

*xcentric technology & consulting*  
Oberwiesenstrasse 5  
CH - 8057 Zurich  
Switzerland

+41 (0)43 255 01 69  
[www.xcentric.ch](http://www.xcentric.ch)  
[info@xcentric.ch](mailto:info@xcentric.ch)

# JAXFront<sup>®</sup>

# HTML Rendering

## Developer Manual





### V2.61

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

## Document description

|                          |   |   |  |                                 |
|--------------------------|---|---|--|---------------------------------|
| <b>Author(s)</b>         | M.Leber, S.Portmann<br><a href="mailto:info@jaxfront.com">info@jaxfront.com</a>   |   |  |                                 |
| <b>Version</b>           | 2.61  |   |  |                                 |
| <b>Web link</b>          | <a href="http://www.jaxfront.com/download/JAXFront-HTML-Rendering.pdf">http://www.jaxfront.com/download/JAXFront-HTML-Rendering.pdf</a> |   |  |                                 |
| <b>Classification</b>    | <input checked="" type="checkbox"/> not classified  | <input type="checkbox"/> internal             | <input type="checkbox"/> confidential                  | <input type="checkbox"/> secret |
| <b>Processing status</b> | <input type="checkbox"/> draft/in processing  | <input type="checkbox"/> Ready for acceptance | <input checked="" type="checkbox"/> definitive version | <input type="checkbox"/>        |

## Conventions used in this document

| Image   | Meaning                                    |
|---|--|
|    | Points to an example-syntax in XML.        |
|   | Points to a code example in Java.          |
|  | Refers to important information.           |
|  | Describes a previously presented example.. |

Copyright © 2001-2009 xcentric technology & consulting GmbH. All Rights Reserved.

Use of this documentation and related software is governed by a License Agreement. This document does not imply a legal contract. Readers assume responsibility for the use and interpretation of the information contained herein. xcentric technology & consulting GmbH strives to ensure the accuracy of the provided instructions but it does not accept any liability or damages for omissions contained in this manual.

# Index of Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>INTRODUCTION</b>                          | <b>4</b>  |
| 1.1      | REQUIREMENTS                                 | 4         |
| 1.2      | HOW TO INSTALL                               | 4         |
| 1.3      | LICENSE FILE                                 | 4         |
| 1.4      | IMPORTANT TO KNOW                            | 4         |
| 1.5      | HOW TO TEST YOUR INSTALLTION                 | 5         |
| 1.6      | LAUNCH FROM XUI-EDITOR                       | 6         |
| <b>2</b> | <b>ARCHITECTURE</b>                          | <b>7</b>  |
| 2.1      | AJAX BASED TECHNOLOGY                        | 7         |
| 2.2      | MULTI CHANNELING                             | 8         |
| 2.3      | JAVA CLIENT VS. HTML FRONTEND                | 9         |
| 2.4      | USING JAXFRONTSERVLET VS. EMBEDDING JAXFRONT | 10        |
| 2.4.1    | <i>Option 1: Using JAXFrontServlet</i>       | 10        |
| 2.4.2    | <i>Option 2: Embedding JAXFront</i>          | 11        |
| 2.4.2.1  | Simple integration                           | 13        |
| 2.4.2.2  | Fully own API integration                    | 13        |
| 2.4.3    | <i>Implement own form actions</i>            | 15        |
| 2.5      | LIFECYCLE                                    | 17        |
| 2.5.1    | <i>Starting a session</i>                    | 17        |
| 2.5.2    | <i>Ending a session</i>                      | 17        |
| 2.5.3    | <i>What is in a session?</i>                 | 17        |
| 2.6      | GENERAL LAYOUT STRUCTURE (DIV LAYERS)        | 18        |
| 2.7      | IMPLEMENT YOUR OWN PLUGINS                   | 20        |
| <b>3</b> | <b>LIMITATION</b>                            | <b>21</b> |

We are looking forward to hear more from you.

We are interested in getting your feedback. Please send any comments/features/wishes or questions to [info@jaxfront.com](mailto:info@jaxfront.com) or call our hotline +41 (0)43 255 01 69.

# 1 Introduction

Instead of generating heavy weight DHTML components, this renderer provides a simple way of generating light weight HTML components.

The main goal is to give the user the ability to extend the system in different ways:

- a) plugin their own HTML code
- b) implement their own actions to retrieve & store the XML
- c) adding their own look & feel (CSS)

## 1.1 Requirements

- Java 1.4 or higher
- Servlet Runtime (Java Servlet 2.3+)
- Browser: Internet Explorer 5.5+, Mozilla 1.7+ or Firefox 1+, Safari, Google Chrome
- Browser must have JavaScript enabled

## 1.2 How to install

- 1) Put the `jaxfront.war` file into the `webapps` directory of your application server (i.e. `c:\tomcat\webapps`). Be sure you delete the "old" directory called "jaxfront" first.
- 2) Restart your webserver

It is not necessary to install the `jaxfront.war` file, one may use JAXFront embedded in its own servlet. See chapter 2.4 for more details.

## 1.3 License File

The JAXFront HTML renderer runs initially with the free community license.

If you have received a license file, replace the community license with your own license file. Locate the installation directory of your WebApplication server (i.e. Tomcat) and navigate to the "webapps" folder. Go to the directory "jaxfront/lic" and replace the file "jaxfront-community.lic" with your own license file. Be sure you delete the community license.

