Last update: 16.12.2008 19:38:24
Author: Joern Turner <joern.turner@chiba-project.org>

Chiba UserGuide

1 - Introduction....................2

2 - Why use Chiba?....................3

2.1 - When to use it?....................4
2.2 - When to not use it?....................4

3 - Getting started....................5

3.1 - Preparations/Requirements....................5
3.2 - Installation....................5
3.3 - What you got installed....................6
3.4 - Default configuration....................8
3.4.1 - web.xml....................8
3.4.2 - chiba-config.xml....................8
3.5 - Deploy pages....................8
3.6 - Use the source Luke!....................9

4 - Building Smart Interfaces with Chiba....................10

4.1 - What's page authoring?....................10
4.2 - XForms Basics....................11
4.2.1 - Concrete Model and Abstract UI....................11
4.2.2 - Host language....................12
4.2.3 - A XForms skeleton....................12
4.2.4 - Anatomy of a XForms control....................13
4.3 - Some guidelines to writing forms....................14
4.4 - Authoring patterns....................15
4.4.1 - Direct layout....................15
4.4.2 - generic layout....................15
4.4.3 - application layout....................16
4.5 - Layouting pages....................16
4.5.1 - XForms Transcoding....................16
4.5.2 - Styling individual controls....................18

5 - Appendices....................20

5.1 - Configuration....................20
5.1.1 - default.xml....................21
5.2 - Links....................21
5.3 - Glossary....................22

1 Introduction

Chiba is a XForms framework and a toolkit that evolved around a XForms Processor implementation in Java (commonly called the 'Chiba Core'). While the first versions of Chiba started up with a server-side approach and a request/response interface it quickly became apparent how powerful the combination of AJAX and server-side XForms could be for applications especially XML applications[1].

Chiba Web 2.0 was the first attempt to implement that model. While interaction and dynamic behavior were good the use of native controls in the browsers also exposed some restrictions in styleability, extensibility and the integration of custom controls.

This guide covers Chiba Web starting from Version 3.0.0b1 and focusses on AJAX-enabled web applications. Version 3.0 marks a new generation of Chiba as it now allows to mix and match the power of XForms with the attractive features of Dojo. You're not an AJAX, Javascript, webapp programming multi-talent? Don't worry – Chiba is for you if you're willing to learn something new.

This is the Chiba UseriGuide[2] and as its name implies it's mainly targetted at people trying to quickly build a user interface or application without worrying much about the details. For those interested in the technologies and details under the hood please refer to our DeveloperGuide.

As Chiba builds on top of XForms and (for the time being) you'll need some XForms know how. This document is no XForms tutorial at all though some fundamentals will be mentioned. Please refer to W3C and Wikipedia as starting points for learning about XForms.

If you find mistakes, miss important topics or have suggestions please feel free to contact us via the Chiba users mailinglist.

[1] Whatever XML applications might be. Here it's just in the sense of 'if you use XML a lot'.

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

2 Why use Chiba?

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

- ***Based upon open standards:*** Chiba is addicted to open standards like XML, XForms, XPath, XQuery, CSS, Javascript, Java and XSLT. Chiba itself provides an open implementation of the XForms 1.0 + 1.1 standards.
- ***Quality:*** the Chiba project was founded in 2001 and was designed with modularity and extensibility in mind. The code has undergone many years of development, testing and improvement and has been used in dozens of projects in banking, commerce, science, bpm and ecm successfully. Over 500 unit tests assure proper operation of the Core processor and make sure that the code stays alive. Test coverage ranges somewhere at 80+ % for the whole Core and over 90% in the core packages (lots of the classes having 100%). Dojo on the other hand stands for a high-quality widget and layout system for really good-looking and accessible front-ends.
- ***Minimal-intrusive architecture:*** Chiba comes as a ServletFilter filtering requests to certain (configured) urls on your server. If your application somehow returns documents containing XForms markup Chiba will act upon them. This will be completely transparent to your application and unless you don't need special optimizations without any coding involved but by editing your web.xml.
- ***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 the XForm will be designed for a purpose (what it should do) and not saying HOW it's done[3]. 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 all without writing a single line of imperative code. Not stopping there you can still extend your pages with custom script, implement your own data-connectors or add your custom XPath functions for the last mile of requirements that a special-purpose application might have.
- ***Typing, validation and calculation:*** XForms provides strong datatyping for your app if you need it and takes care everything is valid and consistent in your data. Extension functions allow to bind even the most complex validators or calculators and allow to choose between different languages to implement them.
- ***ideally suited for REST and XRX:*** integration with RESTful services or architectures is a matter of minutes. In combination with XQuery even complex data-management tasks can be managed. When your data is already XML you can hardly get faster in development progress.

