

# CaptainCasa & Java Server Faces



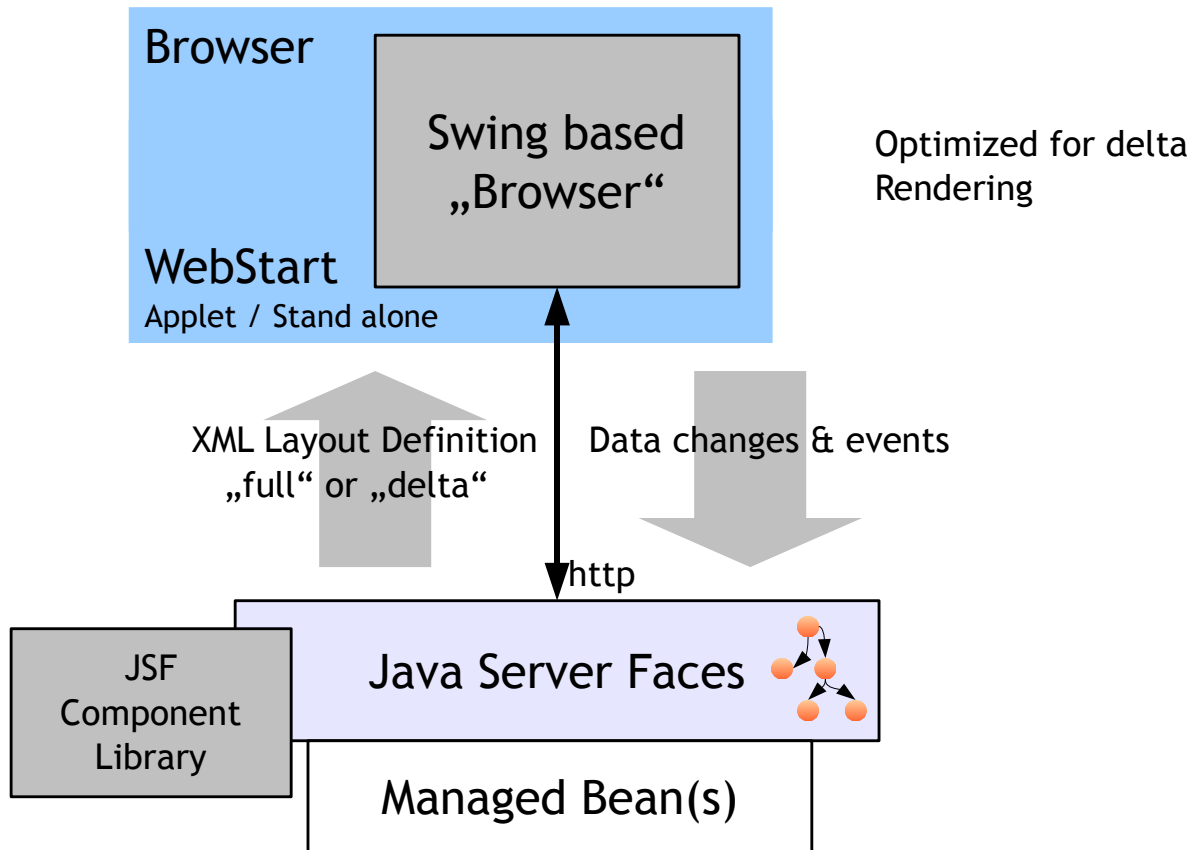
## Table of Contents

Overview.....	3
Why some own XML definition - and not HTML?.....	3
CaptainCasa Enterprise Client - A Browser for Enterprise Applications.....	4
...Java Server Faces joins the Scenario!.....	4
Java Server Faces is complex! ...?.....	4
What Java Server Faces means for CaptainCasa Enterprise Client.....	5
How CaptainCasa Enterprise Client uses Java Server Faces.....	5
Usage of Java Server Faces in more Detail.....	6
What we use - and do not use.....	6
The CaptainCasa Enterprise Client Component Library.....	6
Own Component Libraries.....	7
Usage of JSP Pages.....	7
JSF Expression Binding to Managed Beans.....	7
Non-Usage of Validation.....	8
Data Format Validation.....	8
Semantic Validation.....	8
Non-Usage.....	8
Non-Usage: Navigation.....	8
Non-Usage.....	9
CaptainCasa Corporate Community.....	10

# Overview

CaptainCasa Enterprise Client is a rich client framework consisting out of:

- A generic frontend client, implemented by using Java-Swing
- A backend connection through Java Server Faces (JSF)



This might first sound a bit strange - the normal mode of using Java-Swing in the frontend is to explicitly code a Java-Swing program for the client, that then calls server side functions via defined APIs.