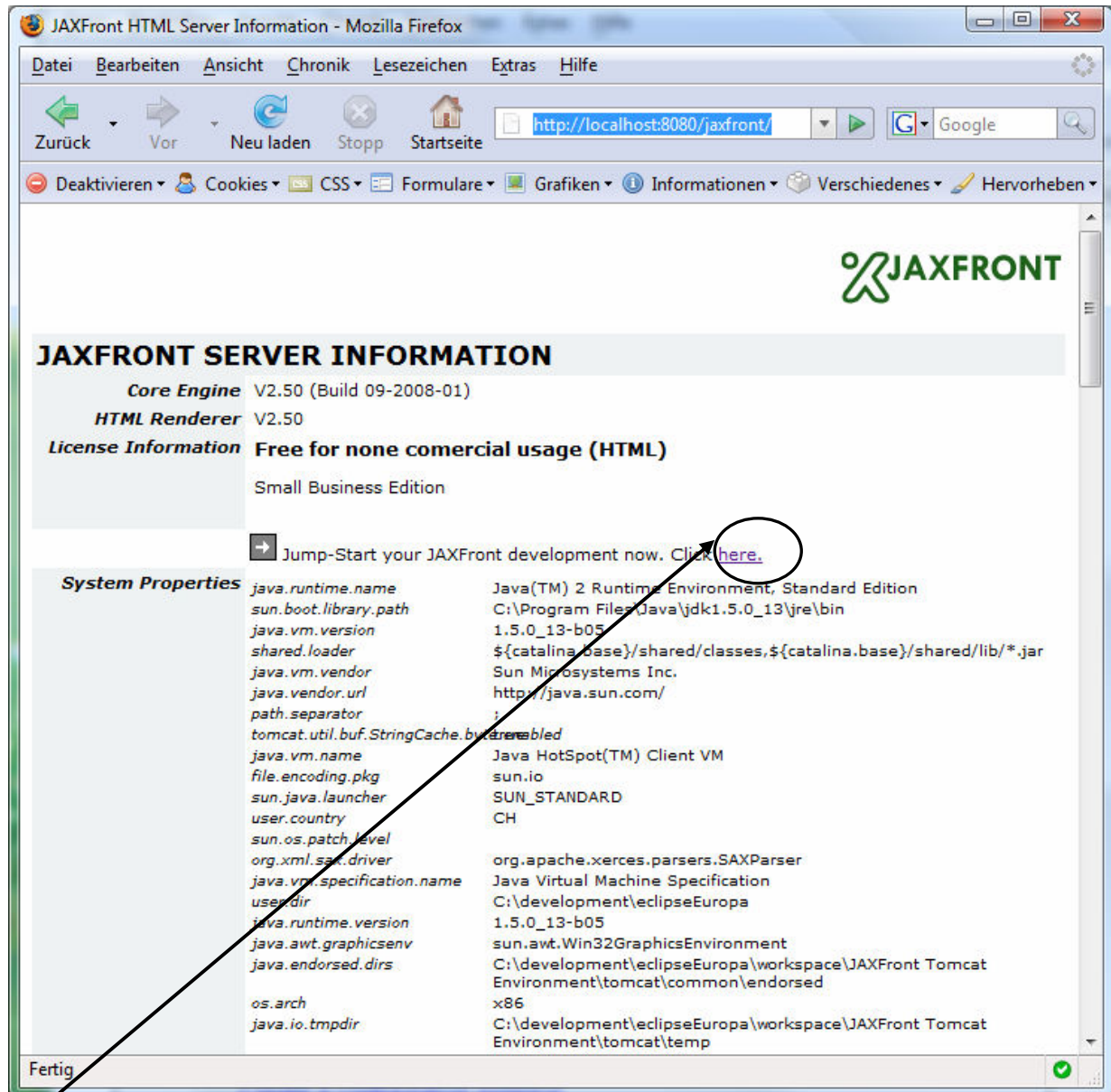
## 1.4 Important to know

- The HTML renderer does not support all XUI features.
- Tomcat uses old Xerces XML-Parser libraries. If you get troubles, replace the jars in the "tomcat\common\endorsed" folder with the ones in the war-file.

## 1.5 How to test your installation

After successful installation of the JAXFront HTML Server (Servlet), point your browser to the following location: <http://localhost:8080/jaxfront/>

You should now see the following within your browser.

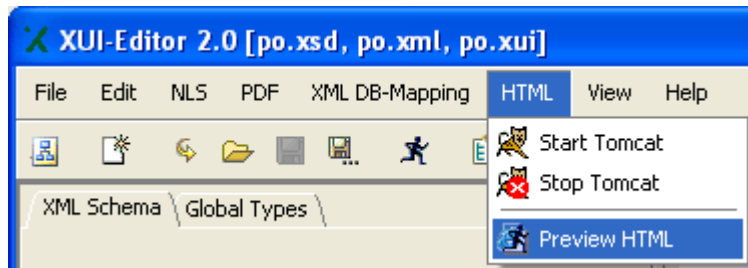


Click on the this link to jump start your JAXFront experience. Visualize any of your own XML Schema with the JAXFront HTML renderer.

## 1.6 Launch from XUI-Editor

If you have the XUI-Editor installed with the HTML extension, you should see a top level menu called 'HTML'. To test your web forms, do the following:

- 1) load any XSD (XML, XUI)
- 2) be sure your webserver is started (otherwise start your local tomcat server)
- 3) select 'Preview HTML' from menu 'HTML'

