

**united  
planet**

**ix** INSIDE INTREXX

UNITED PLANET INTREXX XTREME  
RELEASE 4.5

---


**Contents**

<b>1. Introduction</b> .....	<b>5</b>
<b>2. Business Logic Mode of Operations</b> .....	<b>5</b>
<b>3. System Architecture</b> .....	<b>5</b>
3.1. Important Terms and Technologies .....	6
3.2. Directory Structure.....	7
3.2.1. Main Directories .....	7
3.2.2. Directory Construction of a Portal .....	7
3.2.3. External/htmlroot .....	7
3.2.4. Internal .....	8
3.2.5. Application .....	9
3.2.6. Layout .....	10
3.2.7. system .....	10
3.3. The Files object.js, form.js and api.js .....	10
<b>4. Publishing an Application</b> .....	<b>11</b>
4.1. Calling a Page (First Transformation) .....	11
4.2. Calling a Page (Second Transformation) .....	12
<b>5. JavaScript</b> .....	<b>12</b>
5.1. Request Parameters .....	14
5.2. Velocity .....	16
5.3. Calculations with JavaScript.....	17
5.4. Hiding Controls .....	19
5.5. Inserting Text into View Controls.....	19
5.5.1. Type-defined View Elements .....	21
5.6. Showing and Hiding Elements .....	22
5.7. Show/Hide upon Loading a Page.....	23
5.8. Copy a Data Record with JavaScript .....	24
5.9. Date Calculation with JavaScript, Part I .....	25
5.9.1. Syntax of the Method <i>dateAdd()</i> .....	26
5.10. Date Calculation with JavaScript, Part II .....	26
5.10.1. Syntax of the Method <i>setFullDays()</i> .....	27
5.11. Handling Tables via a JavaScript Object .....	27
5.11.1. Calculate Table Column .....	27
5.12. Automatic Reload of a View Table .....	30
5.13. Dynamic Jump from View Table .....	31
<b>6. Velocity</b> .....	<b>32</b>
6.1. What is Velocity?.....	32
6.2. Where will Velocity be used?.....	32
6.3. References .....	32
6.3.1. Variables.....	32
6.3.2. Properties .....	33
6.3.3. Methods.....	33
6.4. Comments .....	33
6.5. #set.....	34
6.6. #if / #else / #elseif.....	35
6.6.1. #if .....	35
6.6.2. #else .....	35
6.6.3. #elseif.....	35
6.7. Relational and Logical Operators.....	35







6.8. #foreach.....	36
6.9. #include .....	36
6.10. #parse.....	36
6.11. #macro.....	36
6.12. Range Operator.....	37
<b>7. Calling a Page (2nd Transformation) .....</b>	<b>37</b>
<b>8. Context Objects in Velocity .....</b>	<b>37</b>
8.1. \$User .....	37
8.2. \$Request .....	38
8.3. \$Loader (BusLogicCaller).....	38
8.4. \$null .....	39
8.5. \$charset .....	39
8.6. \$lang .....	39
8.7. \$UriBuilder .....	39
8.8. \$ObjectHelper .....	39
8.9. \$Factory .....	39
8.10. \$Response .....	39
8.11. \$Portal.....	40
8.12. \$Session .....	40
8.13. \$DbConnection .....	40
8.14. \$Request .....	40
8.15. \$Loader .....	40
8.16. \$Chat .....	40
8.17. \$OMUC .....	40
8.18. \$VelocityContext .....	40
<b>9. Direct Velocity Call .....</b>	<b>40</b>
<b>10. Static VMs in Velocity .....</b>	<b>40</b>
<b>11. Query from Velocity.....</b>	<b>41</b>
11.1. Example for PreparedQuery .....	41
11.2. Simple Database Query .....	42
<b>12. Velocity and Parameters.....</b>	<b>43</b>
12.1. Variably Querying Table Names .....	44
<b>13. Buttons.....</b>	<b>45</b>
13.1. Properties and Methods of the Action Control Object.....	46
13.2. LinkType .....	47
13.3. ActionId .....	47
<b>14. Ajax.....</b>	<b>47</b>
14.1. What is Ajax?.....	47
14.2. Ajax in Intrex Xtreme .....	47
14.2.1. The upXMLHttp Object .....	48
14.2.2. ActionControls.....	48
14.2.3. Request Processing.....	49
14.2.4. oRequest Object.....	49
14.3. Returning the Ajax and XML Response.....	50
14.3.1. Returning Values .....	50
14.3.2. Sending with Request Values and Form Elements .....	50


### Writing Conventions

In this document, text passages will be displayed in *italics* when they refer to settings in the dialogs shown. Menu items that can be reached from context menus are always available from the main menu as well. Main menu items will not be described unless they are not reachable from the context menu. Context menus can be opened by clicking with the right mouse button on the element described.

A description of the general main menu items can be found in the  *Center* handbook. Programming code in the text will be displayed in the Courier font.

<*xtreme*> refers in the following to your Intrex installation path, such as *C:\xtreme\* on Windows, or */opt/xtreme/* in Linux. The following symbols will be used for the designation of special kinds of information:

-  Important information
-  Tips and background information
-  Links to additional information in an Intrex Xxtreme handbook
-  Directories
-  URLs
-  Buttons in dialogs or assistants

Passages with the  **[Copy-Paste]** note can be directly copied out of the document in order to avoid typing errors.

### Previous Knowledge

In order to understand this documentation, you should have experience with Intrex Xxtreme, as well as experience programming with Java and/or Groovy.

**1. Introduction**

This document will give you an introduction to the internal structures of Intrex Xtreme and explains the use of expert attributes, JavaScript, and the XML/XSL technologies in use. Through this, you will receive an overview of the backgrounds and possibilities of Intrex Xtreme. In order to experience how to reach into the system close to real practice, United Planet offers the seminars *Development I*, *Development II*, and *Development III*. In these seminars, you will learn in a short time how complex activities can be replicated, or how interactive web applications can be realized with the help of Ajax. You can find additional information on the seminars at <http://www.unitedplanet.com/en/academy>.

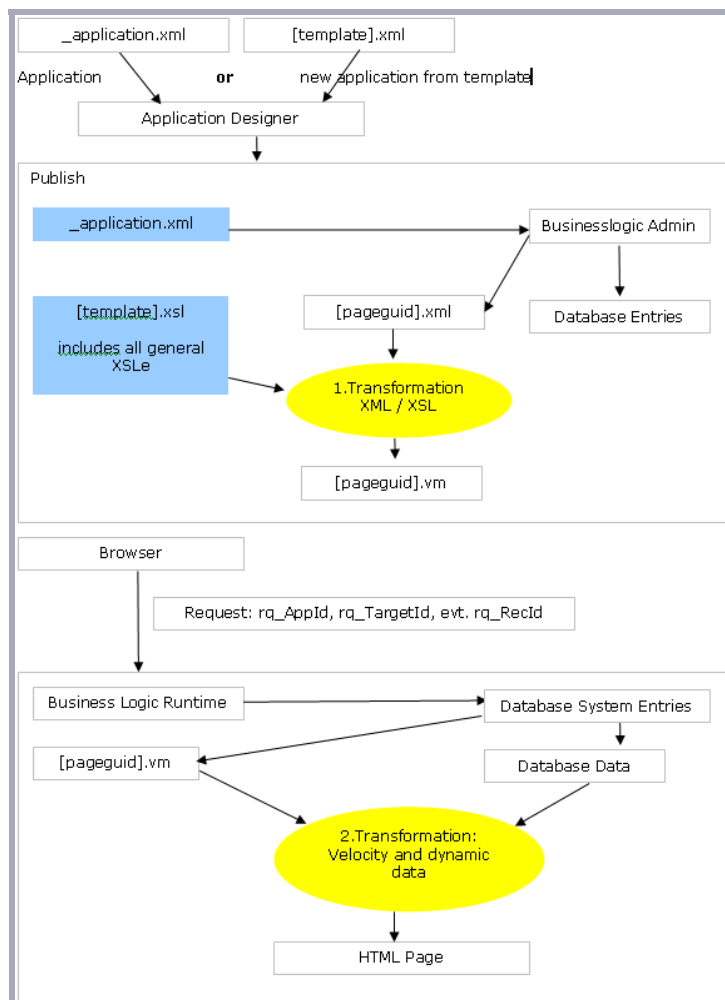
**Tool Recommendations:**

- XML/XSL editor and transformation
- XML-Spy <http://www.xmlspy.com> (home edition)
- Microsoft Visual InterDev