CaptainCasa is different to this mode: with CaptainCasa the client is a generic rendering engine, that renders an XML layout definition coming from the server side. The client receives user input and - when receiving certain events that are relevant to trigger a round-trip - sends the user input as request to the server side processing.

Compare this architectural approach with the normal browser (Internet Explorer, Firefox, ...) way of processing screens: the browser receives HTML from the server, renders this HTML and returns back events as http-requests to the server. - This is exactly how CaptainCasa Enterprise Client operates, too. Only difference: instead of processing HTML, CaptainCasa Enterprise Client uses a certain XML definition that is sent from server to client.

## Why some own XML definition - and not HTML?

You might ask: when the CaptainCasa Enterprise Client is such similar to the browser processing, why is there a necessity at all to define a new type of client with an own XML layout definition?

There are two good reasons:

- CaptainCasa components are more high-level components than are HTML components. With one XML component tag, you directly define a complex component - and do not have to assemble the component out of low-level HTML components.
- And, even more important: The XML layout definition is sent based on a delta protocol. This is the big advantage compared to HTML. An HTML page is always sent as a whole page - consequently the browser needs to render this page as a whole page. - With CaptainCasa Enterprise Client only layout changes are sent from the server to the client - resulting in a much lower usage of network resources. And resulting in a much faster rendering on client side, because only the changes within the layout are processed.

## CaptainCasa Enterprise Client - A Browser for Enterprise Applications

You might already see, what CaptainCasa Enterprise Client is designed for: enterprise applications. ...the ones used by power users, spending a great part of their working time in front of their applications.

CaptainCasa is not a zero-installation client, compared to HTML/JavaScript clients. It's very simple to install, but requires a Java plug in on client side. We call this type of installation a "managed installation".

### ...Java Server Faces joins the Scenario!

Because the frontend part of CaptainCasa Enterprise Client technically behaves like a normal browser, all the pieces of technology that exist on server side in order to render HTML can now be used as well - now for rendering XML, of course.

Looking into what Java offers on server side, there is a simple and clear selection of what to use: Java Server Faces (JSF). Java Server Faces is a server side user interface framework, that from the beginning on was designed to serve default HTML-browser clients, but to also be able to serve any other type of client as well.

The base of Java Server Faces is the management of a server side tree of components. The tree is built up, e.g. using a static JSP page declaration and then is processed in the follow on processing: events and data changes on client side are transferred to their component instances on server side, triggering corresponding data updates and function calls within connected objects ("managed beans"). Finally the tree is rendered by recursively rendering each component, the result of rendering (in our case: the XML) is sent to the client to be physically rendered there.

### Java Server Faces is complex! ...?

Many developers think about Java Server Faces to be a complex framework. Are they right?

Ad hoc response: well, yes, they are. You need to dig into a lot of parts of Java Server Faces to get a feeling how to use it for your application development.

But...: Java Server Faces solves a complex problem! As a standard framework it needs to be flexible - covering different types of clients, luckily!, connecting to different types of applications, being open to integrate new user interface components, providing different types of functions apart from the component tree management: validation, navigation, etc.

So, from our opinion, the complexity of JSF meets the complexity of what JSF provides.

And: you do not have to use everything that is part of JSF.

You should not pass JSF to your application development directly, but need to select what parts of JSF you use in which way. JSF needs to be tailored before getting used by non-JSF-experts.

This is exactly what CaptainCasa does. JSF is used internally, but tailored to the needs of enterprise application developers. Our users like JSF! - because they see JSF in a filtered and structured, guided way, and because they get tools (most important: the layout editor) to efficiently deal with JSF.

## What Java Server Faces means for CaptainCasa Enterprise Client

Java Server Faces takes a huge load of CaptainCasa Enterprise Client: our non-standard browser-client is set on top of a 100% standard JSF server side framework.

Would you trust some motivated developers to build a frontend client? Yes, we hope so!

Would you trust these guys to build the server side framework to connect this client to your applications... - having in mind all the “ugly things” like load management, fail over management, security management. Well, we are not so sure about this...