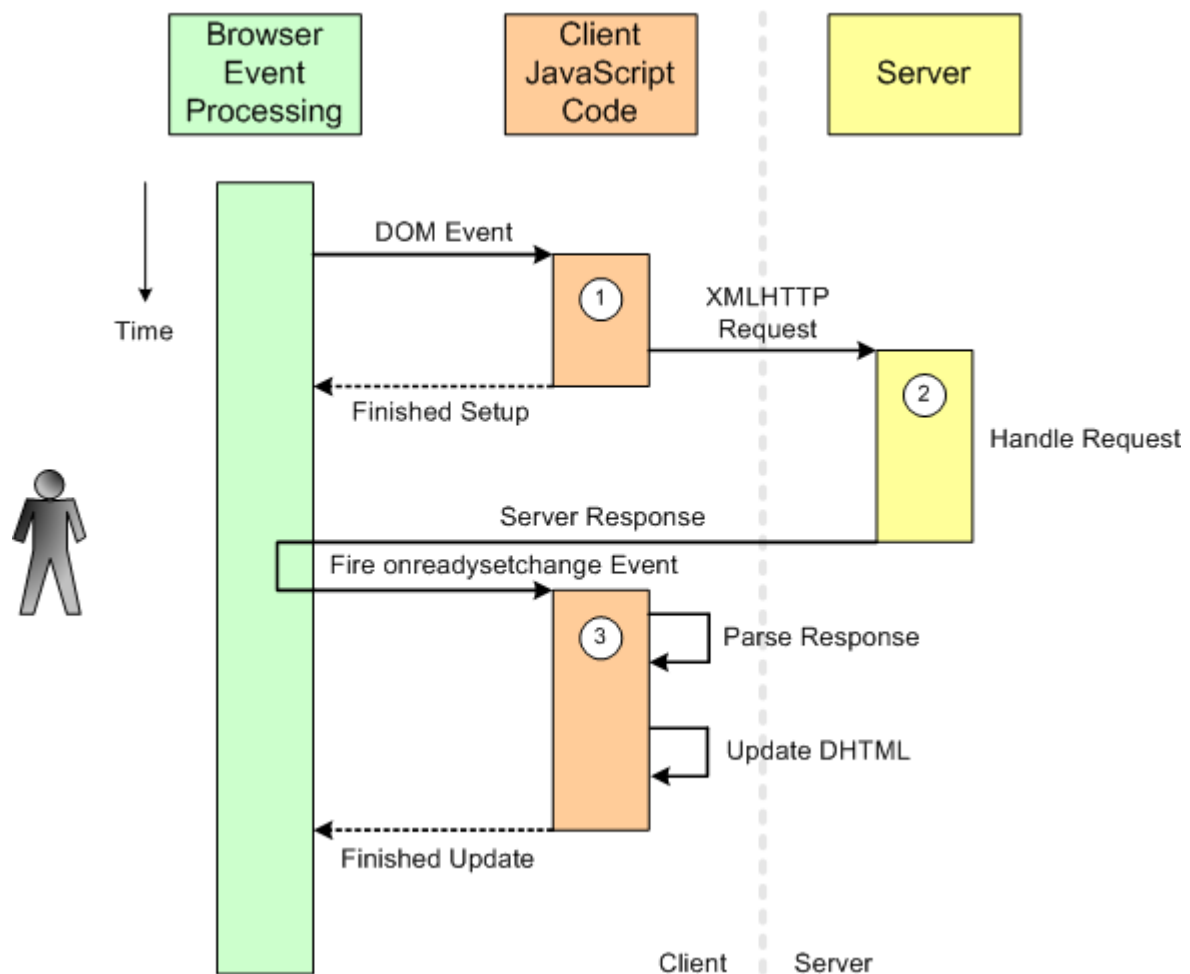


## 2 Architecture

### 2.1 AJAX based technology

The JAXFront HTML Renderer is a J2EE compliant AJAX technology. The renderer runs within a servlet container on the server side.

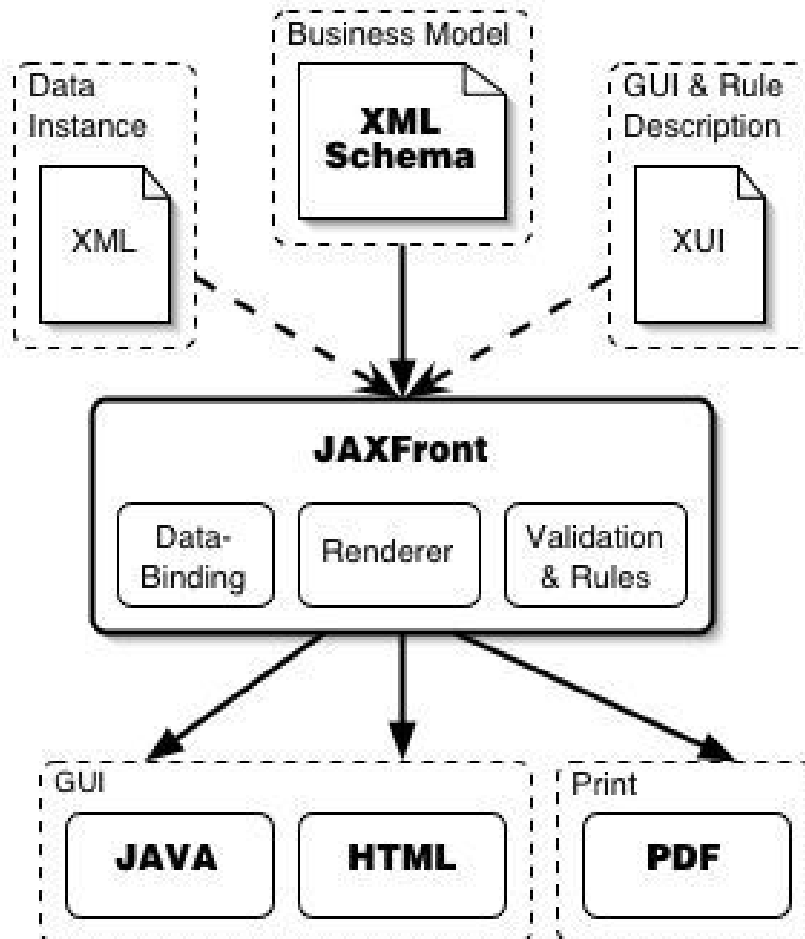
The client-server communication is realized with AJAX technology (asynchronous JavaScript/XML). Instead of having synchronous blocking HTTP request, AJAX uses asynchronous none-blocking HTTP request. AJAX is communicating with browser embedded AJAX components.



Asynchronous calls allow the user to interact with the frontend anytime and are not blocked by calling server roundtrips. Most UI updates occur through asynchronous updating of the HTML DOM with local JavaScripts.