**Free Editors (UTF8 Capable):**

- Notepad++ <http://notepad-plus.sourceforge.net/uk/download.php>

**2. Business Logic Mode of Operations**



**3. System Architecture**

The Intrex Xtreme Portal Manager communicates with the Intrex Xtreme Application Server via SOAP on the basis of Apache Axis. SOAP is an XML-based protocol for communication between systems in distributed environments.

The manager creates the applications, the basic format of which is XML, through a transformation via XSL in the Velocity Markup Format into the desired output format. Through this transformation, new output media can be supported quickly and reaction to changes in forms of communication can be quickly accomplished. By using Velocity Markup, high performance is ensured.

The Intrex Xtreme Application Server takes over the tasks of

- User management and authentication
- Session management
- Integration of business logics
- Access to databases.

Using connectors (contained in delivery package), various web servers can be supported, such as Apache, Tomcat, and Microsoft Internet Information Server.

### 3.1. Important Terms and Technologies

#### XML

Extensible Markup Language, a further development of the SGML standard. As opposed to SGML documents, XML documents require no schema description in the form of a DTD. They consist primarily of text and tags; the tags create a tree structure in the document. If the XML document is properly structured, meaning that the tags are properly nested within one another, it will be designated as *well-formed*. If a DTD exists for the document in addition, it will be designated as *valid*.

#### XML Data

A method for the input of XML schemas. XML data expands the capabilities of DTDs in regard to data types.

#### XML Parser

An XML parser is a processing program that reads an XML document and returns the structure as well as the properties of the data. It breaks up the data into individual parts and passes them on to other components. If the parser does not only check whether the document is *well-formed* (correctly structured) based on XML rules, rather it also performs a check based on an XML-DTD, the parser will be designated as a *validating* parser. There are various XML parsers on the market:

- Xerces Java Parser from Apache/IBM
- XML4j from IBM
- XML Parser from Microsoft
- XML Parser for Java from Oracle
- Lark from Tim Bray
- Project X from Sun
- XP from James Clark

#### XML Schema

The XML Schema is the most current procedure for the preparation of XML schemas, which has been published in two W3C working drafts (Structures and Datatypes). The schemas serve to describe the structure and restrict the contents of XML documents, as well as the assignment of data types to XML element types and attributes. Earlier schema designs encompass DTDs, XML data, XDR, RDF, SOX, DCDs, XSchema, and DDML.

#### XPath

XML Path Language, a language for addressing parts of an XML document, which can be used by XSLT as well as by Xpointers. The language consists mainly of addressing paths and addressing expressions:

Via *child::para[position=(1)]*, for example, the first para-child of the current context node could be selected. As expressions, normal elements such as Boolean operators, numbers, node sets, and so on may be used.

**XSD**

XML schema declaration, a suffix of files which contain a description of an XML schema according to the XML schema specifications.

**XSL**

Extensible Stylesheet Language, a language for the creation of stylesheets, which describe how data, which will be sent using Extensible Markup Language (XML) via the web, will be presented to the user.

**XSLT**

XSL Transformations, a language for the transformation of XML documents into other XML documents. XSLT has been developed as a part of XSL, a stylesheet language for XML. XSL possesses an XML vocabulary over and above that of XSLT for the input of formatting information. XSL defines the format of an XSL document by using XSLT. With XSLT, it will be described how the document will be transformed into another XML document that uses the formatting vocabulary.

**GUID**

Global Unique Identifier – worldwide unique number

















**Velocity**

Java-based template engine, which enables dynamic data to be integrated into static websites via Java class calls.






**3.2. Directory Structure**

**3.2.1. Main Directories**

The directory structure of Intrex Xtre is divided into the following main areas:


-  *bin*                executable files
-  *cfg*                 general configuration files
-  *client*             files that will be required for the Portal Manager
-  *docs*              Intrex Xtre documents
-  *export*            Portal exports
-  *groovy*            Groovy script files
-  *help*              Intrex Xtre online help
-  *hsqldb*            HSQL database files
-  *jre*                 Java Runtime Environment for Intrex Xtre
-  *lib*                program source code
-  *log*                portal log files
-  *org*                all created portals
-  *orgtempl*        portal templates
-  *res*                resources for Setup and Portal Manager
-  *tmp*                temporary files
-  *upplib*            Sources for online update


**3.2.2. Directory Construction of a Portal**

-  **org**                A portal is always divided into the main areas *internal*, *external*, and *log*.
  -  **myportal**
    -  **external**
    -  **internal**
    -  **log**

**3.2.3. External/htmlroot**

The *external/htmlroot* directory contains all files that can be directly accessed via the webserver. This includes:

-  *css*  
Cascading Stylesheet files (standard, and those created in the Portal Designer)

 *download*

Setupfiles of downloads (e.g. Office Integration)

 *htc*


JavaScript files which simulate the CSS behaviour of the IE 6.0.

 *html*

Miscellaneous HTML Files


 *images*


Image files that will be used by Intrexx Xtreme. If images are inserted directly into applications, a directory will be automatically created in this directory with the name of the GUID of the application. The image files of the application are contained in this directory.

 Changes to the image files in this directory will be overwritten the next time the application is saved. Please only make changes to image files directly in the Application Designer.

 *include*

JavaScript files that will be implemented by Intrexx Xtreme. If JavaScript is used by applications, a directory with the GUID of the application as its name will be created in this directory, which contains the JavaScript.

 Changes to the JavaScript file in this directory will be overwritten the next time the application is saved. Please only change the JavaScript in the Application Designer, or copy the contents of the script editor back into the Application Designer.

The directory  *external/htmlroot/include/custom* contains the file *custom.js*, which is integrated into all pages in Xtreme. In this file, you can make individual adjustments. The file will not be overwritten by an update.

 *is*

Registry Files

 *thirdparty*

Files from third-party providers, whose applications are in use by Intrexx Xtreme. Currently, the files for HTMLAREA are located here, a free HTML editor that can be activated for longtext fields.

 *tmp*

Temporary files, which will either be removed upon deleting the portal, or by a user logging off (such as images that are no longer in use).

 *userfiles*

If files are uploaded via the FCK editor, they will be saved to this folder. The various data types will be saved to additional subfolders.

 *WEB-INF*

Configuration file for the Tomcat web server.

### 3.2.4. Internal

The *Internal* directory contains all files that will be processed directly in the Intrexx Xtreme server (such as for the handling of requests).

 *application*

All files that relate directly to applications (XML, XSL, VM)

 *bpee*


Configuration files for *BPEE* scripts, which will be executed for services such as web service calls.

