Developer's Guide

RISC add-ons

Inhaltsverzeichnis

Adding own JavaScript Files....4
Where to place and how to register own .js files....4
Add .js files to /webcontent/eclnt/risc/plugin....4
Registering own JavaScript files inside system.xml....4
Rules / Conventions....4
Overriding / Influencing the RISC Client by own JavaScript....5
Multi Language / Internationalization....5
Predefined exits....5
All Exits – View /webcontentcc/eclnt/risc/plugin/plugin.js_template....5
Exit - Providing an own Error Page Layout....5
Adding own Style Sheet Files (e.g. Fonts)....8
What internally happens....8
Adding own CSS files....9
Adding CSS files via system.xml....9
Placing CSS files into the style directory....9
webcontent, webcontentbuild, webcontentcc....9
Embedding a CaptainCasa dialog into a non-CaptainCasa page....10
Scenario / Example....10
Concepts....10
The outside page....11
The CaptainCasa page....11
Simple String Protocol....13
Using a own version of CKEditor....14
Integration into CaptainCasa....14
Default scenario....14
Own version....14
What to do....14
What NOT to do....15
Embedding CaptainCasa dialogs into SSO (single sign on) scenarios....16
In general – Use “COOKIE” session tracking....16
Accessing http information from your application....16
Special topic: Calling CaptainCasa page and passing “Authorization” http header value...16
Example....16
Recording http traffic so that CaptainCasa can replay....19
Security Issues....19
Configuration....19
Recording....20
Step 1 – Switch the recording to “on”....20
Step 2 – Clear your browser cache....22
Step 3 – Run the URL of your application....23
Step 4 – Switch the recording to “off”....24
Step 5 – Send the recorded information to CaptainCasa....24
...you want to replay on your own?....24
Transferring Enterprise Client from Swing/FX to RISC....25
The way to go....25
Install the latest version....25
Upgrade your project....25
Update your web.xml....25
Reload/Redeploy your application....25
Start your page....25
Incompatibilities....25
BGPAINT Attribute....25
FILEUPLOAD* components....26

Removing all “non-RISC”-parts.....26
Files to be removed from your project.....26
Cleaning up your Installation.....26
After re-install: re-deploy your projects.....27

Adding own JavaScript Files

Where to place and how to register own .js files

When calling a RISC-page (e.g. "http://xyz/xyz/abc.risc?ccstyle=defaultrisc") then internally an HTML page is generated at runtime that contains the corresponding HTML and JavaScript to start the page. Inside this generated HTML page the JavaScript-files (*.js) are listed that are part of the page processing.

By default all JavaScript files are included that form the RISC-client. But you may define own ".js" files on your own and add them to the list. The purpose of own files is to either extend or sometimes override the default JavaScript that comes with the RISC-client.

Add .js files to /webcontent/eclnt/risc/plugin

Any ".js"-file that you add into this directory will automatically be loaded. There is no additional registration required.

Registering own JavaScript files inside system.xml

You may add own JavaScript files by updating the system.xml configuration file, located in "/webcontent/eclntjsfserver/config/system.xml".

Example:

```
<system>

...
...

    <riscclientscript src="xxx/yyy/zzzz.js" type="text/javascript" />
    <riscclientscript src="aaa/bbb/ccc.js" type="text/javascript"/>

...
...

</system>
```

The *.js files that you add must reside within the webcontent directory of your project:

```
<project>
  webcontent
    aaa
      bbb
        ccc.js
    xxx
      yyy
        zzzz.js
```

Rules / Conventions

CaptainCasa uses two the namespaces for all its variables / functions that are defined on window-level ("global level"):

- "RISC*" for all RISC related issues – this is the rendering of components
- "CC*" for all CaptainCasa related issues – this is the binding level between the rendering and the server side processing

Do not use this namespace on global level!

Overriding / Influencing the RISC Client by own JavaScript

The JavaScript sources that you add are loaded after the loading of the default RISC JavaScript files. This means: you may override certain behavior of the RISC client.

While JavaScript itself is 100% open and allows you to do “everything” we strongly recommend to only extend the behavior of the RISC client where we explicitly provide possibilities for extension!

Multi Language / Internationalization

(Up to version 20190107 you had to add own, new countries by adding JavaScript code to the client code. From 20190107 on the configuration of client countries and languages is done on server side by adding XML files into the directory webcontent/eclnt/risc/i18n. Please check the Developer's Guide for more details.)

Predefined exits

There are predefined exits within the default RISC-client. An exit is an API-call that the RISC-client executes to some outside JavaScript-code. If the outside code exists then it is called, if not then a default behavior is executed.

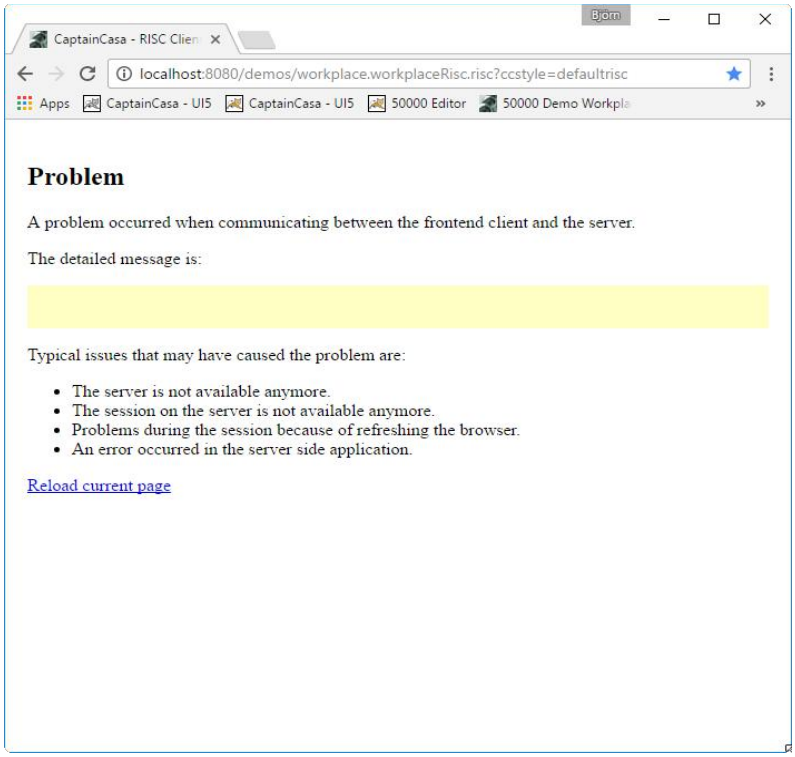
All Exits – View /webcontentcc/eclnt/risc/plugin/plugin.js_template