## 2.2 Multi Channeling

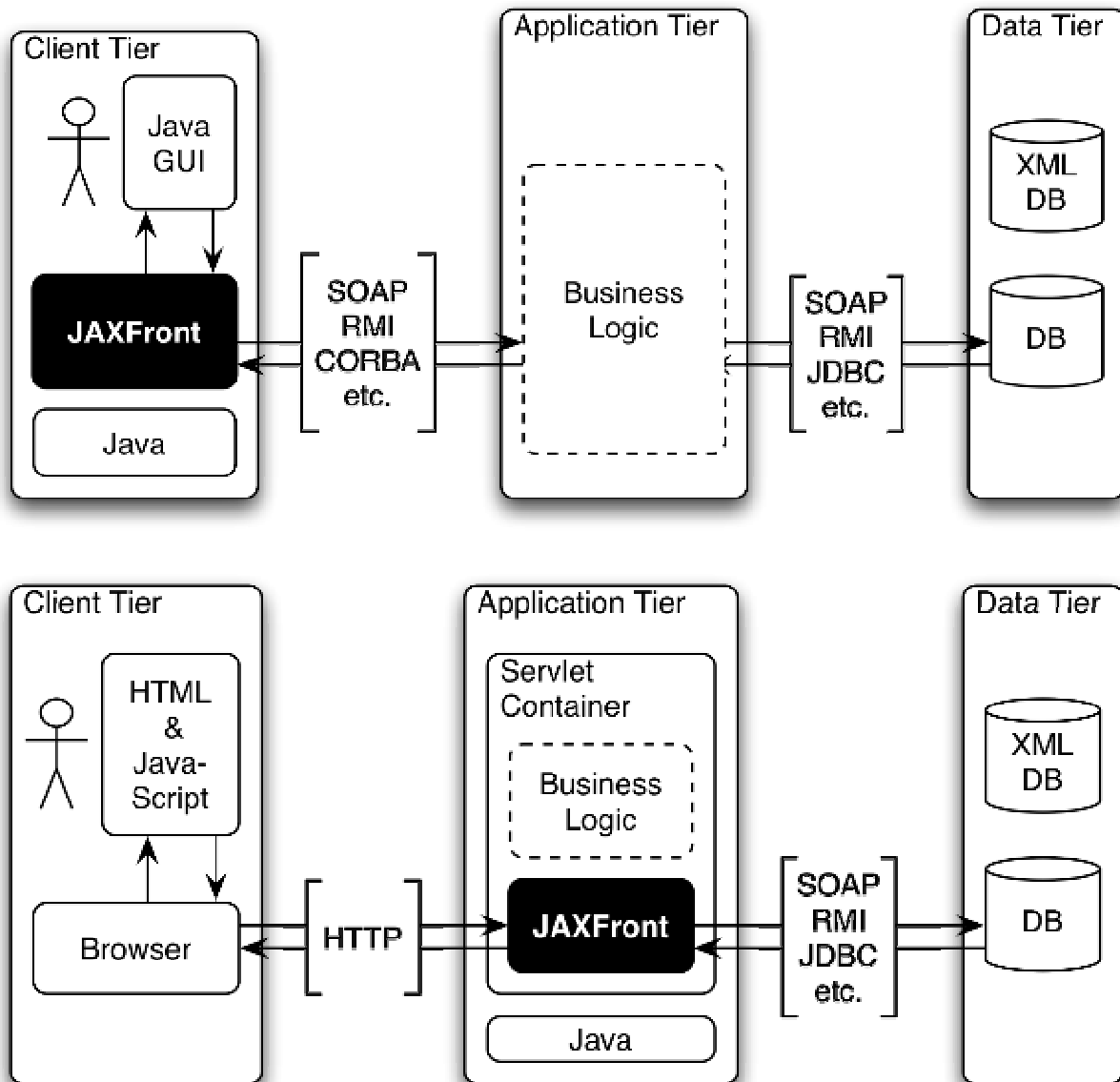
The HTML renderer creates dynamic HTML forms on-the-fly based on an XML Schema (required), a XML Instance and an XUI (both optional). The goal is to use the same XUI for a standalone JavaClient as well as a web based frontend (HTML) to reduce maintenance effort.





## 2.3 Java Client vs. HTML Frontend

Instead of having a client side Java application running on top of a local Java VM, the JAXFront HTML-Renderer is a servlet application and is therefore located in the servlet container of the server. On the client side a Browser (IE, Mozilla, Firefox with JavaScript) is sufficient.



When the user request a page a DOMHandler will be created on the server. The handler will then parse the xml files and build a DOM representation. Based on the structure of the DOM a HTML page will be created and sent back to the user. The user can then edit or create a new XML instance.

The JAXFront HTML-Renderer is highly dynamic and will adjust immediately to user input. Therefore input data is exchanged with the DOMHandler on the server side. This is done with asynchronous JavaScript (AJAX).

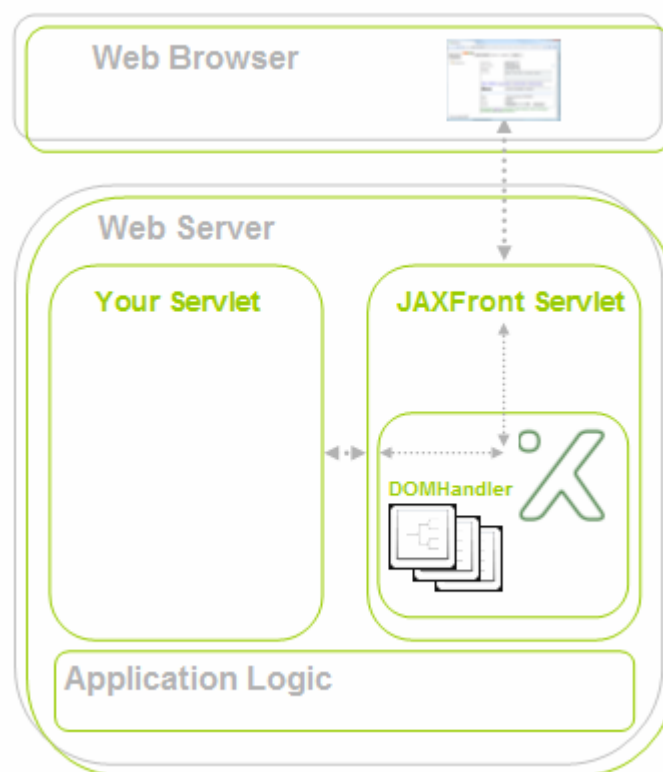
When the XML instance should be saved, the DOMHandler will serialize the DOM (holding the instance data) into an XML document. The XML can then be sent i.e. to a Webservice.