[3] See Wikipedia: http://en.wikipedia.org/wiki/Declarative_programming

2.1 When to use it?

- You heavily use XML in your backend and to quickly generate need user interfaces for data maintainance
- 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 Webservices
- 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, Internationalisation and Localisation are important to you
- + all the reasons listed under „Why use Chiba?“

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
- If you're experiencing anaphylactic reactions when getting in contact with XML you should stay away

3 Getting started

3.1 Preparations/Requirements

To run Chiba you'll need the following:

- JDK/JRE 1.5 or higher
- a Servlet 2.3 compatible webcontainer like Tomcat or Jetty

3.2 Installation

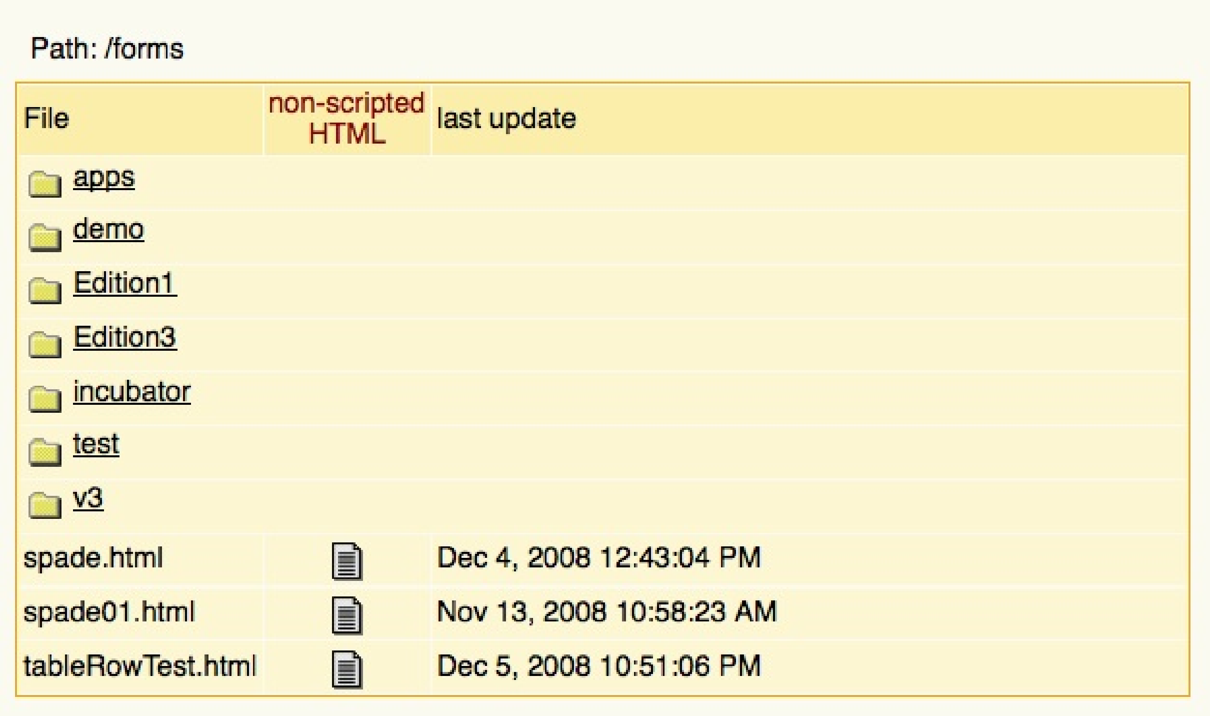
1.Download Chiba war file from Sourceforge http://sourceforge.net/project/showfiles.php?group_id=20274&package_id=154662

2.Move the Chiba war file to your the 'webapp' directory of your servletcontainer

3.Start the container – that's it

After startup the servletcontainer should unpack the Chiba war file and you may access the Chjba forms by browsing to http://localhost:8080/chiba-web. If you're experiencing problems please check our Wiki Installation page for futher details.