 *cfg*

Configuration files for their corresponding portals.

 *files*

TheUpload files in applications

 *layout*

Files directly related to the Portal Designer (XML, XSL, VM)

 *lucene*

Files for the Lucene search engine

 *mailroot*


Files that will be sent via eMail

 *schemas*

XML schema files (definition files for Intrex document types)

 *statistics*

Logfiles for statistics

 *system*

Files for special system tasks (such as appointment series, language management)

 *tmp*


Temporary portal files

 *uploadfiles*

Temporary directory for uploaded files

 *usrimg*

Images that have been saved under specific users

 *webservice*

Configuration files for registered web services, web service calls, and provided web services

 *workflow*

Process descriptions

 *wsrp*

Configuration files for WSRP

### 3.2.5. **Application**

The *Application* directory contains all files that exist in direct relation to applications.

 *Template*

Subdirectory with the names of existing templates. These directories contain the corresponding images for the applications, as well as the XML file that describes this application.

 *vm*

The applications will be transformed into the format *Velocity Markup* via XSL Files. These files can be parsed and executed from the application server. Here the Java business logics will be called.

 *xml*

Files for the validation of XML files

 *xsd*

Files for the validation of XML files

📁 *xsl/*

Files for the transformation – in the current version, only HTML and text files will be created. The text files serve the sending of eMails (with the *PlainText* setting). This directory contains files as well that are specially created for template types.

📁 *xsl/common*

XSL files that will be used by multiple output formats.

Located within both the directory 📁 *xsl/common* and in 📁 *xsl/html/*, there will be found a directory 📁 *custom*, which will not be overwritten upon an update. Individual adjustments can be saved here.

### 3.2.6. Layout

The *Layout* directory contains all files that exist in direct relationship to the Portal Designer.

📁 *assets*

Image families, which can be assigned in Intrex Xtre from the Portal Designer, such as the image family that is contained in the delivery package, *Intrex\_Default*. Individual image families can be created and saved to this directory. When publishing a portal, the directory will be copied into the directory 📁 *external/html/images/assets*.



Direct changes to image files in the *external* directory will be overwritten upon saving the active layout.

📁 *stored*

All layouts saved on the server

📁 *vm*

All files that have been created by the Portal Designer in an XML/XSL transformation, or those that find use as complete VMs from this directory (such as portlets).

📁 *xml*

The current layout, in the form of an XML file. The file *menudesigner.xml* can be found here as well. This contains the menu structure and special formatting for individual menu items.

📁 *xsl*

All files for the transformation of the layout XML into the resulting VM files.

### 3.2.7. system

This directory contains, essentially, static Velocity Markup files that take on corresponding specific tasks.

📁 *vm/html/actioncontrol*

Additional controls that can be assigned to the individual frames in the Portal Designer. Individual additional controls can be developed by adjusting the XML file.

All additional directories are self-explanatory.

### 3.3. The Files *object.js*, *form.js* and *api.js*

In the directory 📁 *external/htmlroot/include* you will find the JavaScript files described in the following, which contain the definitions and methods for the adjustment of Intrex Xtre.

*object.js*

This file contains the entire window handling in Intrex Xtre. Additionally, the object definitions of action controls are contained here.

*form.js*

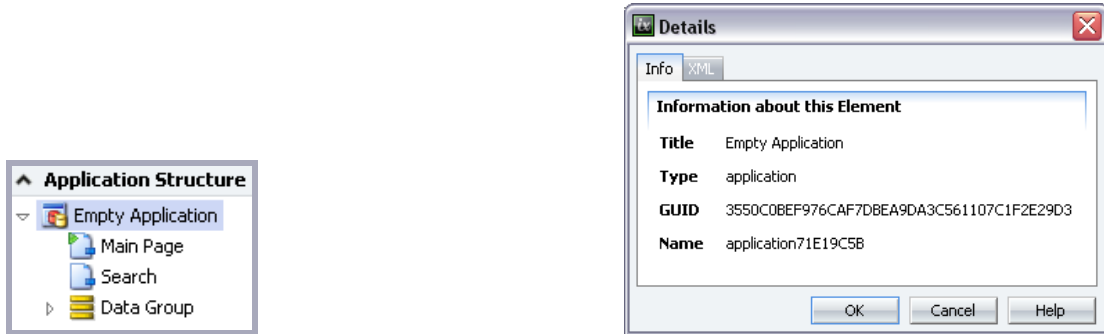
This file contains all definitions of edit elements in Intrex Xtre.

*api.js*

The file *api.js* contains helpful methods for the handling of values.

#### 4. Publishing an Application

When publishing a new application, the *\_application.xml* will be saved as a basis for the application to the directory *internal\application\xml\[GUID of the application]*. It contains the entire application in XML notation for the Application Designer. The *GUID* of the application can be ascertained in the Application Designer by selecting the application node and pressing the <F4> key. The expert mode must be activated from the menu *Extras/Options* for this function to be available.

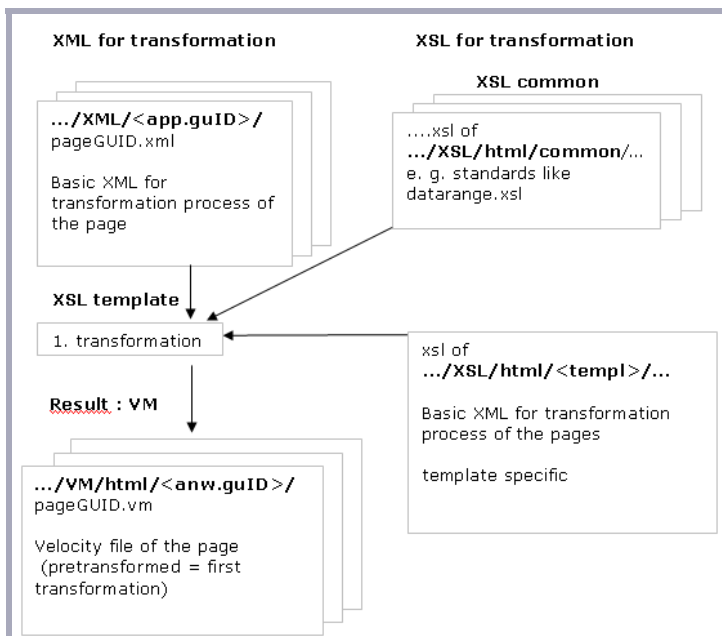


Additionally, for each page of the application, a page XML will be written, and resulting from this the page VMs will be created and saved through a first transformation process to include the XSL. For each page of the application, the following will be created:

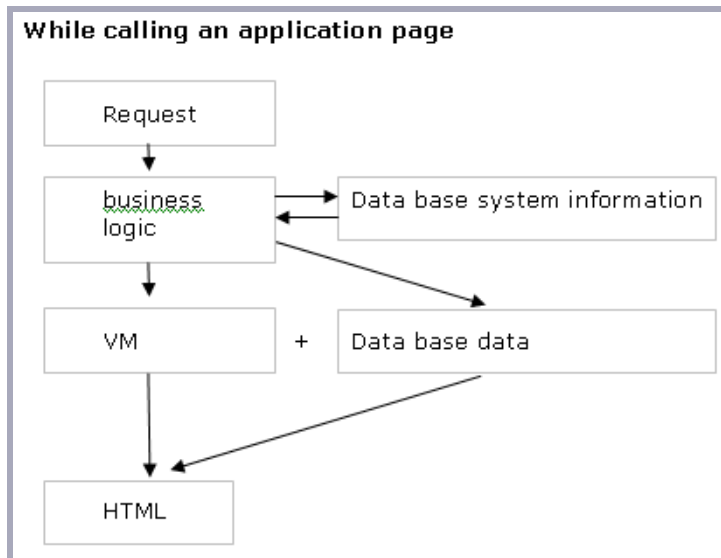
- XML/[application GUID]/[page GUID].xml
- VM/[application GUID]/ [page GUID].vm

All system data required for the application will be entered to the database.