## 2.4 Using JAXFrontServlet vs. Embedding JAXFront

Using the JAXFront HTML Renderer, one has two different options. You may either use the JAXFrontServlet directly to generate web forms or you may embed JAXFront within your own web application and create the web forms through a Java API and return them as XHTML streams.

### 2.4.1 Option 1: Using JAXFrontServlet

All servlet calls will be handled by the JAXFrontServlet. The allocated JAXFront DOMHandlers will reside in the JAXFront Servlet memory scope.

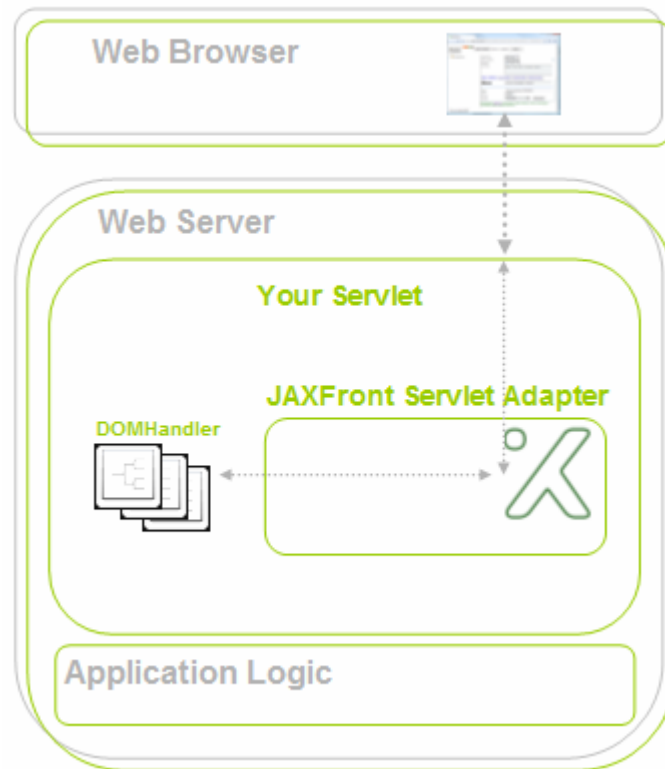


To generate web forms through the JAXFrontServlet, use the following URL:

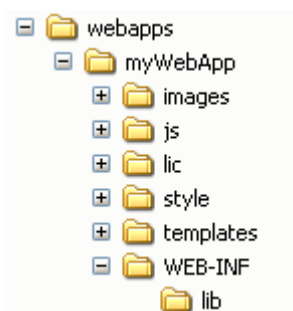
```
http://localhost:8080/jaxfront/JAXFrontServlet?app=jaxfront
&action=loadjaxfront
&xsdDoc=file:/C:/temp/po.xsd
&xmlDoc=file:/C:/temp/po.xml
&xuiDoc=file:/C:/temp/po.xui
&root=purchaseOrder
```

### 2.4.2 Option 2: Embedding JAXFront

Use the JAXFront ServletAdapter to dispatch all JAXFront relevant servlet requests within your own servlet. All requests are handled by “Your Servlet”. You have also full control and access to the instantiated JAXFront DOMHandlers. The created JAXFront DOM’s resist in the memory scope of your servlet.



To generate JAXFront web forms within your own servlet, copy the folders (\_images, js, lic, style, templates, WEB-INF/lib) from the jaxfront.war into your webapp directory (e.g. myWebApp) where your servlet is located:



In your servlet you need to dispatch all JAXFront relevant servlet calls to the JAXFrontServletAdapter. Do this in the service() method or in your doGet/doPost() methods.

Method: service()



```
protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException {

    //do your own stuff here...

    if (JAXFrontServletAdapter.doesConcern(request)) {
        JAXFrontServletAdapter.getInstance().handle(request, response);
    }

    //do your own stuff here...
}
```

Method: doGet(), doPost()



```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException {

    //do your own stuff here...

    if (JAXFrontServletAdapter.doesConcern(request)) {
        JAXFrontServletAdapter.getInstance().handle(request, response);
    }

    //do your own stuff here...
}

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException {

    //do your own stuff here...

    if (JAXFrontServletAdapter.doesConcern(request)) {
        JAXFrontServletAdapter.getInstance().handle(request, response);
    }

    //do your own stuff here...
}
```

Your servlet is now responsible to create JAXFront web forms. This means that you have to create a JAXFront DOM first, visualize & serialize it as an XHTML stream. There are two ways creating a JAXFront web form:

### 2.4.2.1 Simple integration

Use the short cut action name 'loadjaxfront' to generate a JAXFront web form based on the passed xsdDoc, xmlDoc and xuiDoc url parameters.

```
http://localhost:8080/myContext/MyServlet?app=jaxfront
&action=loadjaxfront
&xsdDoc=file:/C:/temp/po.xsd
&xmlDoc=file:/C:/temp/po.xml
&xuiDoc=file:/C:/temp/po.xui
```

### 2.4.2.2 Fully own API integration

If you would like to be in control of the DOM instance on your own because you want to load a specific xml instance or saving the DOM to an external persistency source (db, file system, ...), you need to create a JAXFront DOM (hold a reference to it), visualize and serialize it as an XHTML stream in your own servlet method.