If you managed to start the servletcontainer and browse for the above URL you should see a screen similar to the picture below. The file names and directories you'll see may differ but here you have a very simple directory browser for your pages which you can use to put your forms by-the-side when starting your own experiments.

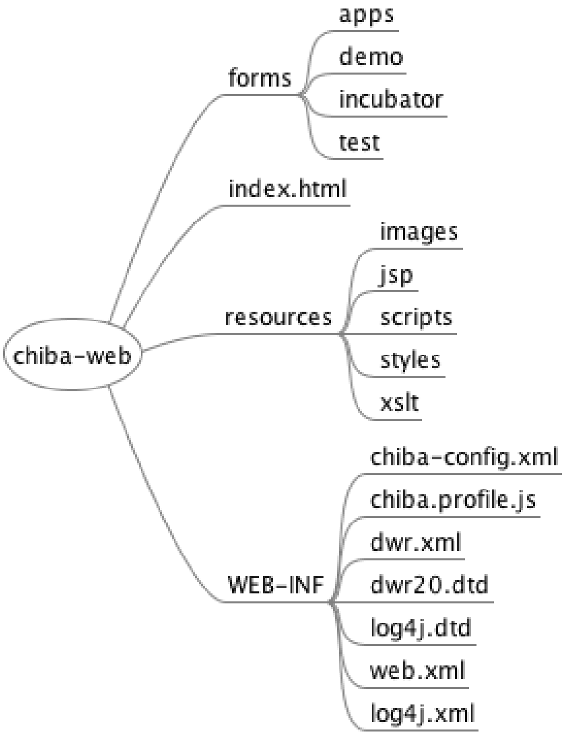


Path: /forms

File	non-scripted HTML	last update
apps		
demo		
Edition1		
Edition3		
incubator		
test		
v3		
spade.html	🗎	Dec 4, 2008 12:43:04 PM
spade01.html	🗎	Nov 13, 2008 10:58:23 AM
tableRowTest.html	🗎	Dec 5, 2008 10:51:06 PM

3.3 What you got installed

Below you see a graph of what you should have on disk after tomcat (or whatever servletcontainer you're using) has expanded the war during your first start of the server.



You should read over this before you start writing your own forms.

Here's a short description of the contents in the tree[4]:

Directoy/File	***Description***
forms/	This is the root of the forms coming bundled with Chiba. There a subdirectory structure which groups the documents in rough categories. You may simply create your own directories and pages here to keep them seperate[5].
index.html	This is the welcome page for Chiba Web.
resources/	In these directories Chiba stores all the things it needs for processing. You'll normally not touch these files unless you're a advanced user.
images/	as the name says – some pictures we're using as icons
jsp/	only a few pages to support the demo webapp. The forms browsing, an errorpage and a debug submission that you often like to use for testing.

[4] Don't worry about the red arrows on the picture – i've just taken a screenshot from FreeMind

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

Directoy/File	***Description***
scripts/	the Chiba Javascript Sources including a complete pre-build Dojo distribution.
styles/	here the default XForms and Chiba-specific CSS stylesheets are found. You don't have to touch these directly but can always use overwriting in your documents.
xslt/	Chiba uses XSLT to generate the Client UI. In this directory reside all involved stylesheets
WEB-INF/	
chiba-config.xml	contains all configuration information about Chiba
chiba.profile.js	is a Dojo profile for building a compiled version of the Chiba Javascript
dwr.xml	configuration file for DWR 2
dwr20.dtd	DTD for DWR
log4j.dtd	DTD for log4j
log4j.xml	configuration file for log4j
web.xml	contains the configuration of ServletFilter for Chiba

3.4 Default configuration

Chiba has a lot of configurable properties but only a few are worth mentioning in the context of a UserGuide. A detailed description of all properties in found as Appendix.