#### 4.1. Calling a Page (First Transformation)



## 4.2. Calling a Page (Second Transformation)



## 5. JavaScript

### Where will JavaScript be used?

JavaScript functions can be used in static VMs or written via XSL in the VM. The HTML page will be generated from this. JavaScript can be called and executed in the browser in this way. In order to include individual JavaScript functions, a JavaScript editor is available in the Application Designer for use.

### How can parameters be used from Velocity in JavaScript?

In the JavaScript editor, no direct Velocity calls can be saved, as the functions will not be transformed. A Velocity call can, however, be input as a parameter to a JavaScript function.

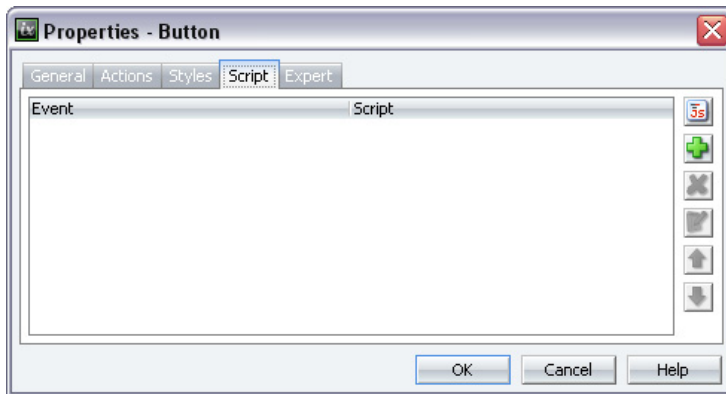
### JavaScript Objects


In Intrex Xtreme, all controls are object-oriented in their construction. In addition to normal HTML controls, a specific object from United Planet exists. These objects are defined in the file *object.js*.

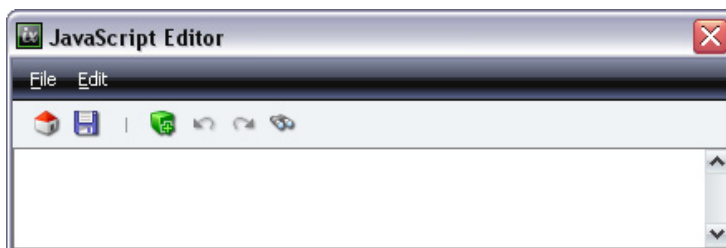
Each HTML control contains the additional method *oUp*, through which the United Planet object can be addressed. The objects contain additional methods, such as a method that takes a number as input, removes its formatting, and returns a number with which calculations can be run


(such as `var f_Float = oHtmlFloat.oUp.toJSNumber(oHtmlFloat.value);`).

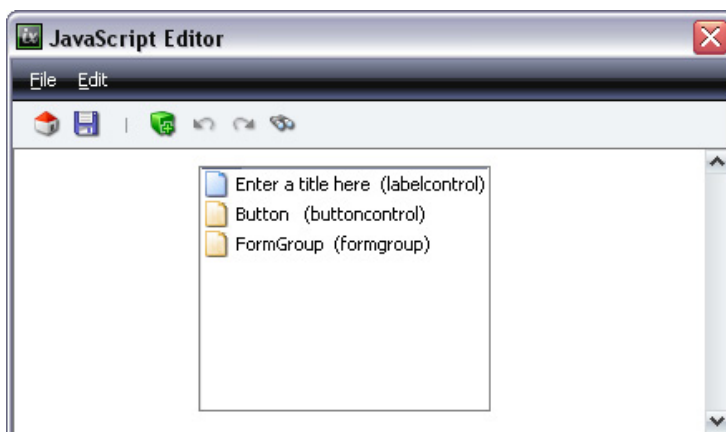
With JavaScript, the functionality of Intrex Xtreme applications will be expanded. The *Script* tab is located on the properties dialog of pages and edit elements. All kinds of event-controlled functions can be entered here, such as those that output user notifications or validate inputs.



By clicking  *Script*, the script editor will open. Enter your JavaScript functions here.



 *Insert control edit* or right-clicking on the desired position in the editor will open a list, from which edit and view elements that are found on the current page can be selected.




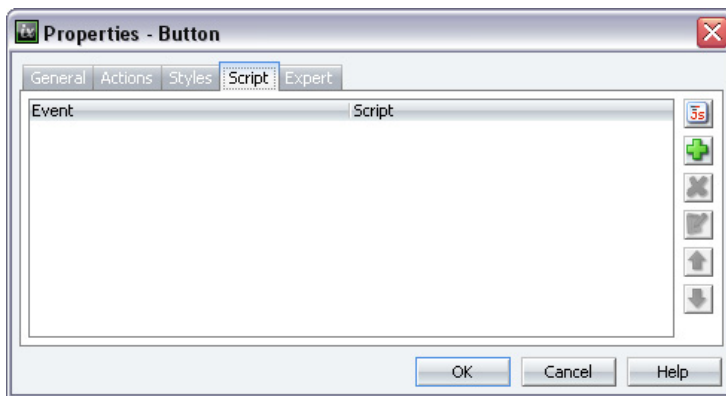
Select the element that should be referenced in the script. The following programming code will be inserted in the script editor to the current cursor position:


```
getElement("GUID of the element")
```

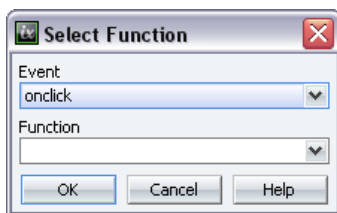
Create your references to the HTML object with the syntax

```
var NameOfElement = getElement("GUID of the element");
```

After completing your function, click  *Save* to save the script.





With  *Insert script call*, the function can now be assigned to the desired event.



Each element provides specific events for use, corresponding to their function. A button, for example, provides the events *onclick* and *ondblclick*.

The script will always be run, after publishing the application, whenever the event to which the function is assigned occurs in the browser.

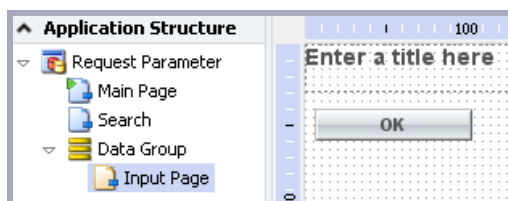
 The script functions are only available in the current application for use. When saving the application, the contents of the script editor window will be extracted to a JavaScript file. The file will be saved to the installation directory of Intrex Xtreme ( `<xtreme>\org\external\htmlroot\include\[GUID of the application].js`). This file will not be reimported. Direct changes to this file have no effect. Please change the script only in the script editor.


### 5.1. Request Parameters

A parameter should be transferred to the next page. In the following example, a request parameter will be transferred from a button to the next page, and there read out with JavaScript.

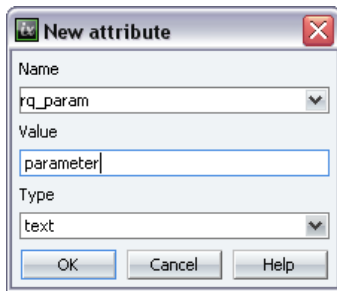
#### Exercise 1

Create the application *Request Parameter*, based on the template *Empty Application*. On the edit page, define a button with the title *OK* and jump target to the main page of the application.