The full example described as follow can be download from the JAXFront product home page:

To see an example demo servlet embedding JAXFront see the java class:  
***com.jaxfront.example.servlet.MyCustomerServlet***



Within your servlet service method, define a URL paramter which indicates the creation of a JAXFront web form (e.q. action=create).

```
public void service(HttpServletRequest request, HttpServletResponse
response) throws javax.servlet.ServletException, java.io.IOException {

    if (JAXFrontServletAdapter.doesConcern(request)) {
        JAXFrontServletAdapter.getInstance().handle(request, response);
    } else {
        String action = request.getParameter("action");
        if (action != null && action.equals("create")) {
            createDOM(request, response);
        } else {
            super.doPost(request, response);
        }
    }
}
```

Your servlet needs to do the following steps after it has been recognized a JAXFront web form request:

- Create a JAXFront DOMHandler based on an XML schema, XML instance and XUI
- Create an ActionController for controlling the DOM actions (save, validate, ...)
- Call the JAXFront HTML API to build the XHTML form stream



The following example method called createDOM() creates a new JAXFront web form based on the po.xsd, po.xml and po.xui. The URL param "id" is been used to identify a xml instance. In this simple example it just points to a file stored in the servlet context.

```
private void createDOM(HttpServletRequest request, HttpServletResponse
response) {
    StringBuffer formContentXHTML = new StringBuffer();
    String id = request.getParameter("id"); // use this id to resolve xsd or
an xml instance as an example
    String xsd = JAXFrontServletAdapter.getRealPath(getServletContext(),
"examples/purchaseOrder/po.xsd");
    String xml = id;
    String xui = JAXFrontServletAdapter.getRealPath(getServletContext(),
"examples/purchaseOrder/po.xui");
    String key = xsd + "_" + xml + "_" + xui;
    String cachedFormID = (String) request.getSession().getAttribute(key);
    if (cachedFormID != null) {
        formContentXHTML.append(API.getForm(request, cachedFormID));
    }
    else {
        DOMHandler domHandler = new DefaultDOMHandler(null, null, request, xsd,
xml, xui, null);
        DOMActionController actionController = new MyOwnDOMActionController();
        String formID = API.buildForm(formContentXHTML, request, domHandler,
actionController);
        request.getSession().setAttribute(key, formID);
    }
    try {
        response.getWriter().write(formContentXHTML.toString());
    } catch (IOException e) {
    }
}
```

To better understand the full html Java API class, see the online javadoc:

<http://www.jaxfront.com/javadoc/html-api/>

### 2.4.3 Implement own form actions

The DOMActionController (i.e. MyOwnDOMActionController class) let you interact with the generated web form. You may even define your own web form buttons to interact with the form (save, validate, print, ...).



To handle a save request on your own, just overwrite the method saveDOM().

```
public ResultObject saveDOM() {
    ResultObject result = new ResultObject();
    String category = ResultObject.OK;
    String statusMessage = null;
    String xmlIdentifier = null;
    if (getDOMHandler().getXMLIdentification().length > 0) {
        xmlIdentifier = getDOMHandler().getXMLIdentification()[0];
    }

    try {
        String xml = getDOM().serialize().toString(); //store this xml to your
        DB as an example
        statusMessage = "Document has been stored successfully";
    } catch (ValidationException e) {
        category = ResultObject.ERROR;
        statusMessage = "Failed to save. Reason: " + e.getLocalizedMessage();
    }

    setClientStatusBarMessage((HttpServletRequest)getDOM().getClientProperty("ht
    tp-request"), getDOM(),result, category, statusMessage, null);
    return result;
}
```



If you want to have your own buttons on top of your form, just overwrite the method getHTMLButtonControls(). If you do not overwrite, the button defined in the template "ControlFrame.html" will be taken.

```
public String getHTMLButtonControls(HttpServletRequest request) {
    //return an xhtml string defining your form buttons
}
```

To call a specific user action in your own ActionController, an action can be called wherever in your HTML web frontend by calling one of the following javascripts:

- a) without any return value (return null), performUserAction('actionName')
- b) with a return value (return anObject), performUserActionWithPopUp('actionName')

Option b) will open a dialog window with the content of the passed Object.

The following html snippets shows how to call an own action.

```
<button id= "button_ownAction1" name="test" type="button"
onclick="performUserActionWithResponseDIV(this,'ownAction1', null,
'DIV_CUSTOM_AREA',null);">
    <p> <b>own action 1</b> </p>
</button>
```



To interact with the button clicked on your web form, just implement the method `performUserAction()`.

```
public ResultObject performUserAction(HttpServletRequest request,
HttpServletRequest response, String actionName) {
    ResultObject result = new ResultObject();
    if ("ownAction1".equals(actionName)) {
        StringBuffer htmlResponse = new StringBuffer();
        htmlResponse.append("<br/>");
        htmlResponse.append("<b>Hello...");
        result.setObject(htmlResponse);
    } else if ("ownAction2".equals(actionName)) {
        StringBuffer htmlResponse = new StringBuffer();
        htmlResponse.append("<br/><br/><br/>");
        htmlResponse.append("<b>Hello JAXFront user...");
        result.setObject(htmlResponse);
    } else if ("forwardToJSPAction".equals(actionName)) {
        RequestDispatcher dispatcher =
request.getRequestDispatcher("/examples/purchaseOrder/hello.jsp");
        try {
            dispatcher.forward(request, response);
        } catch (Exception e) {}
        result.setDispatched();
    } else {
        result = super.performUserAction(request, response, actionName);
    }
    return result;
}
```

The action name (i.e. "ownAction1") gets passed to the current DOM action controller. The action controller can implement the action. Each action is identified by a unique action name.

Be sure that the correct mime type for the returned object has been registered before. To register a mime type for an own user action, call the following method on class "JAXFrontServlet":

```
JAXFrontServlet.registerMimeTypeMapping("actionName", "mimeType");
```



## 2.5 Lifecycle

If you come from developing servlets/JSPs then your first question will be: How do JAXFront sessions relate to HTTP sessions?