3.4.1 web.xml

Here's only one paramer interesting for the user. This sippnet is from the web.xml deployed with Chiba:

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

Here the default useragent will be set for your installation. Normally this should have a value of '`dojo`' which means you're using compiled and compressed Javascript which reduces processing extremely. Other valid value currently are '`dojodev`' (for uncompressed version) and '`html`' for the unscripted version of Chiba.

3.4.2 chiba-config.xml

There are only a one property really of interest for the enduser – the preselected language.

From chiba-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"/>
```

You have to use the two-letter ISO 639-1 Codes to specify the desired language.

3.5 Deploy pages

To deploy and run your own pages on Chiba please do the following:

1. locate the expanded Chiba war file structure on your disk e.g. '`/tomcat/webapps/chiba-web`'
2. step into directory 'forms'
3. create a subdirectory for your pages
4. copy your pages to this directory

5. reload /chiba-web/resources/jsp/forms.jsp in your browser

If you need additional resources (images, styles etc.) you may put them anywhere and reference them relative to the page.

3.6 Use the source Luke!

After you've installed Chiba you may want to have a look at the sample forms. We believe in learning by example and the best you can do is looking at the samples, copy and modify them and begin to understand ;) Thus we constantly try to maintain our demo pages (though not always looking nice) with each new release and show some working XForms functionality. They always use the up-to-date namespaces and syntax currently supported.

Please use our reference forms for questions regarding the syntax, the rendered result, the CSS involved etc. as we're putting special attention to these to make them a foundation for our testcases, a demo and a live reference where you'll find help texts showing the necessary markup and such.

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 may figure out some of the options XForms offers as well as discover some styling tricks.

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

4 Building Smart Interfaces with Chiba

Got a headache when thinking about the name for this chapter without raising wrong expectations. If you write 'interfaces' people associate GUI toolkits – which reaches far too short for XForms and what it can do. If you write 'applications' it sounds that there's some terribly complex machinery needed to tame the dragon. No matter which established buzzword you choose XForms falls between them.

XForms makes solutions to problems 'as simple as can be but no simpler' but can do much more than a plain GUI toolkit. It offers a rich model which helps to refine your companies smart contact form but also to implement complete application with complex logic requirements. The use cases range from simple forms to full-fledged applications with complex logic.

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

As already mentioned this document is not a XForms tutorial. Instead this chapter will show how to use Chiba 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 to page building. Everything you'll write are tags, attributes and CSS rules. You'll not need to write any Javascript or (beware) Java to make things run. You don't even need to understand how all AJAX, XForms magic is done behind the scenes. But you should have an understanding of XHTML, CSS and basic XForms. As you go you'll probably like to learn about Dojo.

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

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

- creating XHTML/XForms documents
- use XForms models to work with XML data
- use XForms @appearance to select between various layout options
- use Dojo layout containers for page layout
- adapt the rich set of pre-existing Classes to your needs
- add custom CSS matchers to finetune styling
- use XForms actions and events to add logic and behavior
- use XForms submissions to load or send data
- add validations and datatypes to certain data nodes

- add calculations
- provide help, hint and alert texts for user support

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 XForm Tutorial we nevertheless should clarify some basic XForms concepts and namings to avoid confusion later and hopefully help understanding the further explanations.

4.2.1 Concrete Model and Abstract UI

This is probably more a personal opinion of the author but one often underemphasized aspect of XForms is that it's 'concrete about the model and abstract for the UI'.