So, JSF is our server side framework, being used in thousands of applications already, ranging from small ones to huge ones. JSF is accepted as trust-able server side framework.

## How CaptainCasa Enterprise Client uses Java Server Faces

First, to make sure: CaptainCasa **uses** Java Server Faces. We are NOT a provider of a Java Serve Faces implementation! - we are users of existing JSF implementations.

There is one major thing that we bring into Java Server Faces: a library of new components. These components are able to render themselves into the XML that is expected by the CaptainCasa frontend client.

The component library - that's the most important thing!

And there are other things that we bring in as well:

- Utility functions for simplifying the usage of JSF for application developers
- Frameworks to flexibly build and structure the managed bean layer
- Cross application functions, e.g. to build up application workplaces
- Design time tools for creating the layouts

CaptainCasa is built 100% on top of Java Server Faces. - Vice versa, this does not mean that we use 100% of all the features of Java Serve Faces! We do, what we described before: we tailor JSF for developers of enterprise applications. So the result is a simple, efficient way of building a rich client user interfaces.

In the following chapter we describe how we use JSF in more detail.

# Usage of Java Server Faces in more Detail

## What we use - and do not use...

The following functions are heavily used:

- Design of own component library + possibility to add 3<sup>rd</sup> party component libraries for serving CaptainCasa Enterprise Client
- Build up of initial component tree from JSP page, internally containing a layout definition
- Binding of component attributes to managed bean properties, binding of component events to managed bean action listeners
- Internal functions: possibility to add own variable and property resolver
- Direct manipulation of component tree by application at runtime

The following functions are explicitly NOT used:

- Validation
- Navigation

Other JSF functions can be used, but require a certain “JSF-awareness” of application developers.

CaptainCasa is currently delivered using the reference implementation of JSF 1.1. It is tested against the JBoss JSF implementation as well. We recommend to stay with the reference implementation, though.

## The CaptainCasa Enterprise Client Component Library

All components that are available on client side are reflected by a corresponding component within the component library.

Each server side JSF component is able to:

- ...save/restore its state
- ...analyze the http request in order to check for data updates or events
- ...allow the definition of JSF expressions to bind to attributes / events
- ...render itself into XML that transfers the component's state to the frontend client

The rendering is done in two phases, within the JSF rendering phase:

- Each component renders itself into XML. While rendering, the component only renders these attributes that have changed during the current request processing. The XML is not written to the http response immediately, but is stored within an interim object structure.
- After each component has rendered, the interim object structure is checked for unchanged “XML areas”. This means, that if the layout has not changed during the request processing within “big parts of the layout”, then these parts are not transferred at all to the client.

As result the data volume of a response is low. If there's no change on the page, then there is only few data that is sent to the client.

## Own Component Libraries

The CaptainCasa Enterprise Client can be extended on client side easily - by adding new Java-Swing components. When extending the client, then these new components need to be reflected by corresponding server side JSF components as well.

In this case users create a new JSF component library that is used in addition to the CaptainCasa JSF component library.

## Usage of JSP Pages

JSP pages are used to hold static layout definitions. Within the JSP page the CaptainCasa component library is registered and the layout is kept as a tree of XML tags.

Starting the CaptainCasa Enterprise Client frontend is always associated with a JSP page that is referenced from server side - just the same as starting a JSP page from the normal HTML browser.

The JSP page is transferred into a component tree by JSF, the component tree is rendered and the XML result is sent to the client to be rendered physically.

In addition JSP pages are used later on to be included into other pages. The including is done by a corresponding component (“rowinclude”). When being included into a certain area of another page, then the JSP pages is parsed by a CaptainCasa parser, the XML is transferred into components that then are plugged into the existing component tree. - There is no usage of the “jspinclude”.

The CaptainCasa include-parser only parses pure component trees, i.e. it does not parse any type of server side Java-scripting. This is the technical reason, why pages may not use Java-scripting. - Apart from the technical reason, we do not like scripting in layout definitions at all. This opens up an area of flexibility, that is way apart from a component tree management. It leads to JSP definitions, in which you find tons of Java code around component definitions... uaaah.

## JSF Expression Binding to Managed Beans

This, of course, is strongly used: component attributes/ events are bound to managed bean properties and action listeners. I.e. data from the client is transferred into corresponding beans properties, and events from the client trigger the calling of corresponding action listener methods.