JAXFront DOM handlers are living in sessions, which are administrated inside the JAXFront runtime environment. The sessions are kept in parallel to HTTP sessions, i.e. HTTP sessions may be used by other servlets/JSPs that may be part of your web application – but JAXFront itself does not require them. It is no problem to reach HTTP sessions from a DOM handler object via an API.

Why is JAXFront not using straight HTTP sessions? Well, the ugly problem is that HTTP sessions are sometimes the same for multiple browser instances. If you, in Internet Explorer, open up a new browser from an existing one then the corresponding session object on the server is shared between the browsers. In the JAXFront session management, each instance of a browser has its own clearly assigned session.

### 2.5.1 Starting a session

The proper start of a JAXFront session is to open a JAXFront page via the following JAXFrontServer servlet: <http://localhost:8080/jaxfront/>... or by dispatching a servlet request to the JAXFrontServletAdapter.

### 2.5.2 Ending a session

A JAXFront session normally ends if the page which was opened with the JAXFrontServer servlet is closed. This happens for example if the user shuts down the browser or if the user is loading a new page into the frame in which previously the JAXFrontServer servlet was called.

If a JAXFront session is without user interaction for a long time, then the session is timed out on the server. Should the user come back to continue interaction then a corresponding message pops up. The default timeout is 20 minutes.

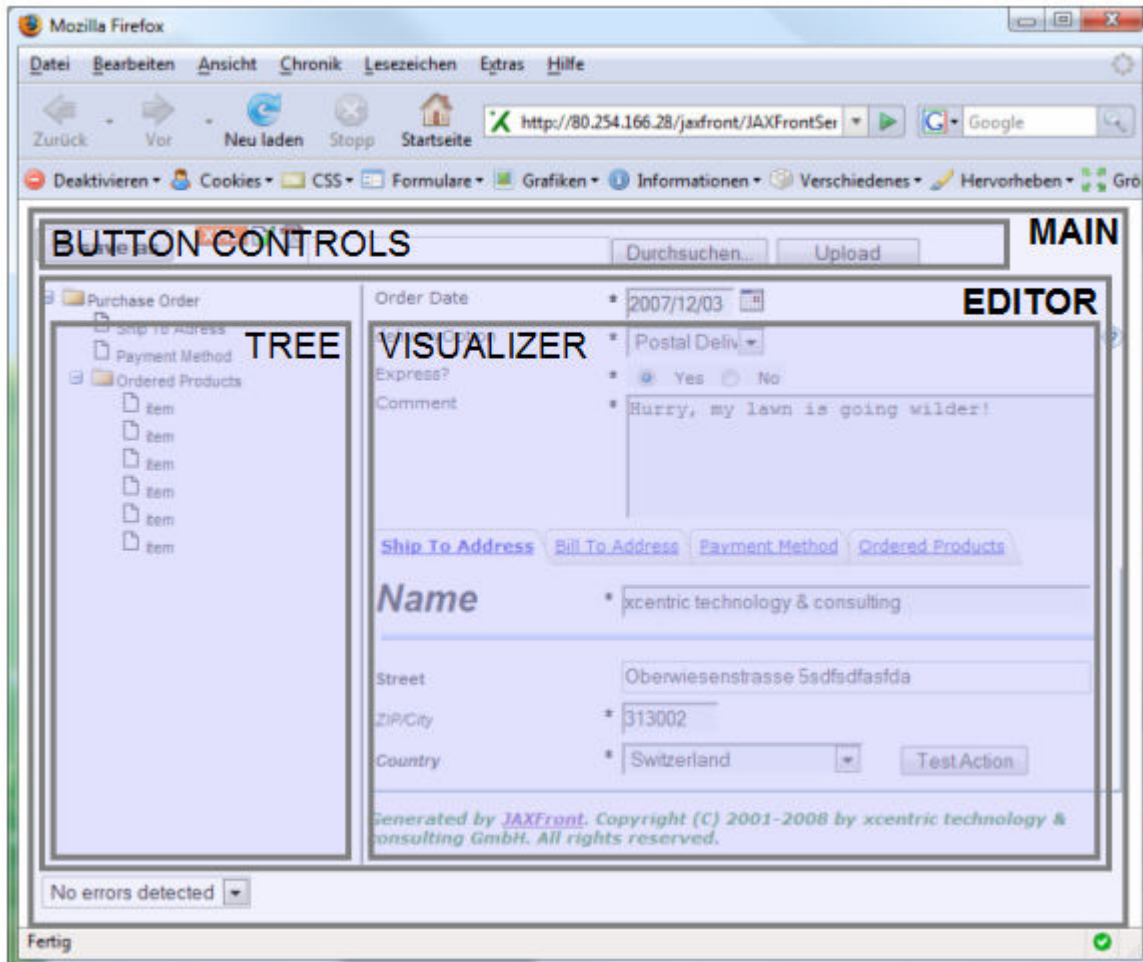
### 2.5.3 What is in a session?

Each JAXFront session holds a DOM handler object (DefaultDOMHandler) which initializes the underlying DOM based on the url params. Interactions with the DOM is done through this handler. If one needs to right some specific interaction code for loading, storing or releasing a DOM, see next caption.

## 2.6 General layout structure (div layers)

The basic structure of a JAXFront web frontend is based on the following layout.