Let me explain: you may define a fine-grained model of your data along with validations, calculations and types. You may even use your own XSD-types or custom XPath functions and fix every detail of data consistency, loading and saving.

For the UI it's different: the XForms UI is abstract. XForms emphazises device-independence and by using tagnames for controls that avoid any association with visual display such as 'checkbox' or 'border'. This is because a form should be 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 when you're interested in supporting multiple clients.

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

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

4.2.2 Host language

The language of specifications sometimes becomes a bit arcane but some basic concepts are important to understand.

First:

> XForms is NOT a standalone markup language!

Instead XForms is an embedded language meant to be inlined into other XML markup languages. Of course the most prominent one is XHTML but there's nothing preventing you from embedding in VoiceML, SVG and whats more in the wide landscape of XML.

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.

Sorry for being loud but this point can't be overemphasized as it sometimes leads to confusion.

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 only 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.
- add a `<xforms:model>` tag to the head section of your XHTML document (or 'xf' if you have choosen that for your prefix). Actually there's nothing from preventing you to put this tag into the body of the document and it will work. It's a mere convention that has been evolved over time. But it's totally up to you where you put the model Element.

  The model itself has the following important children:
  `instance` – is the root Element for the data you work with
  `bind` – adds constraints and types to certain selected data nodes
  `submission` – defines loading and saving operations (interaction with some datasource)
  `action` – if you want to listen for events during model-initialization
- add some XForms UI to the body of your document (this time you really need to be in the body ;) 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

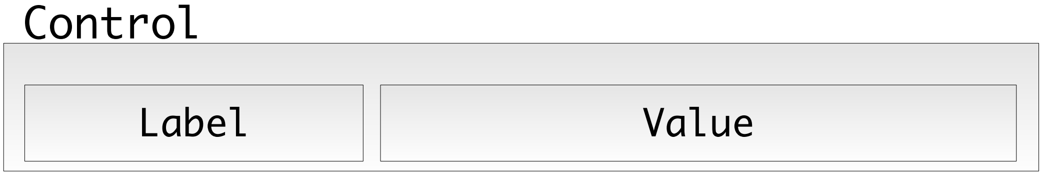
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xforms="http://www.w3.org/2002/xforms">
    <head>
        <title>hello</title>
        <xforms:model id="hello-model">
     (...)
        </xforms:model>
    </head>
    <body>
     <xforms:group>
 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

Chiba 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 Chiba.

4.2.4 Anatomy of a XForms control

Once the rough structures are set it makes sense to point out the differences to classical HTML forms with regard to controls.

In XForms any Control has a required label as a child element. In HTML you have @title Attributes and the <label> tag. In XForms context you have to think of the controls structure as shown below as being constituted by a label and an 'interaction' widget:



Sample Markup:

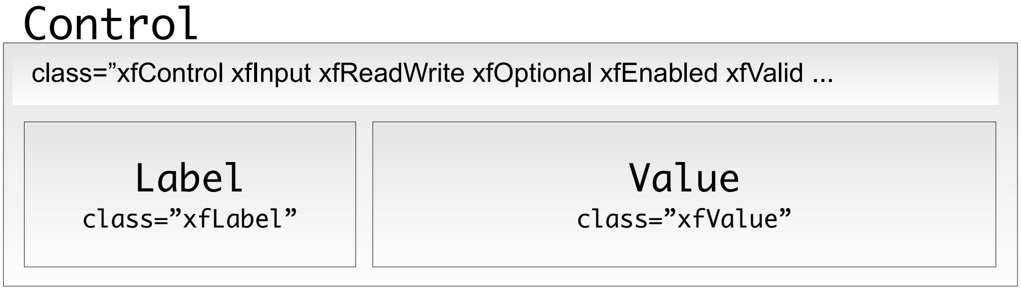
```
<xf:input ref=“foo”>
     <xf:label>bar</xf:label>
</xf:input>
```

The label Element above is mandatory and without it you wouldn't have valid XForms though most processors won't complain i guess. The reasoning behind is that the Working Group wanted to make sure that the accessibility of a form is always given[6].

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's important to understand and keep this struture in mind as the CSS system is relying on it giving you complete control on how certain states will be visualized.

Later on at runtime if you use Firebug or something similar you'll see that the control wears a lot of information represented as CSS classes. For the example above you'll get the following for the input:


Control
class=”xfControl xfInput xfReadWrite xfOptional xfEnabled xfValid ...
Label
class=”xfLabel”
Value
class=”xfValue”

For details about which classes will be assigned to each kind of control or container please refer to our CSS reference card for details which can be found under http://chiba.sourceforge.net/doc/CSSReference.pdf .

4.3 Some guidelines to writing forms

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

- Keep your markup simple