Switch to the *Expert* tab and create a new attribute by clicking  *New*.

**Name:** rq\_param  
**Value:** parameter  
**Type:** text




The value will be transferred to the server in the URL and entered to the request object. All expert attributes that begin with *rq\_* will be sent as request parameters to the following page. Please take note of uppercase and lowercase letters here. On the main page, the following script will be called in the *onload* event of the page:

```
function alertParameter(p_strParam)
{
    // check if set
    if(p_strParam)
        alert(p_strParam);
    return true;
}
```

Enter the function in the onload event as follows:

```
alertParameter('$!Request.get("rq_param")');
```

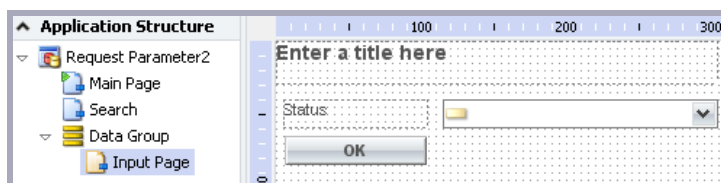
Define the edit page as the start page, save, and test the application in the browser. Clicking  *OK* on the edit page will execute the jump to the main page. The value of the request parameter will be output in a message.



## Exercise 2

A parameter can be transferred as a variable as well, such as when the contents of an edit element are delivered. In the following example, a value will be transferred from an edit page to the main page and read out there with JavaScript.

Create the application *Requestparameter2* based on the *Empty Application* template. Define a selection list on the edit page with the title *Status*, the data type *Text*, and the two user-defined entries *Release* and *Editing*. Create a button with the title *OK* and link to the main page.



Open the script editor via the properties dialog of the button and insert the following script:

```
function sendParameter(p_oHtmlButton)
{
    var oHtmlStatus = getElement("552C...FBA"); /*Status dropdowncontrol*/
    var text = "";
```

```

        if(oHtmlStatus.value == 'Release')
            text = "The document has been released.";
        else
            text = "The document has not yet been released.";

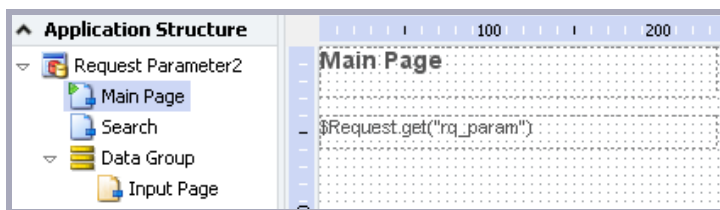
        p_oHtmlButton.oUp.oTarget.addParam =
        Helper.setQsValueByParam("rq_param",text,
        p_oHtmlButton.oUp.oTarget.addParam);
        return true;
    }

```

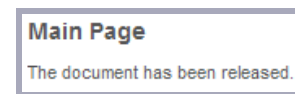
Assign the *onclick* event of the button to the function as follows:

```
sendParameter(this);
```

On the main page, define a view element of type *static text* and select the option *Only default language (such as for programming purposes)*. Enter the title `$Request.get("rq_param")`.



Define the edit page as start page. Now when the value *Release* is selected from the *Status* selection list on the edit page in the browser, clicking **OK** will load the main page, where the text entered in the script will be output.

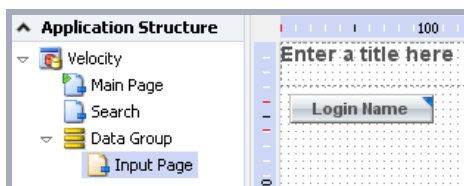


## 5.2. Velocity

A parameter should be read out of the Velocity context and transferred as request parameter to the following page.

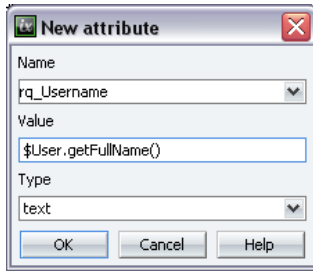
### Exercise

Create the application *Velocity* based on the template *Empty Application*. Define on the edit page a button with the title *Login Name* that links to the main page.

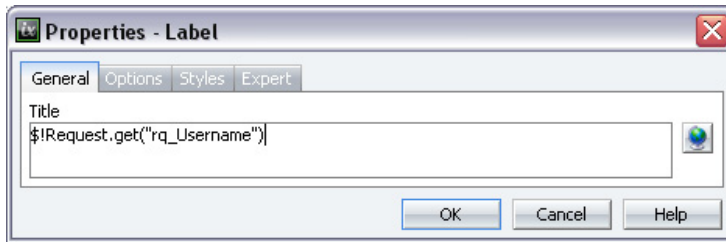


Enter a new attribute in the properties dialog of the button on the *Expert* tab:

**Name:** rq\_Username  
**Value:** \$User.getFullName()  
**Type:** text

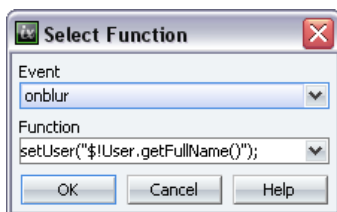


On the main page, define a view element of type *static text* and select the option *Only default language (such as for programming purposes)*. Enter the title `!Request.get("rq_Username")`.



**i** If a Velocity command returns NULL or is undefined, the entire expression will be output. The exclamation point behind the "\$" sign suppresses the output of the Velocity command, if for example no full name has been entered in the User Manager. Do not forget that, for the user name you have used to log on in the browser, the full name must be entered in the user information. If you fill this field afterwards, you must log out and log in again in the browser.

The Velocity expression reads the current login name out of the session information and transfers it with a mouse click as a request parameter (transfer in the URL) to the page to be called up. A direct parameter transfer via the Velocity context is possible as well:



Define the edit page as the start page here and save the application.

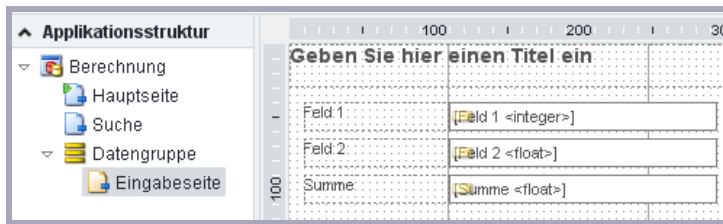
### 5.3. Calculations with JavaScript

#### Exercise

Create the application *Calculation* based on the *Empty Application* template. In the following example, a calculation will be performed on entered values on an edit page and the results will be written in an additional edit field.

Enter the three edit fields onto the edit page:

Title	Data Type
Field 1 Integer	Integer
Field 2 Decimal number	Decimal number
Sum	Decimal number



Open the properties dialog of the edit field *Field 2 decimal number*, click on the *Script* tab, and define the following function in the script editor:

```
function calcValues()
{
    var oHtmlInt = getElement("E8F..8F0"); /*Field 1 integercontrol*/
    var oHtmlFloat = getElement("D6..8C8"); /*Field 2 floatcontrol*/

    //Convert numbers to JavaScript, remove the formatting
    var l_Int = oHtmlInt.oUp.toJSNumber(oHtmlInt.value);
    var f_Float = oHtmlFloat.oUp.toJSNumber(oHtmlFloat.value);

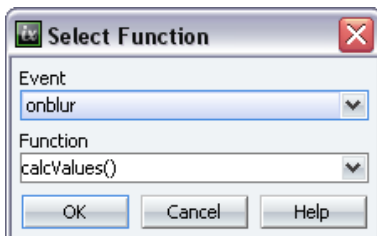
    //Calculation
    var l_Sum = l_Int + f_Float;

    var oHtmlSum = getElement("49..58"); /*Sum floatcontrol*/

    //Write the results with formatting
    oHtmlSum.value = oHtmlSum.oUp.toLocalFormat(l_Sum);

    return true;
}
```

This function will be called in the *onblur* event of the second edit field.