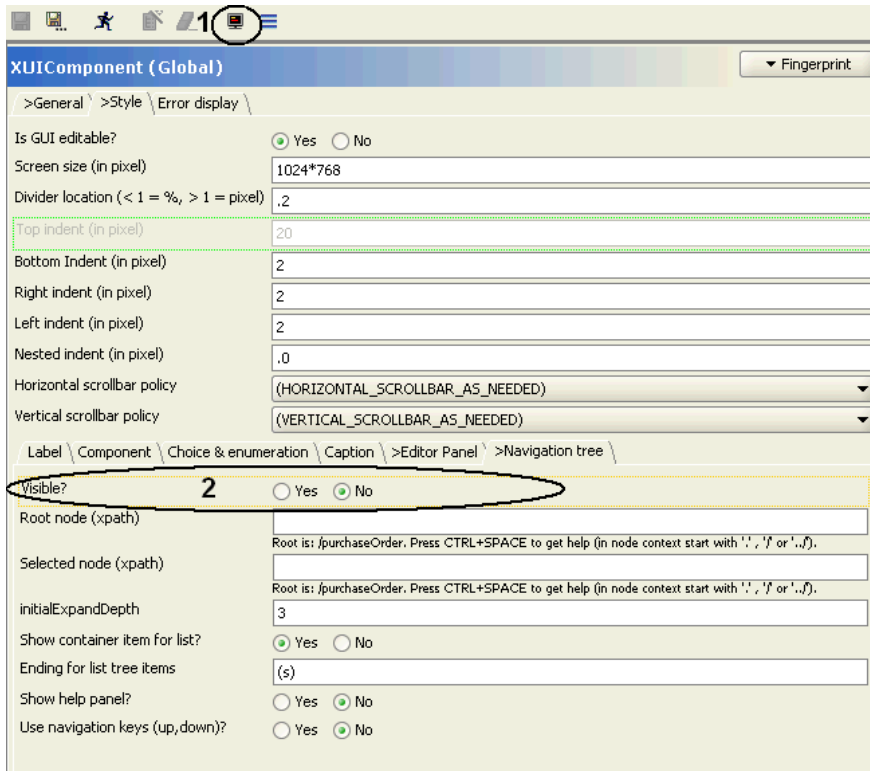
One may change the structure or the layout of the div layers by changing the content of the templates located in the JAXFront webapps folder called "templates". Anything except the reserved tag names `#{reservedTag}` may be changed.



If the tree (on the left side) should not be displayed, set the following global setting in your XUI:

```
<XUI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="xui.xsd"
version="1.60">
  <global>
    <style>
      <navigationTree>
        <visible>false</visible>
      </navigationTree>
    </style>
  </global>
</XUI>
```

In the XUI Editor, do the following:

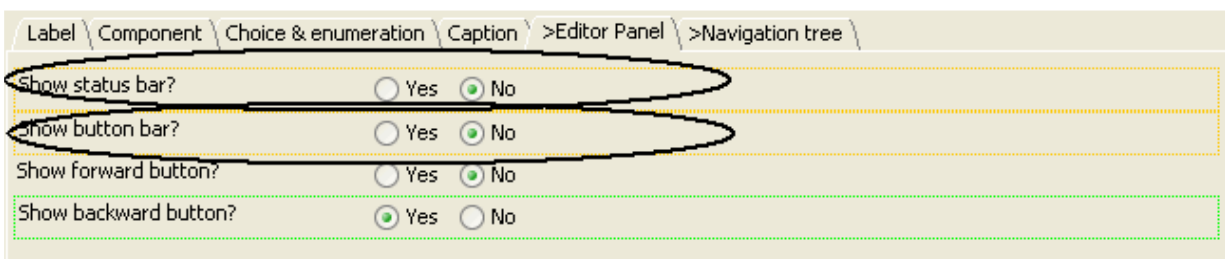


Click on global settings in the toolbar. Select tab “Style”, choose tab “Navigation tree” and set the field “Visible?” to “No”.



If you do not want the HelperFrameSet containing the error list and buttons, set the following global XUI setting:

```
<XUI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="xui.xsd"
version="1.60">
  <global>
    <style>
      <editorPanel>
        <showStatusBar>false</showStatusBar>
        <showButtonBar>false</showButtonBar>
      </editorPanel>
    </style>
  </global>
</XUI>
```



## 2.7 Implement your own Plugins

JAXFront allows you to overwrite any standard widget generated by the visualizer factory. Just specify an own plugin class in your XUI for a certain field/block.



As an example see the po.xui in the jaxfront-demo.war. There is a HTML plugin defined for the xpath: /purchaseOrder/shipTo/street:

```
<component xpath="/purchaseOrder/shipTo/street">
  <style>
<plugIn
class="com.jaxfront.html.plugins.SimpleTypePluginPlainHTMLExampleView">
  </plugIn>
  </style>
</component>
```



The following simple JAXFront HTML plugin just creates a listbox containing three different street names to choose from. As soon as the user selects a street, the value will be updated asynchronously in the server-side existing JAXFront DOM.

```
public class SimpleTypePluginPlainHTMLExampleView extends SimpleTypeView {

protected void createEditorComponent(HtmlContainerWidget container) {
  _component = new HtmlPlainTextWidget(container,
getHTMLContent().toString());
}

public void populateView() {
  super.populateView();

((HtmlPlainTextWidget)_component).setHTMLContent(getHTMLContent().toString()
);
}

public StringBuffer getHTMLContent() {
  StringBuffer sb = new StringBuffer();
  sb.append("<b>Choose your favorite street!</b><br/><br/>");
  sb.append("<select id=\"" + getXPath() + "\" size=\"3\" "
onclick=\"saveData(this)\">");
  String[] names = new String[] { "Nowhere Land", "Palm Street", "Example
Street", "Wall Street", "Bahnhof Street" };
  String selected = "";
  for (int i = 0; i < names.length; i++) {
    if (getValue() != null && getValue().equals(names[i]))
      selected = "selected";
    else
      selected = "";

sb.append("<option " + selected + " value=\"" + names[i] + "\"> " +
names[i] + "</option>");
  }
  sb.append("</select>");
  return sb;
}

public void setSize(String size) {
  // do nothing
}
```

### **3 Limitation**

If you wish to extend JAXFront or want to have new features implemented, do not hesitate to contact us ([info@jaxfront.com](mailto:info@jaxfront.com)). A technical representative will contact you.

Please notify the following limitations.

- Substitution Groups are not supported yet.
- The identity constraint 'unique' is not supported yet.
- The derivedBy 'Union' is not supported yet.
- The use of namespaces may lead to some problems concerning the use of global XUI definitions.
- Be aware that the size of your XML schema in combination with the size of the XML Instance may lead to Out-Of-Memory problems within your JavaVM if you do not provide sufficient memory allocation.