  When mixing HTML and XForms you should avoid to use table and other hardcoded stuff to layout your forms. This can better be done with CSS (well in almost all cases ;) But you should know the exceptions: if some data come in a tabular fashion you shouldn't avoid table at all cost – if they're a natural fit just use tables.
- Start with simple forms
  Start experimenting with simple forms and make them run before advancing to the more tricky parts like 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.
- Avoid common pitfalls
  When something goes wrong with your form check the most common points for failures:
  - you missed to declare a needed namespace

[6] Though this a valid and honorable argument it creates problems in practice sometimes with special layouts. It would be good if the group decides that `<xf:label for=“id”>` would work as in HTML.

- you forgot to declare a 'xmlns=““' (empty namespace) for your instance
- some bindings use XPathes that do not exist in your instance
- your instance cannot be found or loaded

4.4 Authoring patterns

There are two major styling 'patterns' (in the 'gang of four' sense) which have evolved to approach the styling problem:

4.4.1 Direct layout

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

4.4.2 generic layout

is signified by a XHTML document container that only provides minimal structure to embed the XForms markup. Most notably there will be a header and body section. The author will only write down the logical groupings of controls and the sequence in which they appear and avoid to heavily mix XHTML markup and XForms markup in the body of the document. By using appearance and CSS class/style attributes the author may give hints which can be used by the UI-Generator to add layout and styling information.

Dojo layoutcontainers like BorderContainer, TabContainer and others can be used to avoid the heavy testing necessary to make a layout look good on several browsers and make sure your UI scales nicely when resizing the browser window. Some of these containers will be readily available as XForms group for you to plugin into your page. Please study the reference forms. They show you the current set of supported appearances.

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 is similar to CSS in separating content form styling/layout. Most of the Chiba samples follow this pattern.

Please note that Chiba uses XHTML container documents as default. When using the generic layout pattern this does not pose any restrictions for the supported clients cause the styling hints can be used to create a complete different output document which is optimized for that client/device. Chiba is completely ignorant about the surrounding markup and does not process it in any way but to pass it on to the UIGenerator.

4.4.3 application layout

With Dojo it has now become realistic to say that there is a new authoring pattern: 'application layout' which applies at least for browser clients capable and willing to execute Javascript.

Dojo provides all the menues, toolbars, context menues, dialogs and of course layout containers to make your page behave and look like a real application. In such pages you'll more heavily mix Dojo components with XForms markup. As usually layout is something embedding the actual contents (the controls) both languages will not mix a lot. Instead typically you have Dojo on the outsite and XForms markup inside the panes defined by the container. This makes it possible to use some kind of inclusion mechanism (such as jsp:include, XSLT , Velocity or whatever) to further modularize your code. For details please refer to the DeveloperGuide.

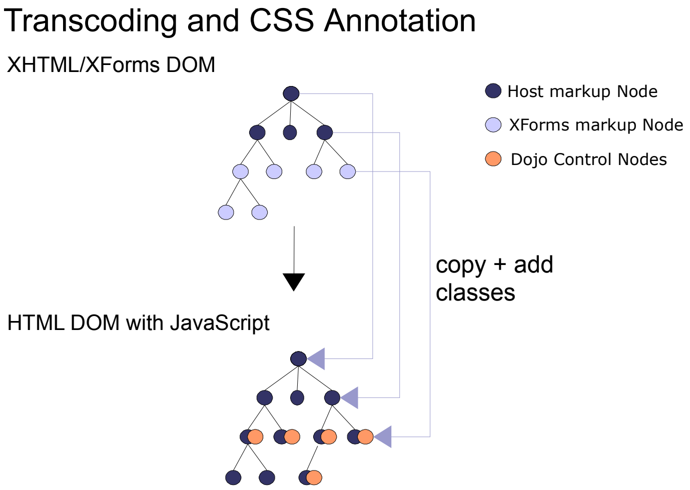
4.5 Layouting pages

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

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

4.5.1 XForms Transcoding

The graphic below show what Chiba does with the input host document after the processor has inited the model:



To turn the abstract XForms UI markup into something the browser can consume Chiba will transcode the page to HTML, CSS + JavaScript[7]. During this process every XForms control or container will be turned into some HTML (usually a simple <div>) along with some attributes and CSS classes. XForms controls turn into Dojo widgets which 'shadow' the semantics of the original XForms control.

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

While transforming Chiba assigns appropriate classes for all relevant XForms properties to the output document to allow flexible styling and layout through the use of CSS. This process is referred to as 'CSS annotation'.

An example in pseudo syntax will illustrate this best.

The following XForms input :