Save the application and open it in the browser. On the edit page, the sum can now be calculated from the two edit fields.

Enter a title here	
Field 1	2
Field 2	4,00
Sum	6,00

The *api.js* provides functions for use that make possible a shorter and more comfortable solution:

```
function calcValuesApil()
{
    var oHtmlInt = getElement("32C1...E80F"); /*Field 1 Integer
integercontrol*/
    var oHtmlFloat = getElement("C134...3D66"); /*Field 2 Decimal number
floatcontrol*/

    //Convert numbers in JavaScript, remove formatting
    var l_Int = getNumberObject(oHtmlInt);
```

```

    var f_Float = getNumberObject(oHtmlFloat);

    //Calculation
    var l_Sum = l_Int + f_Float;

    var oHtmlSum = getElement("AF81...DAE3"); /*Calculate sum
floatcontrol*/

//Write the results with formatting
writeLocalString(oHtmlSum, l_Sum);

    return true;
}

```

This solution can also be further shortened:

```

function calcValuesApi2()
{
    var oHtmlInt    = getElement("32C1...E80F"); /*Field 1 Integer
integercontrol*/
    var oHtmlFloat = getElement("C134...3D66"); /*Field 2 Decimal number
floatcontrol*/
    var oHtmlSum = getElement("AF81...DAE3"); /*Calculate sum
floatcontrol*/

    //Calculate and write results
    calculate(oHtmlInt, oHtmlFloat, "+", oHtmlSum);
    return true;
}

```

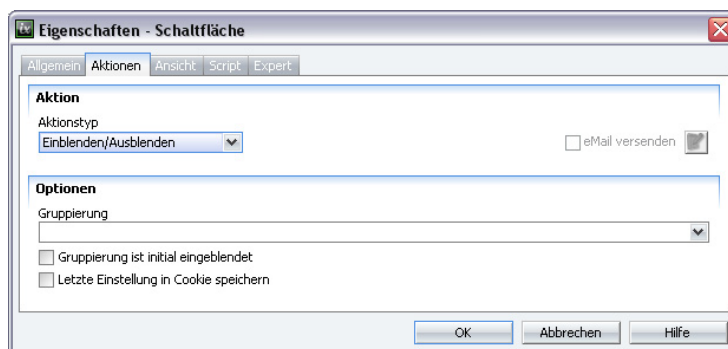
#### 5.4. Hiding Controls

A grouping on a page should be hidden.

##### Exercise

On the page, enter an additional button with the title *hide*. Group all elements that are to be hidden, including the *hide* button. Give the grouping the title *Hidden* via the properties dialog.

Initialize the *Hidden* button with the function *Show/Hide* and enter the container to be hidden. The initial condition is hidden. In this way, all elements in the *Hidden* grouping will not be shown in the browser.



#### 5.5. Inserting Text into View Controls

The current *Data Record ID* should be read out via Velocity and interpreted in a JavaScript method. Depending on the value, an output should appear in a separate view field. In order to read values out of view controls via JavaScript and to write in view controls, there are comfortable methods in the file *api.js*:

```

/*
DESCRIPTION: reads a text node out of a view field or static text
PARAMETERS: oHtml: referent to the html input control
RETURN:      Value
EXAMPLE:     getTextValue(getElement("F28BA735...A1433"));

```

```

*/
function getTextValue(oHtml)
{
    return Browser.getValue(oHtml);
}

//Write
/*
DESCRIPTION: writes a value to a view field or static text
PARAMETERS: oHtml: Reference to the html edit control
              strValue: value
RETURN:      true/false
EXAMPLE:     setValue(getElement("F28BA...66A1433"), "myString");
*/
function setValue(oHtml, strValue)
{
    return Browser.setValue(oHtml, strValue);
}

```

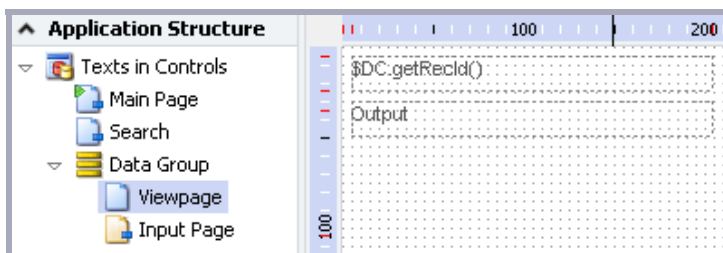
Create the application *Texts in Controls* based on the *Empty Application* template. On the edit page, define an edit field with the title *Data Record* (data type: text) and a *Save* button with a link to the main page.



Next, define a view page in the data group. On the page, create a view element of type *Static Text*. As the title of the field, enter the Velocity command to read out the current data record ID:

```
$DC.getRecId()
```

In the properties dialog (*Options* tab), select the option *Only default language (such as for programming purposes)*. Create a second view element of type *Static Text* and give it the title *Output*.



Write the following JavaScript function:

```

function changeViewcontrol()
{
    var oHtmlRecordid = getElement("C..E"); /*$DC.getRecId()
labelcontrol*/
    var oHtmlOutput = getElement("C2...DB5"); /*output labelcontrol*/

    //Read out the current data record ID
    var recordid = getTextValue(oHtmlRecordid);

    //Interpret and construct the output text
    var text = "";
    if(recordid == 1)
        text = "This is the first data record";
    else

```

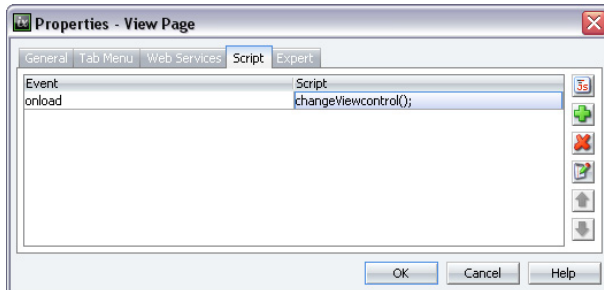
```

        text = "This is an additional data record";

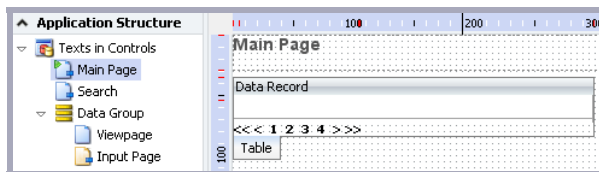
//Set the text
setTextValue(oHtmlOutput, text);
return true;
}

```

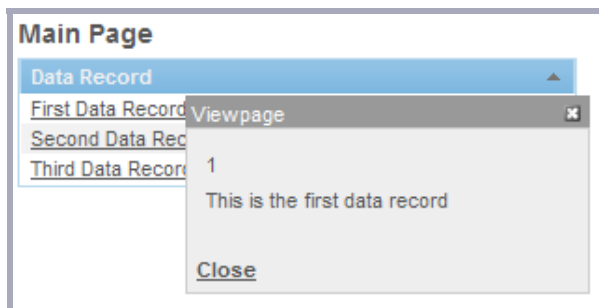
Call up the function in an *onload* event of the view page.