The file “plugin.js_template” is the one that lists all the exits that are provided by the RISC-client. For implementing the exits on your own you may just copy the “plugin.js_template” file as “/webcontent/eclnt/risc/plugin/plugin.jsp” into you project and then implement these exits that you want to implement. You do not have to implement all exits, so you may comment out or remove the corresponding JavaScript-methods.

The number of exits is quite low at the moment (Sep 2017) – but will be increased step by step.

Exit - Providing an own Error Page Layout

In case the client runs into some error (e.g. the server is not reachable), it shows some error page:



You may customize this layout by overriding the following method:

```
/**
 * Returns the HTML that is output if an error occurs during communication.
 */
CCPlugin.createClientErrorPageHTML = function(pReloadUrl,pMessage)
{
    var sb = new Array();
    sb.push("<div style='font-family:Open
Sans;width:100%;height:100%;padding:20px;background-color:#FFFFFF;-moz-box-sizing:
border-box;-webkit-box-sizing: border-box;box-sizing: border-
box;overflow:auto'>");
    sb.push("<p style='font-family:Open Sans;font-size:25px; font-
weight:bold'>Problem</p>");
    sb.push("<p style='font-family:Open Sans;font-size:12px'>");
    sb.push("A problem occurred when communicating between the client and the
server.");
    sb.push("</p>");
    if (pMessage != null && pMessage != "")
    {
    sb.push("<hr>");
    sb.push("<p style='font-family:Open Sans;font-size:12px'>");
    sb.push("Details:");
    sb.push("</p>");
    sb.push("<p>");
    sb.push("<div style='width=100%;background-color:#F0F0F0;padding:10;font-
family:Courier New;font-size:11px;border:0px #c0c0c0 solid'>"+pMessage+"</div>");
    sb.push("</p>");
    }
    sb.push("<hr>");
    sb.push("<p style='font-family:Open Sans;font-size:12px'>");
    sb.push("Typical issues that may have caused the problem are:");
    sb.push("<ul>");
    sb.push("<li style='font-family:Open Sans;font-size:12px'>The server is not
available anymore.</li>");
    sb.push("<li style='font-family:Open Sans;font-size:12px'>The session on the
server is not available anymore.</li>");
    sb.push("<li style='font-family:Open Sans;font-size:12px'>An error occurred in
the server side application.</li>");
    sb.push("</ul>");
    sb.push("</p>");
    sb.push("<hr>");
    sb.push("<p>");
    sb.push("<a
href='javascript:RISCConfirmExit.avoidBlocking();location.reload();' style='font-
size:12px; font-weight:bold'>Reload current page</a>");
    sb.push("</p>");
```

```
    sb.push("</div>");
    return sb.join("\n");
};
```

Adding own Style Sheet Files (e.g. Fonts)

Sometimes you want to register own style sheet files in the CaptainCasa page processing. Typical example: you want to add a font to CaptainCasa, that is not loaded by default.

What internally happens

When placing a ".risc"-URL against the CaptainCasa server then a servlet is invoked that generates the HTML page, which then itself starts the JavaScript client as content.

The HTML page looks the following way:

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="viewport" content="width=device-width,height=device-height,user-
scalable=no" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <link rel="stylesheet" href="eclnt/risc/fonts/opensans/css/OpenSans.css">
    <link rel="stylesheet" href="eclnt/risc/fonts/awesomefont/css/font-
awesome.min.css">
    <link rel="stylesheet" href="eclnt/risc/fonts/sapicons/css/SAPIcons.css">
    <title>CaptainCasa Enterprise Client - RISC-HTML</title>
    <link rel="icon" href="eclnt/images/icon.png" />
<!-- ********** STYLES ************************************************** -->

<link href="eclntjsfserver/styles/defaultlightbluerisc/riscstyle.css"
rel="stylesheet"/>

<!-- ********** JAVASCRIPT ********************************************** -->
<script type="text/javascript" src="eclnt/risc/eclntriscbasics.js"></script>

<script type="text/javascript"
src="eclntjsfserver/styles/defaultlightbluerisc/riscstyle.js"></script>
<script type="text/javascript" src="eclnt/risc/risc.js"></script>

<script type="text/javascript" src="ztest/zzzzz.js"></script>

<!-- ********** BOOTSTRAP JAVSCRIPT ************************************* -->
<script>
var s_pageBrowser = null;

function initializeApplication()
{
    CLog.logINF("initializeApplication() called");
    var vUrl = document.URL;
    var vIndex1 = vUrl.indexOf("://") + 3;
    var vIndex2 = vUrl.indexOf("/",vIndex1);
    var vPrefix = vUrl.substring(0,vIndex2);
    s_pageBrowser = new
CCPageBrowser(vPrefix,"/demos/faces/workplace/workplaceRisc.jsp;jsessionid=950EF8B
923889D176CD23847F8D0CDBE?ccstyle=defaultlightbluerisc");
    return s_pageBrowser.getUI5Node();
}
</script>
</head>