The way we propose to use managed beans in a structured way is to use so called dispatcher beans as entrance points into the “managed bean space”. These are the ones defined in faces-config.xml (“#{d}”). From these dispatcher beans you then navigate into the next level of beans, e.g. representing the logic behind a page (“#{d.address}”). Beans accessed by the dispatcher can communicate with one another through the dispatcher. The dispatcher bean serves as object factory.

As consequence of this managed bean structure we introduced an own property binding: in the default property binding each expression is processed during the render-phase always from the scratch. If there is one expression “#{d.address.communication.email}” and there is another expression “#{d.address.communication.phone}”, then both expressions by default are evaluated with resolving “d”, then “address” and then the last property.

The own property resolver is called a “stacked property resolver”: it keeps in mind a stack representing the last expression that was resolved. In case the next expression is “quite similar” then the stacked values are used to much quicker find the corresponding new value. - Example: after having resolved “#{d.address.communication.email}” the

next expression “#{d.address.communication.phone}” is not resolved by starting with “d”, but by directly accessing the “phone” attribute from the stacked “communication”-object.

## Non-Usage of Validation

### Data Format Validation

In a lot of scenarios the JSF validation is used to check the data format of client side input. Example: if the JSF component restricts the user to input an integer value only, then the string that is coming from the user interface is checked to be a string before being passed to a managed bean property.

The background for this type of validation is, that the components on client side are quite low-level components.

With CaptainCasa the components on client side are high level - and are validating the data format (or e.g. regular expressions) directly within the frontend. There is no necessity to do pure format checks on server side.

### Semantic Validation

A next type of using validations is to semantically validate user interface data before passing it into the application layer.

This type of usage from our usage mode should be done on application level. An application (represented by a managed bean), needs to check input anyway - because it must never trust the UI level to provide consistent data. A big part of what an application does is checking data - on single property level, on object level, on cross-object level. We believe and propose it's most healthy to execute all validations within the application layer, and not have a mixture out of UI-validations on application-validations.

### Non-Usage

Result: we are not using JSF validations: format validations are to be done in the client, semantic validations are to be done on application processing layer.

## Non-Usage: Navigation

We believe the navigation rules proposed by JSF are reasonable to serve many scenarios, but we have problems to synchronize our users' navigation demands with the JSF navigation concepts. We see a certain difference between the typical web-navigation-scenarios and rich-client-navigation scenarios.

Example: a user may work within two screen areas in two decoupled application functions: on the left a customer master is maintained, on the right an order is created for the customer. Both have their own navigation rules, including popups etc. - and there is an outside navigation management on top of both functions.

The navigation concept of CaptainCasa is very basic on the one hand, but very simple and flexible on the other hand. It just bases on the fact, that each page can be included within another page dynamically. A page-switch is done by exchanging the JSP page within the outer page. (The include is not done by “jspinclude” but by an own include-component, see above: “Usage of JSP Pages”).

## Non-Usage

Result: we are not using JSF navigation - but leave the design of the navigations concepts (others would say: the design of the controller structure) to the application development, based on a very simple way of including one page within another page. As consequence of this you e.g. will not find “action”-attributes within the CaptainCasa component tags, but will only find “actionListener” attributes.

# CaptainCasa Corporate Community

CaptainCasa is a company on the one hand, but represents a community of mid range software companies on the other hand. These companies are using CaptainCasa Enterprise Client as their rich client framework for their enterprise applications.

The CaptainCasa Community is open, i.e. members may easily join and address their issues.

CaptainCasa Enterprise Client can be used from “for free” (no service, only binary image) up to “community licensed” (with service, with sources).

We are using big parts of Java Server Faces already, but are always a bit surprised if people tell us about certain functional areas of JSF that we did not touch intensively so far... - so we are open to react on JSF demands! - We are positioning the server side part of CaptainCasa Enterprise Client to be “on top of JSF”. Aside some areas that we explicitly name (navigation, validation), you should be able to use JSF functions that are not tailored/proposed by ourselves.

In case you find problems: please contact us, best via our community forum - Thanks!

CaptainCasa GmbH

Hindemithweg 13  
69245 Bammental

Tel +49 6223 484147

<http://www.CaptainCasa.com>  
[info@CaptainCasa.com](mailto:info@CaptainCasa.com)