Now, enter a view table on the main page for the *data group* with the column *Data Record* and link to the view page in a *tooltip*.



Save the application. In the browser, data records can now be entered to the edit page. If these are loaded via the view table, the text defined in the script will be output.



### 5.5.1. Type-defined View Elements

If calculations are to be executed with values in view elements, the value of the view element can be read out and then converted with individually defined JavaScript methods into a data type for calculations. From Version 2.5 onwards, these conversion methods will be prepared with the expert attribute:

**Name:** jsubject  
**Value:** true  
**Type:** boolean

The expert attribute builds a bidirectional relationship between the HTML object of the view control and the corresponding up-object for its use in JavaScript. For edit controls, the bidirectional relationship is so by default. This allows, for example, calculations to be run with the value of a date view field, whereby the control-specific formatting options will be taken into account. The expert attribute can be implemented for the following control types:

- upIntegerVControl
- upFloatVControl
- upCurrencyVControl

- upDateTimeVControl
- upDateVControl
- upTimeVControl
- upCheckVControl


The value of the view controls can be directly queried via the function

```
getElement("GUID").oUp.displayValue
```

For the view element *upCheckVControl*, the value can be queried with the function

```
getElement("GUID").oUp.checked
```

The return value is *true* or *false*.

 A query using *Browser.getValue()* is only possible in certain circumstances, as not only the contents of the (visible) view field is contained in the return value, rather also the (invisible in the browser) JavaScript block. Because of this, the new methods *displayValue* and *checked* are available only for use in this combination with the view controls.

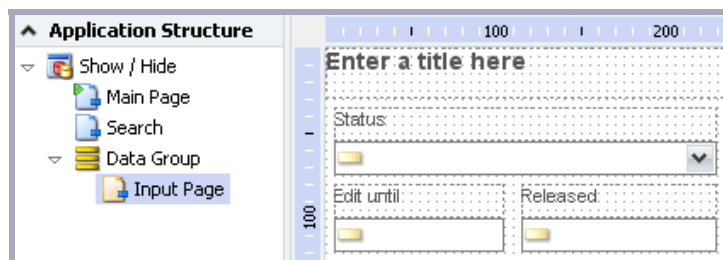
## 5.6. Showing and Hiding Elements

Depending on the selection of a value, control elements should be shown or hidden. On an edit page, an input will be chosen from a selection list. Depending on the value that is selected, various additional controls will be shown.

### Exercise

Create the application *Show / Hide* based on the *Empty Application* template. Define a selection list on the edit page with the title *Status* and the user-defined values *Release* and *Editing*.

Enter two additional edit fields to the page: *Edit until* with data type *Date* and *Released* with data type *Text*. Group each of the edit fields with their title. Give the first grouping the title *Editing* via the properties dialog, and the second the title *Release*.



Define the following function for the *onchange* event of the selection list:

```
function showControls()
{
    var oHtmlStatus = getElement("51E2...6A02"); /*Status
dropdowncontrol*/
    var oHtmlRelease = getElement("FA2F...0E12"); /*Release
simplegroup*/
    var oHtmlEdit = getElement("8A90...92AB"); /*Edit
simplegroup*/

    if (oHtmlStatus.value == 'Release')
    {
        //Show release
        oHtmlRelease.style.visibility = "visible";
        oHtmlRelease.style.display = "block";

        //Hide edit
        oHtmlEdit.style.visibility = "hidden";
        oHtmlEdit.style.display = "none";
    }
}
```

```

        return true;
    }
    if (oHtmlStatus.value == 'Editing')
    {
        //Hide release
        oHtmlRelease.style.visibility = "hidden";
        oHtmlRelease.style.display = "none";

        //Show edit
        oHtmlEdit.style.visibility = "visible";
        oHtmlEdit.style.display = "block";
        return true;
    }

    //Hide edit
    oHtmlEdit.style.visibility = "hidden";
    oHtmlEdit.style.display = "none";

    //Hide release
    oHtmlRelease.style.visibility = "hidden";
    oHtmlRelease.style.display = "none";
    return true;
}

```

Upon selecting an entry in the selection list in the browser, the corresponding grouping will be hidden.

A screenshot of a web form. At the top, there is a dropdown menu labeled 'Status' with 'Editing' selected. Below it is a text input field labeled 'Edit until' which is currently empty and visible.

A screenshot of a web form. At the top, there is a dropdown menu labeled 'Status' with 'Release' selected. Below it is a text input field labeled 'Released' which is currently empty and visible.

- ❗ The *onchange* event will only be activated upon selection in a selection list. Define the selection list in the properties dialog (*General* tab) as a mandatory field (*Input Required*), so that a selection must be made. This ensures that the *onchange* event will be activated in every case.

In order for the two additional edit fields to not be visible or to be shown depending on the defined value, the function should also be called as an *onload* event of the page as well. With the same procedure, all other kinds of controls can be shown or hidden as well.

Automatic showing or hiding can also be accomplished with the help of a button, the action of which has been defined in the properties dialog as *Show/Hide*. The advantage of this is that for slow browsers, the elements to be hidden will not be shown for a short time, but will rather be hidden from the beginning.

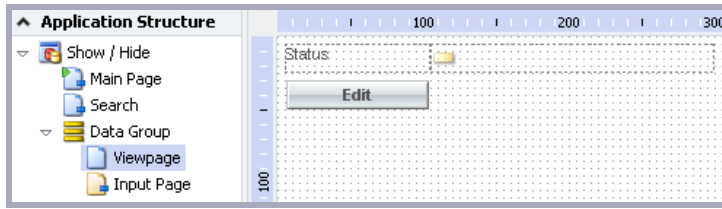
- ❗ Showing and hiding containers can be controlled with the following function in JavaScript: `oButton.oUp.flipFlop(true)`. If the value is *true*, the container will be shown; if the value is *false*, it will be hidden.

### 5.7. Show/Hide upon Loading a Page

Control elements can be shown or hidden upon loading the page (and only here) in the expert mode as an alternative. Depending on a value in a view field, a button should be shown or hidden upon loading the page.

#### Exercise

Define a view page in the application *Show / Hide*. Enter a view field here and connect it to the data field *Status*. In addition, create an *Edit* button.



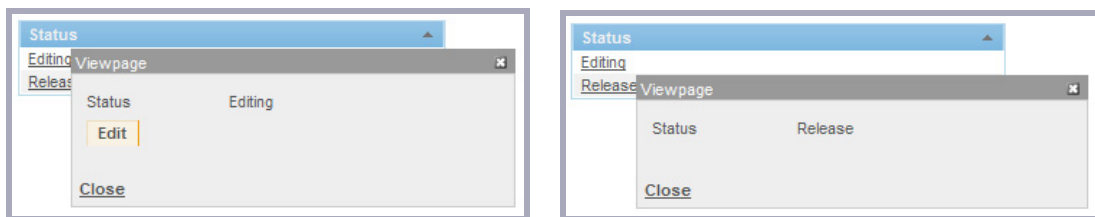
Open the properties dialog of the button. There, switch to the *Expert* tab. Enter the following new expert attributes there:

**Attribute** additionalcheck  
**Value** \$DC.getAttribute('textviewcontrol123456')  
**Type** Text

**Attribute** valueadditionalcheck  
**Value** the comparison value – in our example *Editing*.  
**Type** Text

Replace the value of the first entry (*textviewcontrol123456*) with the actual name of the view field. This can be ascertained with the *F4* key or also by viewing the *Expert* tab in the properties dialog. On the main page of the application, define a view table with a link to the view page you just created, in a *Tooltip*.