<body class="riscbody" onload="RISCPage.initialize(1,false);"
onbeforeunload="return RISCConfirmExit.confirmExit(event);"
onunload="RISCWindowMgr.onunload(event);">
<span><font face="Open Sans" size="100"> </font></span>
<span><b><font face="Open Sans" size="100" > </font></b></span>
</body>
</html>
```

You see, that in the "<head>...</head>" section style sheets are contained – e.g. there are three fonts definitions that are contained, including the OpenSans-font (this is the font addressed by CaptainCasa default style) and including the "awesomefont"-font and the "SAP-icon"-font.

Adding own CSS files

Adding CSS files via system.xml

In the configuration file “eclntjsfserver/config/system.xml” there is a section which allows to directly define the CSS-filese to be included:

```
<system>
    ...
    <!--
      ***************************************************************************
      Configuration of additional styles includes in RISC client
      ***************************************************************************
     -->
    <riscclientstyle src="xxx/yyy/zzzz.css"/>
    <riscclientstyle src="aaa/bbb/ccc.css"/>
    ...
</system>
```

The files need to be part of your web content directory.

Placing CSS files into the style directory

Each style is represented by one directory within “<webcontent>/eclntjsfserver/styles”. Any css-file that is placed here is automatically available on client side.

Please note: only the CSS files of the current style are loaded! If e.g. style “xxx” extends style “yyy” then only the directory for style “xxx” is scanned. There is no cascading scanning.

webcontent, webcontentbuild, webcontentcc...

A typical CaptainCasa project is separated into three webcontent-directories.

- webcontent – this is the directory that contains your artifacts
- webcontentbuild – this is the directory that contains all compiled files
- webcontentcc – this is the directory contains all CaptainCasa addons to you web application

All three directories are copied and merged at runtime (e.g. they are copied into one tomcat/webapps/<app> directory).

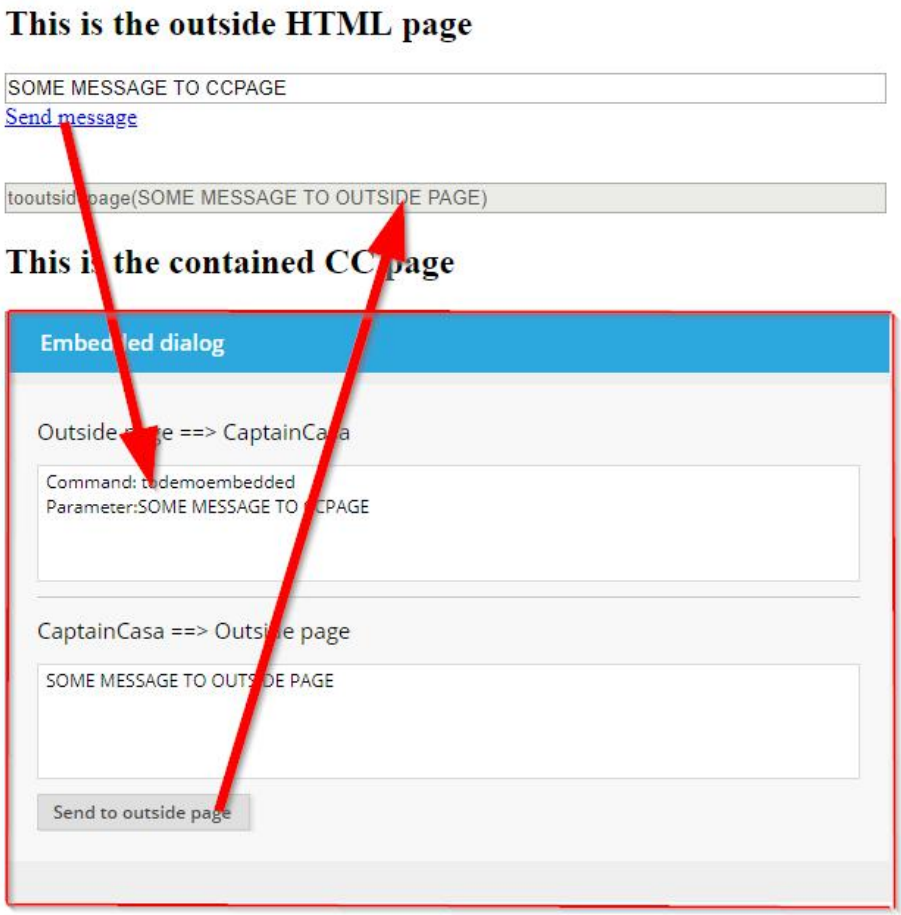
The adequate directory for adding your artifacts always is the “webcontent” directory! So, if e.g. using the default style “defaultlightbluerisc” of CaptainCasa, do NOT add your own .css-files to the webcontentcc-directory (where the original CaptainCasa style is kept), but add your own .css-files to:

```
<project>
  webcontent
    eclntjsfserver
      styles
        defaultlightbluerisc
          own1.css
          own2.css
  webcontentbuild
  webcontentcc
```

Embedding a CaptainCasa dialog into a non-CaptainCasa page

Scenario / Example

Take a look onto the following page:



There is some outside HTML page, that internally embeds a CaptainCasa page. And there is the possibility to interact in both directions.

The outside page is **not** loaded from the same domain as the inside page.

Concepts

In principal pages between different domains can only talk to one another by using message-events. There is no possibility to directly call JavaScript functions due to security restrictions (cross site scripting).

Inside CaptainCasa dialogs there is a so called “Message Bus”. This message bus is a central instance within the client to which you can send messages, and from which you can receive messages.

- Each message is a string in the format “<messageName>(<param>,<param>,...)”
- Listening and receiving is done by using an invisible component MESSAGELISTENER. In this component you register for the message names that you are interested in. Once a message is received then a corresponding ACTIONLISTENER is called no server side. The

message is part of the event data.

- Vice versa a CaptainCasa dialog can send messaged by using Java-API “AsynchMessageBus”. Messages are normally only processed within the CaptainCasa dialog, but you can use API “AsynchMessageBus.addMessage(String message, boolean withParentDelegation)” and can define that messages are also delegated as browser-message-event to the parent of the current dialog.

The outside page

This is the HTML of the outside page:

```
<html>

<script>
function sendMessage()
{
    var f = frames.IFRAME1;
    var t = document.getElementById("SEND").value;
    var m = "messagebus:todemoembedded("+t+")";
    f.postMessage(m,"*");
}
function _receiveMessageEvent(e)
{
    if (e == null) e = window.event;
    var s = e.data;
    if (s != null && s.indexOf("messagebus:tooutsidepage(") == 0)
    {
        s = s.substring(11);
        document.getElementById("RECEIVED").value = s;
    }
}
window.addEventListener("message",function(e) {_receiveMessageEvent(e);});
</script>

<h2>This is the outside HTML page</h2>
<input id="SEND" value="Harry"  style="width:600px" >
<br>
<a href="javascript:sendMessage();">Send message</a>
<br>
<br>
<br>
<input id="RECEIVED" value="(Nothing received yet.)" style="width:600px"
disabled="true">
<br>
<h2>This is the contained CC page</h2>
</html><iframe id="IFRAME1" name="IFRAME1"
src="http://localhost:8080/demos/workplace.demoembeddeddialog.risc" width="600"
height="400"></iframe>
<br>
```

You see:

- The CaptainCasa page is included as IFRAME.
- Messages are sent through the sendMessage()-function – creating a message with the name “messagebus:” and then the text of the message. In this case the message name is “todemoembedded”.
- The page registers for messages (window.addEventListener) and receives the message in a corresponding function (_receiveMessageEvent).

The CaptainCasa page

The inner page is a quite simple one:

```
<t:beanprocessing id="g_1">
      <t:messagelistener id="g_2"
            actionListener="#{d.DemoEmbeddedDialog.onMessageReceived}"
            commandfilter="todemoembedded" />
</t:beanprocessing>
```

```
<t:rowtitlebar id="g_3" text="Embedded dialog" />
<t:rowheader id="g_4" />
<t:rowbodypane id="g_5" rowdistance="10">
	<t:row id="g_6">
		<t:label id="g_7" font="size:15"
			text="Outside page ==&gt; CaptainCasa" />
	</t:row>
	<t:row id="g_8">
		<t:textarea id="g_9" height="100%"
			text="#{d.DemoEmbeddedDialog.messageReceived}" width="100%" />
	</t:row>
	<t:rowline id="g_10" />
	<t:row id="g_11">
		<t:label id="g_12" font="size:15"
			text="CaptainCasa ==&gt; Outside page" />
	</t:row>
	<t:row id="g_13">
		<t:textarea id="g_14" height="100%"
			text="#{d.DemoEmbeddedDialog.messageToSend}" width="100%" />
	</t:row>
	<t:row id="g_15">
		<t:button id="g_16"
			actionListener="#{d.DemoEmbeddedDialog.onSendButtonAction}"
			text="Send to outside page" width="100+" />
	</t:row>
</t:rowbodypane>
<t:rowstatusbar id="g_17" />
```

The Java code is:

```
public class DemoEmbeddedDialog
    extends PageBean
    implements Serializable
{
    // ------------------------------------------------------------------------
    // members
    // ------------------------------------------------------------------------

    String m_messageReceived = "(Nothing received yet.)";
    String m_messageToSend = "Some message to the client.";

    // ------------------------------------------------------------------------
    // constructors & initialization
    // ------------------------------------------------------------------------

    public String getPageName() { return "/workplace/demoembeddeddialog.jsp"; }
    public String getRootExpressionUsedInPage() { return
"#{d.DemoEmbeddedDialog}"; }

    // ------------------------------------------------------------------------
    // public usage
    // ------------------------------------------------------------------------

    public String getMessageToSend() { return m_messageToSend; }
    public void setMessageToSend(String value) { this.m_messageToSend = value; }

    public String getMessageReceived() { return m_messageReceived; }
    public void setMessageReceived(String value) { this.m_messageReceived = value;
}

    public void onSendButtonAction(javax.faces.event.ActionEvent event)
    {
        AsynchMessageBus.addMessage("tooutsidepage("+m_messageToSend+")",true);
    }

    public void onMessageReceived(javax.faces.event.ActionEvent event)
    {
        if (event instanceof BaseActionEventMessage)
        {
            BaseActionEventMessage e = (BaseActionEventMessage)event;
            StringBuffer sb = new StringBuffer();
            sb.append("Command: " + e.getMessageCommand() + "\n");
            String[] params = e.getMessageParameters();
            for (String param: params)
            {
                sb.append("Parameter:" + param + "\n");
            }
            m_messageReceived = sb.toString();
        }
```

```
    }
}
```

You see:

- In the page there is a MESSAGELISTENER component listening to message name “todemoembedded”. It is bound to metho “onMessageReceived”.
- In the code there is a calling of “AsynchMessageBus.addMessage(...)” if the “Send to outside page” button is pressed by the user.

Simple String Protocol

The protocol into both directions is completely up to you. The basic structure is:

```
nameOfMessage(param1,param2,param3,...)
```

Do not use “strange characters” within the message parameters. If you want to transfer e.g. complex XML data or JSON strings, then e.g. use Base64 encoding to transfer all content into simple characters before passing it as parameter of the message.

Using a own version of CKEditor

First of all: CKEditor is a 3rd party product – please carefully read the corresponding license and usage terms from https://ckeditor.com!

Integration into CaptainCasa

The integration is done using the SIMPLEHTHMLEDITOR component. This component internally opens up an IFRAME in which CKEditor is started. The communication is done through a bridging page which receives JavaScript messages from the component and transfers them into corresponding JavaScript calls for CKEditor.

Default scenario

CaptainCasa comes with the following default scenario: the CKEditor is loaded from the following directory:

```
<project>/webcontentcc/eclnt/risc/ext_ckeditor/v492
```

"v492" is the version of CKEditor – which may be different from release to release. CaptainCasa by default only provides the "Basic Package" of CKEditor, there are multiple more packages available – with a different and a configure-able set of features.

You may download these packages from the CKEditor web site. The current download address for CKEditor is "https://ckeditor.com/ckeditor-4/download/".

There is one additional file in this directory – the bridging page between the CaptainCasa processing and the CKEditor processing. The name of this file is:

```
ext_ckeditor.html
```

In the SIMPLEHTMLEDITOR component you can explicitly specify the bridging page by suing attribute BRDIGEPAGE, the default currently is "eclnt/risc/ext_ckeditor/v492/ext_ckeditor.html".

Own version

What to do

The best way to install own versions of CKEditor is:

- Download the corresponding version from CKEditor
- Place the version in your webcontent – we recommend to use a similar directory location as CaptainCasa does.
- Copy the bridging page "ext_ckeditor.html" from the CaptainCasa default directory (webcontentcc) directory into the directory:

Example:

```
<project>
  webcontent
    eclnt
      ext_ckeditor
        v492_OwnVersion
          …
          … <== all CKEditor files
          …
          ext_ckeditor.html
  webcontentbuild
  webcontentcc
```

(If using Maven-style projects than the directory structure looks different. The “webcontent” directory then is “source/main/webapp”.)

In the dialog definition (.jsp) in which you use the SIMPLEHTHMLEDITOR component assign the value “eclnt/ext_ckeditor/v492_OwnVersion/ext_ckeditor.html” to the attirbute BRIDGEPAGE.

What NOT to do

- Do not start the attribute value of BRIDGEPAGE with a leading “/”!
- Do not copy your version of CKEditor into the “webcontentcc”-directory! This directory is owned and managed by CaptainCasa.

Embedding CaptainCasa dialogs into SSO (single sign on) scenarios

In general – Use “COOKIE” session tracking

CaptainCasa knows two session tracking modes:

- “URL” (URL encoding)
- “COOKIE” (Cookie usage)

The default is “URL” - but you may change to “COOKIE” easily. Please read the chapter “Http Session Managament” within the Develoer's Guide for more information.

SSO frameworks typically transfer information based on cookies, or they store information based on the JSESSIONID-cookie-value. So in genreal use “COOKIE” based session tracking within CaptainCasa.

Accessing http information from your application

There is quite some variance of frameworks in the SSO area. Typically the user is routed to some login page of the SSO framework automatically when first time accessing the application. After successful logon the browser is routed back to the application page that was requested.

After logon SSO information typically is available by accessing information that is part of the http-request of the http-session. Use the CaptainCasa facade “HttpSessionAccess” in order to access these objects:

```
HttpSessionAccess.getCurrentRequest()
HttpSessionAccess.getCurrentHttpSession();
```

Example: from the request you may access the user-principal information, that is some standarf for storing login information.

Please note: you need to reference the servlet-API library within the classpath of your project for accessing the http-objects. For Maven-style projects it is already integrated, for default projects you may embed the tomcat/libs/servlet-api.jar.

Special topic: Calling CaptainCasa page and passing “Authorization” http header value

In single sign on (SSO) scenarios you may call a CaptainCasa page e.g. from some starter page – and you may pass an authorization token (e.g. Bearer-token) to be used with every http request that is called from the CaptainCasa dialog.

You do so by setting a cookie “CCHeaderField_Authorization” and passing the full authoritation info (type + token).

Example

Example: you may define an “index.html” within your webcontent:

```
<html>
<script src="keycloak/keycloak.js"></script>
<script>
var keycloak = new Keycloak('keycloak/keycloak.json');
keycloak.init
(
```

```
   {
       onLoad: 'login-required'
   }
).success
(
    function(authenticated)
    {
        keycloak.loadUserProfile().success
        (
            function(profile)
            {
                CCDataTransfer_BearerToken = keycloak.token;
                document.cookie =
                    "CCHeaderField_Authorization=Bearer"+keycloak.token;
                startPage();
            }
        );
    }
);
function startPage()
{
    document.location.href = "test.risc?ccstyle=defaultlightbluerisc";
}
</script>
<body>
</body>
</html>
```

On the receiving side you may then access the http header parameter in the following way:

```
package managedbeans;

import java.io.Serializable;

import javax.servlet.http.HttpServletRequest;

import org.eclnt.editor.annotations.CCGenClass;
import org.eclnt.jsfserver.defaultscreens.Statusbar;
import org.eclnt.jsfserver.pagebean.PageBean;
import org.eclnt.jsfserver.util.HttpSessionAccess;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;

@CCGenClass (expressionBase="#{d.TestUI}")

public class TestUI
    extends PageBean
    implements Serializable
{
    String m_info = "";

    public TestUI()
    {
        collectAuthInfo();
    }

    public String getPageName() { return "/test.jsp"; }
    public String getRootExpressionUsedInPage() { return "#{d.TestUI}"; }


    private void collectAuthInfo()
    {
        m_info = "";
        HttpServletRequest req = HttpSessionAccess.getCurrentRequest();
        String authorization = req.getHeader("Authorization");
        m_info += "AUTHORIZATION:\n" + authorization + "\n\n";
        if (authorization != null && authorization.startsWith("Bearer"))
        {
            String token = authorization.substring("Bearer".length());
            m_info += "BEARER TOKEN:\n" + token + "\n\n";
        }
    }

    public String getInfo() { return m_info; }

}
```

In the example some bearer-token is passed from the client to the server. You may now use some JWT decoder to interpret the data contained in the token.

Recording http traffic so that CaptainCasa can replay

The CaptainCasa server environment comes with a special feature that is inevitably important for solving complex bugs within the CaptainCasa frontend processing – that (as usual...) only happen inside your own environment and cannot be reproduced at CaptainCasa side.

There is a certain recording of all http traffic that is exchanged between the browser and the server. The recorded files can then be sent to CaptainCasa so that CaptainCasa can replay the frontend – the requests are responded from the files that you recorder. The result: CaptainCasa can exactly replay your scenario.

Security Issues

The recorded information contains all the traffic between your browser and your server. If the content contains critical information (including passwords, business information, …) then this is logged as consequence – and readable as clear text. So only use the recording feature in demo systems with demo user accounts...! If in doubt about security, please contact CaptainCasa.

Configuration

Recording (and replaying) is done by a servlet filter which is part of the normal CaptainCasa server environment.

Register the filter in the web.xml of your application as follows:

```
  <filter>
    <filter-name>org.eclnt.jsfserver.util.CompressionFilter</filter-name>
    <filter-class>org.eclnt.jsfserver.util.CompressionFilter</filter-class>
  </filter>
  <filter>
    <filter-name>org.eclnt.jsfserver.util.ResponseLoggerFilter</filter-name>
    <filter-class>org.eclnt.jsfserver.util.ResponseLoggerFilter</filter-class>
  </filter>
  <filter>
    <filter-name>org.eclnt.jsfserver.util.SecurityFilter</filter-name>
    <filter-class>org.eclnt.jsfserver.util.SecurityFilter</filter-class>
  </filter>
  <filter>
    <filter-name>org.eclnt.jsfserver.util.ThreadingFilter</filter-name>
    <filter-class>org.eclnt.jsfserver.util.ThreadingFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>org.eclnt.jsfserver.util.CompressionFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>org.eclnt.jsfserver.util.CompressionFilter</filter-name>
    <url-pattern>*.xml</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>org.eclnt.jsfserver.util.ResponseLoggerFilter</filter-name>
    <url-pattern>*</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>org.eclnt.jsfserver.util.SecurityFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
  </filter-mapping>0
    <filter-mapping>
    <filter-name>org.eclnt.jsfserver.util.ThreadingFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
  </filter-mapping>
```

Your filter configuration might look a little bit different (e.g. you might not have the security filter installed). Please install the filter “behind” the CompressionFilter and “in front of” the SecurityFilter and ThreadingFilter. Please note: the sequence of filters is defined by the sequence of filter-mapping-definitions – and NOT by the sequence of filter-definitions.

Please also pay attention: the filter MUST NOT be active in production scenarios! It is only to be used in development scenarios. (Otherwise any traffic is recorded!)

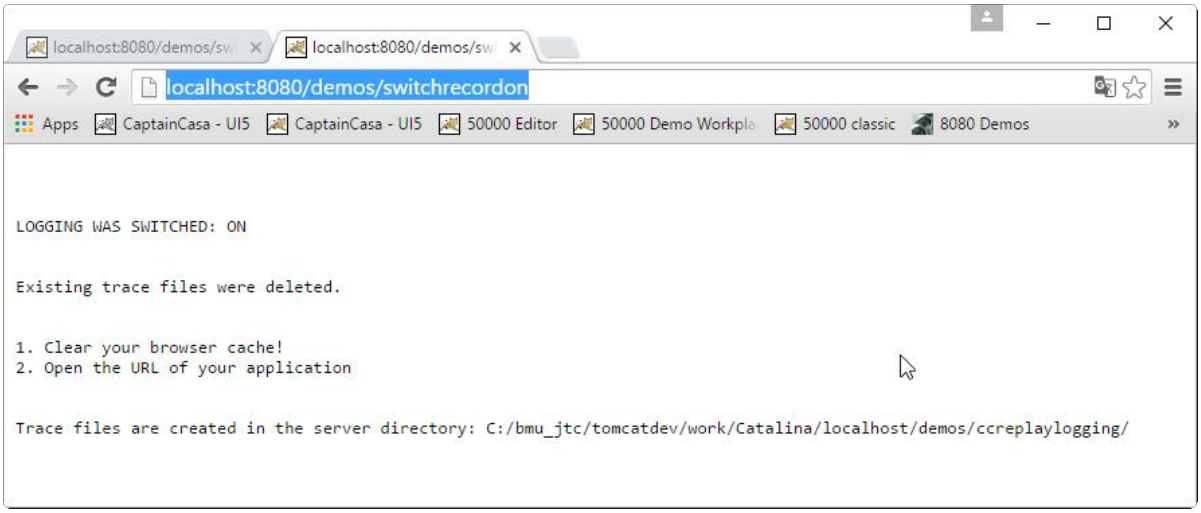
(Actually we added some protection mechanism here...: the filter is only active if the query is directed to the address http(s)://localhost:xxxxx. So you cannot invoke the filter if working from outside the local environment. But, despite of this, we strongly recommend to deactivate the filter in production scenarios!)

Recording

We explain the sequence of steps by using as example our demo workplace, which is available as “http://localhost:8080/demos/workplace.workplaceRisc.risc?ccstyle=defaultrisc” after installation. - **Please note: when recording your application you need to exchange the “demos” of the URL to the name of your application for all the steps that are explained in the following text!**

Step 1 – Switch the recording to “on”

To do so you just need to call URL “http://localhost:8080/demos/switchrecordon”. Remember: you need to exchange the name “demos” with the name of your application!



The text within the screen tells you the directory, in which the recorded files are placed.

Step 2 - Clear your browser cache

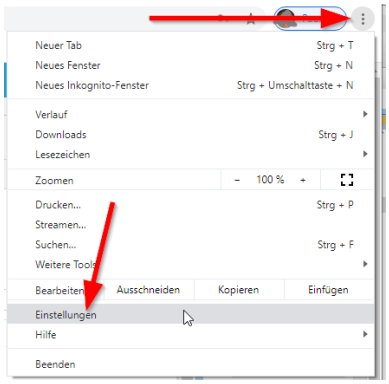
Now: clear your browser cache!

