instead it's time to modularize your forms as you would do with any other code. As a sideeffect this will also have a very positive influence on runtime performance.

As XForms follows a MVC architecture the XForms model is the first logical candidate when decomposing larger forms into smaller pieces. Aggregating more complex forms from little snippets of UI (or snippets of a model) is a limited approach as the interesting parts are located on the bind Elements. This is where controls learn about their constraints, states, calculations and data types. Instead of just glueing pieces of markup together the inclusion of complete models allow the reuse of all the semantics defined within them.

This section describes two approaches of form (and instance) aggregation but will be focussed on aggregating complete XForms models with their appropriate UI or complete forms into larger documents. Though possible aggregation of separate little Model or UI snippets is not covered here.

6.1 Dynamic embedding (subforms)

> Attention:
> The feature described here is not part of the XForms standard and is therefore not portable to other implementations. The matter is discussed by the WG and it's likely that there will be a similar solution in XForms 1.2 [22]. Future releases may change the syntax or processing model as further specifications mature.

In complex applications that work with a multitude of different data structures the form that needs to be loaded for a given structure is sometimes not known before runtime. Especially if larger data structures have to be aggregated from a set of substructures (like XML Schema complex types) a mechanism is needed to dynamically load a subform for a given data structure, embed it into the running page and let the loaded part interact with the already loaded model(s).

22 W3C XForms Group Wiki: Load Embeding (http://www.w3.org/MarkUp/Forms/wiki/Load_Embedding)

Building 'super-forms' that incorporate all the logic and handle all the substructures only works to a certain extend. When forms grow into thousands of lines of markup a critical phase with regard to maintainability and extensibility is reached.

Another aspect is reuse. Subforms can be complete XHTML/XForms documents that run independently of the 'masterform'.

betterFORM provides a solution for these use cases through a tiny extension of the XForms '*load*'[23] action syntax. The XForms load action traverses an URI referenced by the value of the `resource` child Element or Attribute. It further provides an attribute 'show' which allows 'replace' and 'new' as possible values. 'new' loads the referenced URI into a new (browser) window while 'replace' replaces the current document with the content of the specified resource (thereby ending the running XForms session).

Example:

```
<xf:load show="replace">
  <xf:resource value="someURI"/>
</xf:load>
```

XForms borrows the `show` Attribute and the two possible values 'new' and 'replace' from the W3C XLink standard. XLink itself defines more possible values for the `show` Attribute. One of these is 'embed' which is adopted by betterFORM. Last thing needed is a way to say where to place the loaded subform into the document. BetterForm borrows the Attribute targetId Attribute from XForms for that purpose. The targetId is an idref to an Element which must exist in the document.

BetterFORM provides two different ways to reference a subform for embedding:

1. value="FileName.xml"
2. value="FileName.xml#Id_Of_Node_To_Embed

The difference is that for 2. the part after the fragment identifier ('#') is interpreted as an idref pointing to a node in the target document 'FileName.xml'.

6.1.1 Embedding complete documents

The following sample shows the load markup within a masterform to embed a **complete** subform into the specified target node 'inputMountPoint'

```
...
<xf:load show="embed" targetId="inputMountPoint">
    <xf:resource value="'Input.xml'"/>
</xf:load>
```

23 http://www.w3.org/TR/2009/REC-xforms-20091020/#action-load

```
...
<div id="inputMountPoint"/>
```

Executing the `load` action above loads and parses the resource denoted by the `resource` value (here Input.xml) and initializes it as XForms. The initialized markup then **replaces** the node specified by the targetid ( ).

To allow further reference to that node within your document the newly imported node will get the `id` of the Element referenced by `targetId`. The sample shows a complete XForms subform with its model and UI controls.