```
<xf:input id="my-input" class=‟myInputClass‟ xf:bind="somebind">
     <xf:label>your name</xf:label>
</xf:input>
```

will be transformed into:

```
<div  id="my-input"  class="myInputClass  xfControl  xfInput  xfReadWrite  xfOptional
xfEnabled          xfValid (...)">
     <label for="my-input" class="label">your name</span>
     <input type="text" id="my-input-value" class="value" />
</div>
```

provided that the modelitem-properties the node 'my-bind' points to evaluated to readwrite, optional, enabled and valid.

You can find a complete Chiba CSS Reference in a seperate pdf at: http://chiba.sourceforge.net/ChibaCSSReference.pdf

4.5.2 Styling individual controls

The above approach is fine for a generic styling of your forms. It makes re-use of layout- and styling information for multiple forms easy and should always be the preferred way.

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

Chiba will apply the original ids from your input XForms document to the output document. An example will illustrate this best.

For a this input markup:

[7] Of course with applies only to the 'dojo' useragent. But the handling of CSS applies in general also to the unscripted 'html' version.

```
<xf:input id=“myId“ ref=“foo“>

     <xf:label>bar</xf:label>
</xf:input>
```

the client will get the following output:

```
<div id=“myId“ class=“(...)“>
     <label id=“myId-label“ for=“myId-value“>bar</label>
     <input id=“myId-value“ ...>
</div>
```

By carefully assigning ids to your XForms document you lay the ground for fine-grained layout control. Using these ids you now have the handle to write CSS ID Selectors for styling every individual control. Another example shows how easy it is to write styles with this approach:

```
Somewhere in your XForms:
<xf:input id="title"....>
    <xf:label>Document Title</xf:label>
</xf:input>

In your CSS file:
#title{
    font-weight:bold;
}
#title .xfLabel{
    //put your label styles here
}
#title .xfValue{
    //put your control styles here
}
```

Unlike shown above it's convenient to put such rules into the header of the form itself inside a style section like this:

```
<head>
     <style type=“text/css“>
             /* put your custom styles here */
     </style>
</head>
```

This is the end for now. Creative input and comments of all sorts about this document are welcome and should be directed to the Chiba user mailing list or to the author joern.turner at chiba-project dot org.

5 Appendices

5.1 Configuration

All configuration parameters are found in a file named 'default.xml' which is located in package `org.chiba.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>chiba-config.xml</param-name>
      <param-value>WEB-INF/chiba.xml</param-value>
      <description>location of config-file</description>
</context-param>
```

ChibaServlet will expect the given location to be relative to the root of the webcontext.

5.1.1 default.xml

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 Chiba configuration. For documentation of these properties please see the comment in default.xml.
- `<error-messages>`
  used to configure the error-messages of the Chiba 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 Chiba 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.

5.2 Links

- Chiba homepage – http://chiba.sourceforge.net
- Chiba project page at sourceforge – http://www.sourceforge.net/projects/chiba
- Chiba Download – http://sourceforge.net/project/showfiles.php?group_id=20274
- Dojo Toolkit – http://dojotoolkit.org
- DWR – Direct Web Remoting – http://directwebremoting.org
- The official homepage of the W3C – http://www.w3c.org/markup/forms
- XForms on Wikipedia - http://en.wikipedia.org/wiki/Xforms
- interesting book about XForms from Micah Dubinko - http://dubinko.info/writing/xforms/book.html
- Links to 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

5.3 Glossary

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.