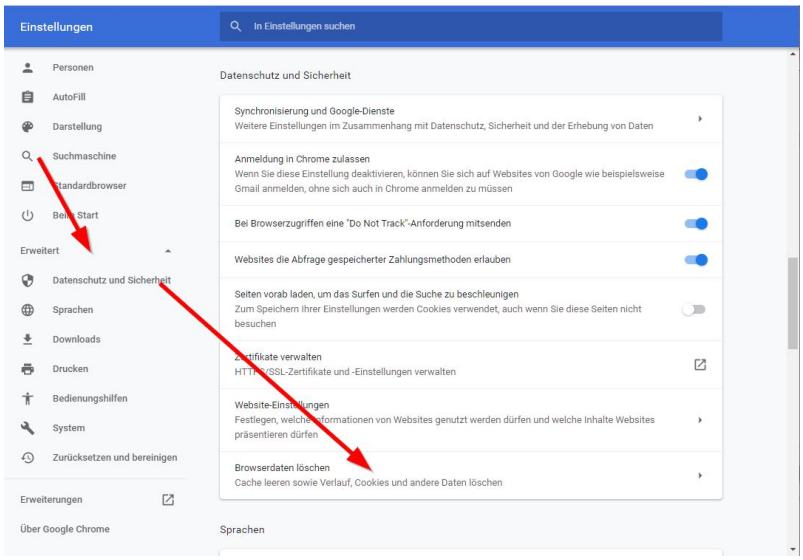
This is 100% important and **the most common problem when recording**. If the browser cache is not cleared then certain resources are not requested through the network and are not recorded on server side as consequence.

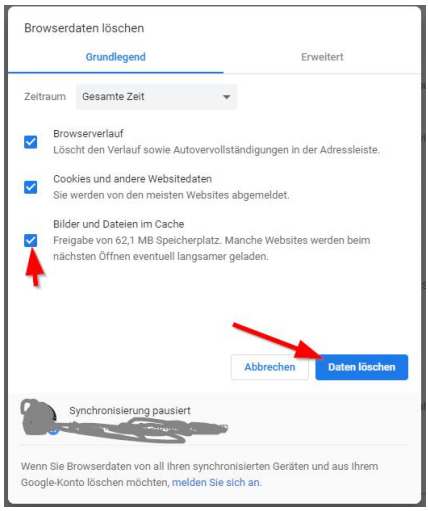
You have to clear the browser cache every time when recoding a sequence!

Unfortunately clearing the cache is much different from browser to browser.

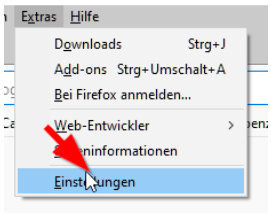
- In Chrome:

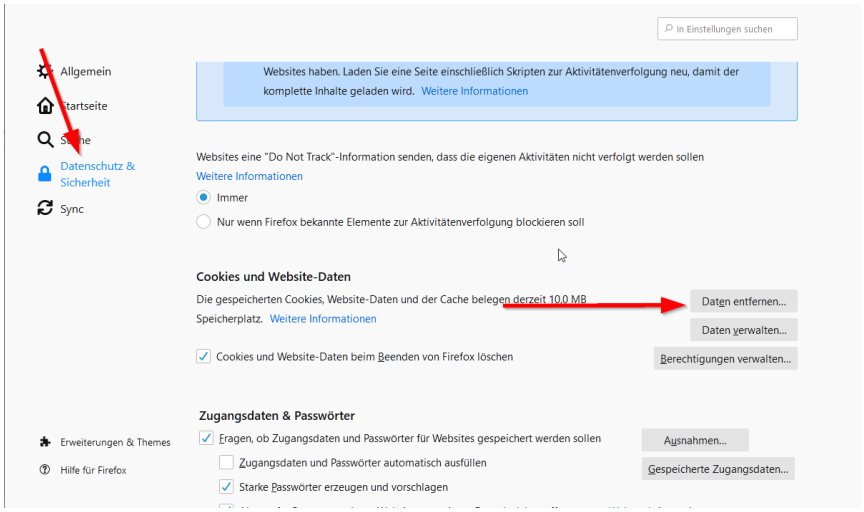


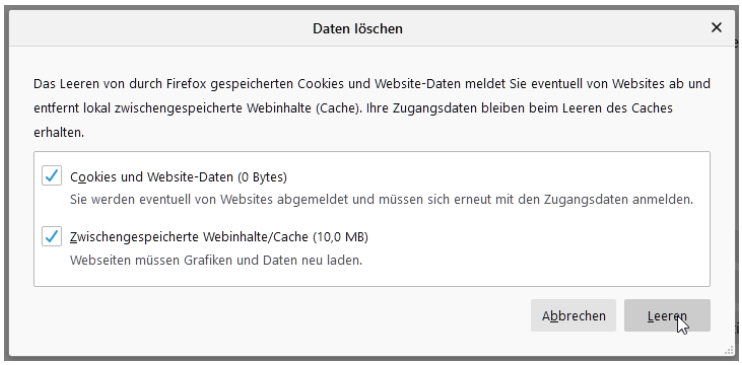




- In Firefox:

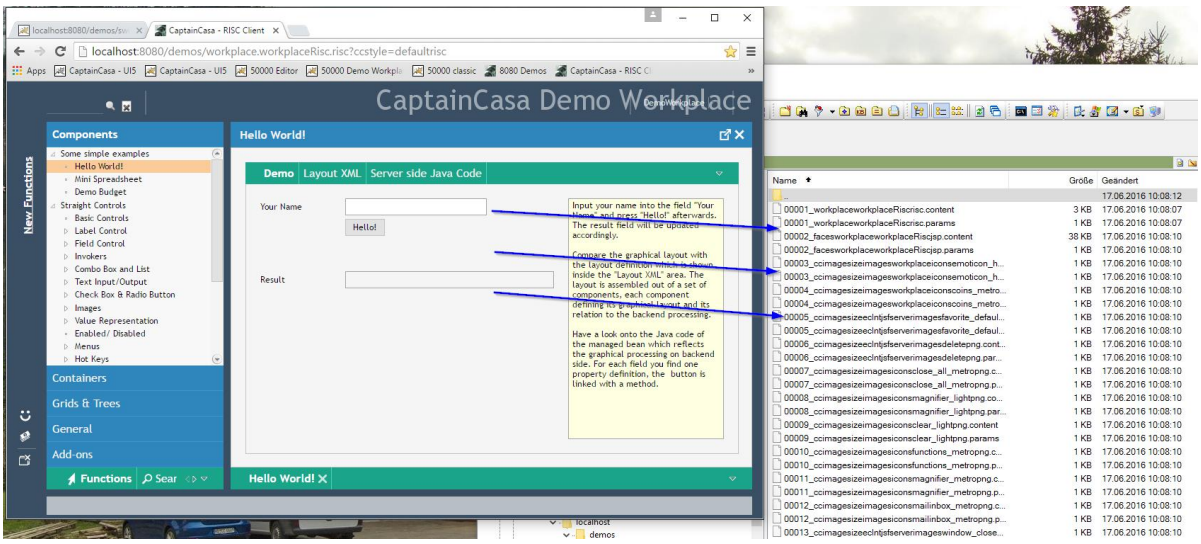






Step 3 - Run the URL of your application

Now you may start (e.g. in a second tab) the URL of your application just as normal. In the background on server side each request/response is recorded.



Per request/response two files are written:

- The content of the response (".content")
- Parameters of the response (".params")

Step 4 – Switch the recording to "off"

Call URL "http://localhost:8080/demos/switchrecordoff" - and again replace "demos" by the name of your application:

Step 5 – Send the recorded information to CaptainCasa

Now you may zip all of the recorded files to one file – and send it to CaptainCasa – together with the start-URL that you used.

...you want to replay on your own?

Well, replaying typically is done on CaptainCasa side, but you may want to do it for any reason on your own.

Just call URL "http://localhost:8080/demos/switchreplayon" and now the server will not work normally anymore but will fulfill all request by finding responses in the recorded files.

When calling your application URL "http://localhost:8080/demos/workplace.workplaceRisc.risc?ccstyle=defaultrisc" then all responses are coming "immediately" from the file system.

Finally switch the replay mode off by calling "http://localhost:8080/demos/switchreplayoff".

Transferring Enterprise Client from Swing/FX to RISC

There's a high level of compatibility between Swing/FX client and RISC client. Nevertheless there are some issues to take care of.

The way to go...

Install the latest version

When coming from the JavaFX/Swing environment then please download the most up to date version from http://www.CaptainCasa.com/java. This is the “big” download, in which also the Swing and FX-client is included. The “normal” download only includes the RISC-HTML client anymore.

Upgrade your project

Just normally upgrade your project. This is typically done by opening the project within the CaptainCasa toolset. A dialog will appear in which you are informed that your project needs some update – and a corresponding button needs to be pressed.

Update your web.xml

There are a couple of new items in the web.xml that need to be added. Copy all the CaptainCasa items from web.xml_template into your web.xml.

Reload/Redeploy your application

Use the “Reload” within the toolset to redeploy your application.

Start your page

Start your outest page (starting page) within the browser in the following way:

```
http://<host>:<port>/<webapp>/[<dir>.]<page>.risc?ccstyle=defaultrisc
```

Example: you run the application on the default Tomcat, your application has the name “testapp”. Inside your application there is an “outest.jsp”, directly located in the webcontent-directory. In this case the URL is:

```
http://localhost:50000/testapp/outest.risc?ccstyle=defaultrisc
```

Example: the same situation but now the outest page is located in the “webcontent/pages”-directory:

```
http://localhost:50000/testapp/pages.outest.risc?ccstyle=defaultrisc
```

Incompatibilities

BGPAINT Attribute

- The BGPAINT attribute is not supported with all the options that were available in the Swing/FX environment. There is a certain compatibility level – but the best idea is to switch to these BGPAINT-commands that are available for all platforms (error, mandatory – and all the commands starting with “bg”, e.g. “bgimage”).

FILEUPLOAD* components

- In Swing/FX the file name always was transferred with its full absolute path (e.g. “c:/xyz/xyz/abc.txt”). In RISC the client only transfers the file name – due to security reasons (e.g. “abc.txt”).

Removing all “non-RISC”-parts

Files to be removed from your project

When having used CaptainCasa Enterprise Client with its Swing or with its JavaFX client then your project contains some files which are obsolete when “only” using the RISC client anymore.

These files may be removed from the project in order to decrease its size and in order to tidy up. When using the default project layout then these files are part of the “webcontentcc” directory – so they are part of the content, that is added by CaptainCasa into your project.

```
webcontentcc
      eclnt
            lib                        <== remove
            libfx                      <== remove
            ui5                        <== remove
      eclntjsfserver
            ht                         <== remove
            htstyle                    <== remove
            javainstall                <== remove
            styles
                  default              <== remove
                  defaultfx            <== remove
                  ccdark               <== remove
                  cceditor             <== remove
                  cceditorlight        <== remove
                  cceditorlightfx      <== remove
                  ccmetrofx            <== remove
                  cctouch              <== remove
                  cctouchlight         <== remove
      WEB-INF
            lib
                  jnlp-servlet.jar     <== remove
```

Cleaning up your Installation

The best was to clean up your CaptainCasa installation is:

- Remove the existing installation
  - You may “remove” your existing installation by just renaming the directory of the installation – in this case you still have your old installation pre-served.
- Install CaptainCasa-RISC in the directory, in which your installation was done

There is only one type of content which you need to preserve, i.e. which you have to save before removing and which you have to bring in again after installation: the project definitions of the CaptainCasa toolset.

So please save all files in...

```
<installdir>
      server
            tomcattools
                  webapps
                        editor
                              config
                                    projects
                                          *.xml        <== to be saved
```

...before removing the installation. And reapply the files at the same location after having installed CaptainCasa-RISC.

After re-install: re-deploy your projects

Part of the CaptainCasa installation is a Tomcat that is used for deploying your projects.

```
<installdir>
      server
            tomcat
```

This tomcat is “empty” if having removed the existing installation and having re-installed. In order to deploy your projects you need to open the project within the CaptainCasa toolset and you need to execute the “Reload Server” function.