> The host document in this example uses a HTML snippet instead of a full HTML file. This is because you can't load an HTML document into the `body` of another without using iframes.

```
<div  xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xf="http://www.w3.org/2002/xforms">
    <xf:model id="m-child-1">
        <xf:instance id="i-child-1" xmlns="">
            <data>
                <item>false</item>
                <item>2009-10-01</item>
                <item>2009-10-01</item>
                ...
            </data>
        </xf:instance>
        <xf:bind  nodeset="item[1]" type="boolean" readonly="true()"/>
        <xf:bind  nodeset="item[2]" type="date"/>
        ...
    </xf:model>
    <xf:group>
        <xf:input ref="item[1]" model="m-child-1">
            <xf:label>item 1</xf:label>
        </xf:input>
        <xf:input ref="item[2]" model="m-child-1">
            <xf:label>item 2</xf:label>
        </xf:input>

        ...
    </xf:group>
</div>
```

> The document must have a single root node. Here the `model` and UI are wrapped into a HTML `div` Element giving a single handle to both parts.

6.1.2 Embedding models

Embedding complete documents is a start but degrades the subform to a piece of markup that only makes sense in the context of a 'master' or 'parent' form. It also complicates testing of the subforms as you are forced to run in 'embedded mode'.

Having forms that can act standalone and/or be embedded into others certainly offers new perspectives for modularizing an application.

This example shows how to load a piece of markup (which happens to contain a complete XForms model and UI) into a running XForms:

```
<xf:load show="embed" targetid="controlMount">
     <xf:resource value="'../reference/Output.xhtml#xforms'"/>
</xf:load>
```

Note the fragment identifier ('xforms') on the resource URI. This will be used as an idref pointing to an Element in the target document. Only this Element will be grabbed for embedding.

The approach to load a subform with a single idref enforces a slightly different authoring style than might be found in XForms examples on the net. Many people put the XForms `model` into the `head` of a XHTML document and the UI into the `body`. This is fine but a mere convention. The XForms `model` is perfectly allowed to live in the `body`. That makes no difference for the processing and is a key premise of the betterFORM implementation for the efficient embedding of forms.

The example below uses the Element with `id` 'xforms' to wrap `model` and UI in the `body`. This gives us the handle to fetch the model along with its UI in one go. The `model` itself is additionally wrapped with a `div` with `style="display:none;"`. That prevents the browser from becoming puzzled through the foreign markup.

The document for this example would look like this (shortened):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xf="http://www.w3.org/2002/xforms">
    <head>
        <title>Output Control</title>
    </head>
    <body>
      <!-- other content -->
      <div id="xforms">
            <div style="display:none;">
                  <xf:model id="foo">
                  ...
```

```
                    </xf:model>
            </div>
            <xf:group>
            <!-- XForms UI controls here -->
            </xf:group>
      </div>
      <!-- other content -->
    </body>
</html>
```

If there's already a `model` loaded which is a descendent of the target[24] Element that model will be unloaded by dispatching a XForms `model-destruct` event to each descendent `model`. After the `model`(s) have shut down the new form will be imported and initialized as specified by the XForms Specification by dispatching `xforms-model-construct`, `xforms-model-construct-done` and `xforms-ready` events.

To unload a previously embedded form dispatch a `load` action with the value 'none' instead of 'embed' for the `show` Attribute:

```
<xf:load show="none" targetid="controlMount"/>
```

Embedding works very fast and stable though the functionality also raises some questions:

a) how can models interact with each other?
b) how can models exchange data without breaking the rules?
c) how can id collisions be avoided?

We have found a few first answers for a) and b) leaving c) for later versions of betterFORM. For the moment the form author has to take care that id collisions do not happen.

6.1.3 Accessing nodes in a foreign model

Just loading models and putting them side by side into a processor at runtime is fine but also of limited value if you can't exchange data between these models.

Each model is handled within the XForms standard as a separate entity with its own data, dependencies, constraints, types, states and lifecycle. You may think of a XForms model as an object as you find it in object-oriented languages like Java. Thus the developers of betterFORM decided to take the conservative road by not 'breaking

24 The Element that has an `id` Attribute matching the value of Attribute `targetId`

the capsule' of the XForms model. The approach will evolve the more experiences with embedding XForms have been made.

To access a single node or a nodeset in a foreign model you can use the `instanceOfModel` XPath function. It works similar to the XForms `instance` function but takes the model id as first parameter.

The example shows how to copy the 'origin' node value 'foo' in model 'model-a' to the 'clone' node in model 'model-b':

This example shows how to copy a value from 'model-a' to 'model-b' with a `setvalue` action:

```
<xf:model id="model-a">
      <xf:instance id="a-default">
            <data xmlns="">
                  <origin>foo</origin>
            </data>
      </xf:instance>
</xf:model>
<xf:model id="model-b">
      <xf:instance>
            <data xmlns="">
                  <clone/>
            </data>
      </xf:instance>
</xf:model>
...
<setvalue model="model-b"
          ref="clone"
          value="instanceOfModel('model-a','default')/origin" />
```

This is a good and simple way of passing a few parameters between forms like modes, language settings and the like.

> Special attention should be paid to the fact that using `instanceOfModel()` ties your forms together. They will not work any more when a model with a matching id cannot be found at runtime. Conditional execution of the function can prevent this.

6.1.4 Exchanging instances between models

The `instanceOfModel` function is a convenient way if you only want to share small portions of data. The `ModelSubmissionHandler` allows to exchange larger quantities or whole trees of instance data between models. As the name suggests it

enables submissions between models that work just the same as every `submission` in XForms with full support of all `submission` Attributes.

The `ModelSubmissionHandler` supports the `GET` and `POST` `submission` methods to:

- GET - fetch data from a foreign `model` and copy them into a local `instance`.
- POST - send local `instance` data and update the foreign `model`.

Example a:

```
<xf:submission id="s-load-foreign-instance"
     resource="model:foreign#instance('contact')"
     replace="instance"
     method="get"
    model="local"/>
```

This `submission` will fetch the instance 'contact' from `model` 'foreign' and replace its own default `instance`.

> Of course you can use all other features of `submission` and select only a part of an `instance` and insert it at a target node in the replaced one.

Example b:

```
<xf:submission id="s-update-foreign"
resource="model:foreign#instance('contact')/data/postaladdress[@id='2']"
         replace="none"
         method="post"
        model="local"/>
```

This `submission` sends the default `instance` of the local `model` to `model` 'foreign' for updating but only the data will be inserted at a node 'postalAddress' in the foreign `instance`.

6.2 Static inclusion

Often it is sufficient to just aggregate larger documents from different source files instead of using dynamic embedding as just shown. This so called static inclusion takes place before the XForms processor receives the form document to process it.

Dynamic embedding gives you ultimate control over when to load and embed at runtime while static inclusion is just an aggregation (but nevertheless sometimes useful) mechanism like server-side includes or XInclude.

Why not use XInclude then? It turned out that with XInclude it's harder to control *when* the inclusion is wanted and to suppress it when it is not wanted. Further the Xinclude implementation support in Xerces did not make it easy to get to the desired results (e.g. limited XPointer support).

> The inclusion feature is not enabled by default in betterFORM. You have to activate it by editing the betterform-config.xml found in the WEB-INF directory of your installation. Change the value of property 'webprocessor.doIncludes' to 'true' and save the file.

betterFORM uses a XSLT transformation for the inclusion which is called before a new form session starts. At the moment betterFORM only supports fetching of complete documents and fetching of document fragments by id though extensions are made easy (e.g. selecting an Element with an XPath location path) by simply adapting the stylesheet.

Static Inclusion Example:

```
<bf:include src="foo.xhtml#bar"/>
```

When the document which contains this tag is requested the transform will be triggered, fetch the file 'foo.xhml' and extract the Element denoted by id from it. This Element will then become part of the source document and replace the `<bf:include />` Element.

7 Best practices

This section lists some unsorted tips that have evolved over the years of working with XForms to avoid common pitfalls.

7.1 Avoid (too) large forms

To a certain extend big forms are ok. But if a form grows bigger than 1000 or probably even several thousand of lines of markup you should begin to think about breaking it down to pieces. This is good practice for any programming language and also applies to XForms. XForms is a powerful tool and as this you can abuse it but also create nicely modularized pieces of markup.

XForms allows an unlimited amount of models within your document. Use this capability to tailor your forms. This will have major impact on reusability and performance as the rather expensive rebuild, recalculate, revalidate, refresh cycles run per model – many smaller models will perform much better than one giant form that handles all at once.

If that doesn't help enough please refer to Chapter 6 and try dynamic loading of subforms. Users have a limited capacity with regard to the amount of information they can handle at once therefore it's better anyway to only present smaller groups of controls.

7.2 Use UI bindings

XForms knows two flavors of bindings: UI bindings and model bindings. The former are represented by the Attributes `nodeset` or `ref` while the latter come as `bind` Elements. Bind elements are mainly used to attach constraints to instance nodes but can also be referenced from the UI.

Example of `model` / UI binding:

```
...
<xf:instance><bar xmlns=""/></xf:instance>
<xf:bind id="foo" nodeset="/bar" />
...
<xf:input bind="foo" id="model_binding">
```

```
...
<xf:input ref="/bar" id="ui_binding">
...
```

The input control 'model_binding' references the bind Element 'foo'. This attaches the node referred by the nodeset Attribute of the bind (foo) to the input control. In other words: the input indirectly binds to the instance. The bind Element will be the middle man.

Generally this is fine though it sometimes leads XForms beginners to the wrong assumption that referencing the bind from the input control is needed to attach the constraints specified on the bind. This is plain wrong: any MIP defined on a bind will be attached to nodes referenced by the nodeset attribute regardless of it being referenced by a UI control or not.

The problem with using model bindings comes when hand-authoring large forms. You always have to follow the indirection to know where a control binds to in the instance and which calculations and constraints are in place. This can take significant time of your development work. With the arrival of powerful XForms editors this problem should one day be solved[25].

UI bindings are easier to manage and quicker to understand. And by using scoped addressing the pathes won't get that long.

Example scoped addressing:

```
<xf:group ref="instance()">
     <xf:group ref="address">
          <xf:input ref="street">
```

The bound node for the input will be 'instance()/address/street'.

7.3 Don't forget about namespaces

Namespaces are a pain for many people though once they are in place there's nothing mysterious about them and you can easily leave them alone. To avoid problems put all the namespaces you want to use on the root node of your document. Then in case you know where you have to search.

25 For HTML it took many years before really useable editors became available.

7.4 Don't forget the default namespace

A very common mistake when writing XForms is to forget the empty namespace declaration on an `instance`. Any `instance` that does not use a specific namespace should look like this:

```
<xf:instance>
     <data xmlns="">
```

Alternatively you can put the declaration on the `instance` Element itself.

7.5 Avoid mixing of ui logic and form purpose

When developing complete applications in XForms you'll get a mixture of XForms markup that deals with the logic of the application UI and markup that deals with the actual purpose of the form (e.g. collecting addresses).

For instance, you may control access of certain controls by hiding/showing them, use XForms controls to implement wizard logic or activate/deactivate controls depending on some application mode.

You should be aware of this and plan ahead if your UI logic starts to get complex. Try to factor out a separate model for the UI logic to keep different things separate.

8 Debugging

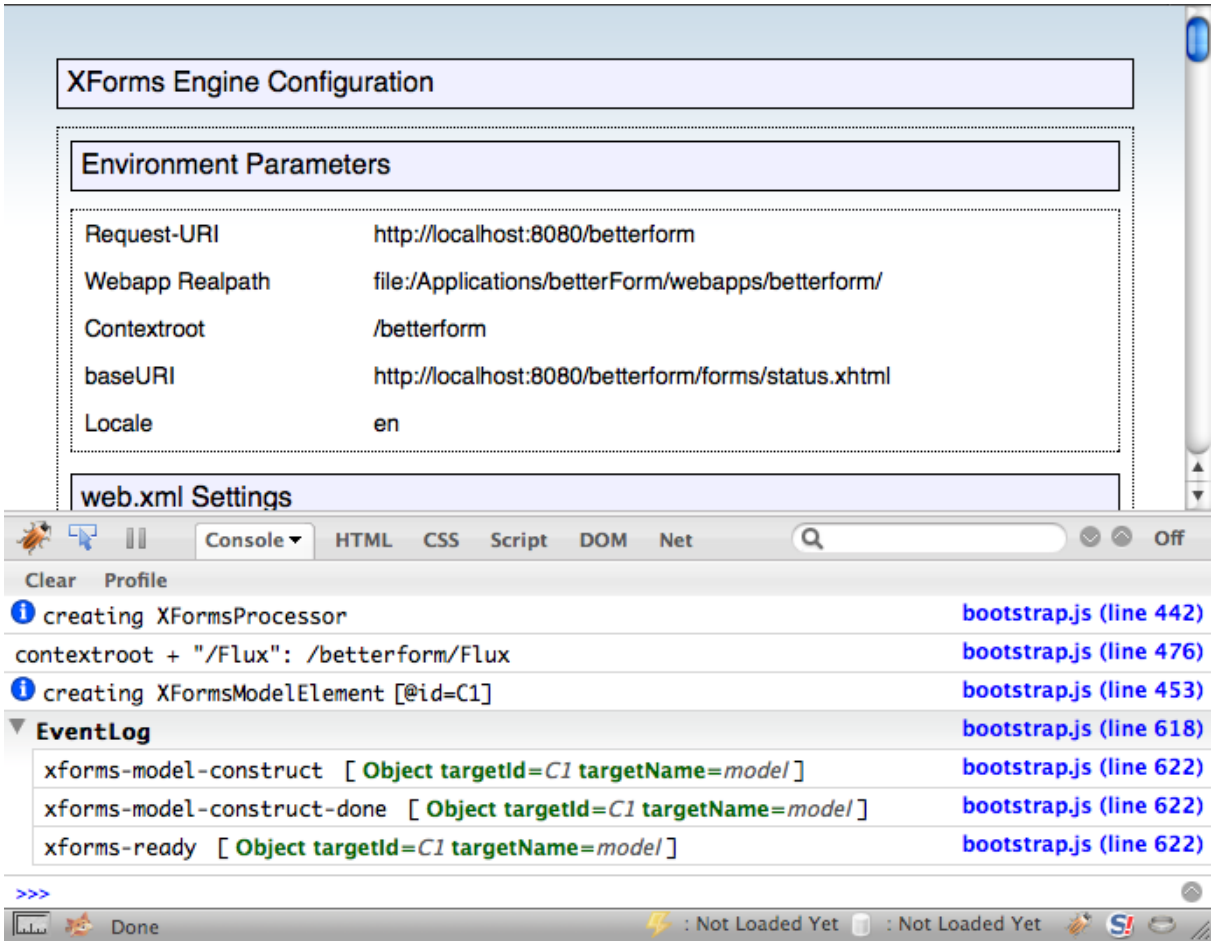
By default betterFORM is distributed with compiled JavaScript libraries. The optional 'DeveloperPack' installation pack[26] makes the uncompiled JavaScript resources available and enables debugging capabilities.

Before going into the details you should check, if the DeveloperPack is installed on your machine. You can easily detect this by looking for the directories 'dojo', 'dijit' and 'dojox' as direct children of directory BETTERFORM_HOME/resources/scripts. If these directories exist all resources needed for debugging are in place.

One of the best tools to debug forms is Firefox with the famous Firebug plugin installed[27]. The betterFORM code makes heavy use of the console feature of Firebug to output messages to the developer. Here's a sample screenshot where you can see the logging of XForms events plus some log info generated by our script libary:

26 Available with the web installer as an optional installation package.

27 Safari and Chrome also provide developer tools that evolve quickly and are comparable to Firebug. We have no specific tips for IE as it is not our platform.



Especially the EventLog will often be useful to find problems in forms. Furthermore all regular XForms exceptions like `xforms-binding-exception`, `xforms-compute-exception`, `xforms-link-exception` and others will be shown on the console. On the right side of each entry you find the Object that is associated with the respective event. By clicking the green text you'll jump into detail view and see all event context information that was returned by the `XFormsProcessor`.

9 Appendices

9.1 Configuration

betterFORM has a lot of configurable properties but only a few are worth mentioning in the context of a user guide. A detailed description of all properties can be found in the Appendix. All config files are placed within the WEB-INF directory which is a direct child directory of the betterFORM web application.

9.1.1 Deployment Descriptor (web.xml)

The Deployment Descriptor defines how the utilized servlet container deploys the betterFORM web application.

Here only the parameter value of the useragent is interesting for the user. This snippet is from the web.xml deployed with betterFORM:

```
      <filter>
            <filter-name>XFormsFilter</filter-name>
            <filter-
class>de.betterform.agent.web.filter.XFormsFilter</filter-class>
      <init-param>
                  <param-name>useragent</param-name>
                  <param-value>dojo</param-value>
            </init-param>
      </filter>
```

The default useragent for your installation is specified here. Normally this should have a value of '`dojo`' which means JavaScript and CSS resources are inlined, compiled and compressed. This results in faster loading and speeds up processing extremely. Other valid values currently are '`dojodev`' (for the uncompressed version) and '`html`' for the unscripted version of betterFORM.

9.1.2 betterFORM Config (betterform-config.xml)

Only one property of the betterform-config is of interest for the enduser – the preselected language.

From betterform-config.xml:

```
<!--
 Language selection rules at runtime:
 [1] 'lang' parameter on Url e.g. host:8888/foo.html?lang=en
 [2] presense of a 'lang' request Attribute
 [3] value of 'preselect-language' property below
[4] if above not present use accept-language header sent by browser
 -->
 <property name="preselect-language" value="en"/>
```

The desired language has to be specified by two-letters following the ISO 639-1 Codes.

9.2 Links

betterFORM Project related technical resources:

betterFORM homepage	http://www.betterform.de
betterFORM project page at sourceforge	http://www.sourceforge.net/projects/betterform
The Dojo Toolkit	http://dojotoolkit.org
DWR – Direct Web Remoting	http://directwebremoting.org

Selected XForms references and tutorials:

The official XForms homepage of the W3C	http://www.w3c.org/markup/forms
XForms Quick Reference	http://www.w3.org/MarkUp/Forms/2006/xforms-qr.html#Submission
XForms on Wikipedia (Wikibook)	http://en.wikipedia.org/wiki/Xforms

XForms book from Micah Dubinko	http://dubinko.info/writing/xforms/book.html
Other XForms implementations	http://www.w3c.org/markup/forms in section 'XForms Implementations'
A short introduction to XForms from Micah Dubinko (Author of the above book and member of the working group)	http://www.xml.com/pub/a/2001/09/05/xforms.html
XForms for HTML Authors	http://www.w3.org/MarkUp/Forms/2003/xforms-for-html-authors.html
XForms for HTML Authors, Part 2	http://www.w3.org/MarkUp/Forms/2006/xforms-for-html-authors-part2.html#Masterdetail1

Other:

FireBug-Extension for Mozilla FireFox	http://getfirebug.com/

9.3 Glossary

Connector

> *A Java class handling loading and/or saving of data from/to an arbitrary datasource. Connectors subdivide into URIResolvers and SubmissionHandlers. The first handles resolution of instances via the* `src` *Attribute while the second handles submissions.*

host document

> A document in any XML compatible markup language like (XHTML, SVG, VoiceML, FlashML) that embeds XForms markup

host language

> A XML markup language that embeds XForms markup. Must be at least well-formed in the XML sense.

Model Item Property (MIP)

One of the bind Attributes required, readonly, constraint, type or calculate which are used to attach constraints to node in the instance data.

UI

Short form for 'User Interface'