From the edit page, create an entry with the status of *Release* and an entry with the status *Editing*. If a data record with the status of *Editing* is now selected, the button will be shown. Upon a status of *Release*, the button will not be shown.

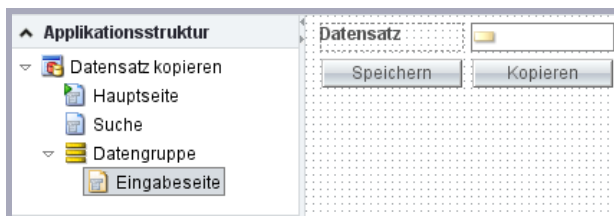


### 5.8. Copy a Data Record with JavaScript

An existing data record should be copied within the same data group.

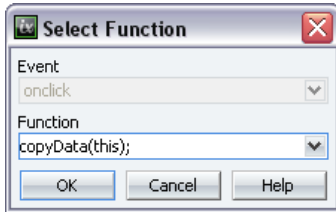
#### Exercise

Create the application *Copy Data Record* based on the *Empty Application* template. Enter an edit field with the title *Data Record* on the edit page. Define a button to save the data record with a link to the main page and the settings *Close popup/tooltip* and *Reload opening window*. Copy this button and give the copy the title of *Copy*.



Assign the function *copyData(this)* to the *onclick* event of the button *Copy*, and write the following script:

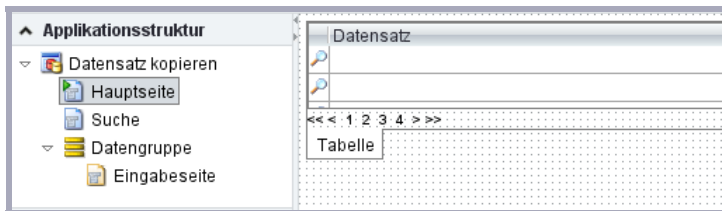
```
function copyData(p_oButton)
{
    var oFup = p_oButton.oUp.oFup;
    oFup.recId = "-1";
    return true;
}
```



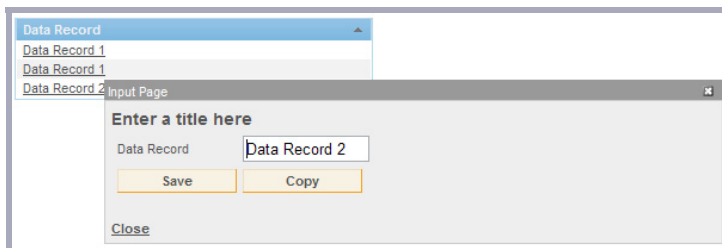
If an existing data record is now loaded in the browser, the data record ID will already be defined. For the reference of the button to the formula, *recId* will be entered as a variable. By setting the data record ID to the value *-1*, an insert procedure will be executed upon saving.

By setting the *RecId* to a specific value, a specific data record can be called up.

On the main page of the application, enter a view table for the field *Data Record* with a link to the edit page in a tooltip.



Save the application. In order to test the result, enter any number of sets via the edit page and save them. If an existing data record is selected via the view table, it can be duplicated by clicking the *Copy* button.



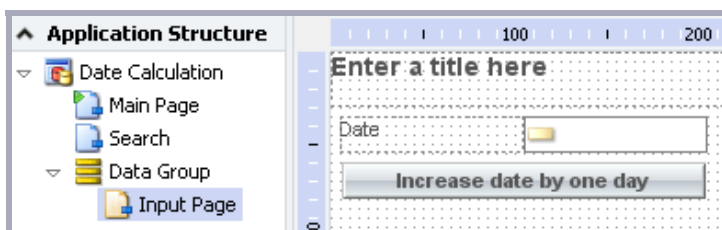
Please note that for this method, neither data records from subordinate data groups, nor files can be copied.

### 5.9. Date Calculation with JavaScript, Part I

An existing date should be increased by one day by clicking a button.

#### Exercise

Create the application *Date Calculation* based on the *Empty Application* template. On the edit page, define an edit field with the title *Date* and the data type *Date*. Open the properties dialog of the edit field and set the preset *Current date and time* on the *Options* tab. Then, create a button with the title *Increase date by one day*.



For the *onclick* event of the button, insert the following script:

```
function tomorrow()
{
    var oHtmlDate = getElement("00B8....77B1"); /*Date datecontrol*/
    var dtOldDate = oHtmlDate.oUp.setDateObjectFromLocal(oHtmlDate.value);
    var dtNewDate = dateAdd("d", 1, dtOldDate); // + 1 day from current date
    oHtmlDate.value = oHtmlDate.oUp.toLocalDateString(dtNewDate);
    return true;
}
```

With the variable *oHtmlDate*, a reference to the *Date* edit field will be constructed. The date will be converted with the method *setDateObjectFromLocal()* into a JavaScript date object, and assigned the variable *dtOldDate*. With the method *dateAdd()*, the date will be increased by one day and assigned the variable *dtNewDate*. This does not, however, suffice to show the value in the form. The formatting must be adjusted. For this, we will be served by the method *toLocalDateString()*, to which we will assign the value from *oHtmlDate* again. Upon clicking on the button in the browser, the date entered will be increased by one day.

### 5.9.1. Syntax of the Method *dateAdd()*

*dateAdd( Interval, Number, Date)*

#### Interval (Type String)

Possible values:

"ms" for milliseconds  
"s" for seconds  
"mi" for minutes  
"h" for hours  
"d" for days  
"mo" for months  
"y" for years

#### Number (Type Integer)

Number of the unit entered in *Interval*, to which the date will be added. If the interval should be subtracted, the number will be entered as a negative value (such as -1).

#### Date (Type Date)

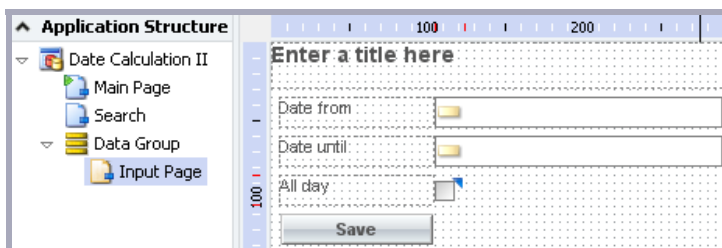
JavaScript date object, which should be taken as a basis for the calculation.

### 5.10. Date Calculation with JavaScript, Part II

An appointment should be marked as „all day“ by setting a checkbox.

#### Exercise

Create the application *Date Calculation II* based on the *Empty Application* template. On the edit page, define an edit field with the title *Date from* and the date type *Date & Time*. Repeat this procedure and create an additional field *Date until* with the data type *Date & Time*. Now we also require a *Checkbox* with the name *All day*. Then, create a button with the title *Save*, link to the *Main Page*, *Close popup/tooltip*, and *Reload opening window*.



Insert the following script for the *onclick* event of the *checkbox*.

```
function myFullDay()
{
    var oHtmlFrom = getElement("07...63"); /*Date from  datetimecontrol*/
    var oHtmlTo = getElement("67...73"); /*Date until  datetimecontrol*/
    var oHtmlCheck = getElement("AC...C1"); /*All Day checkcontrol*/

    return setFullDays(oHtmlCheck, oHtmlFrom, oHtmlTo);
}
```

With the variable *oHtmlDateFrom*, a reference to the edit field *Date from* will be created. With the variable *oHtmlDateTo*, a reference to the edit field *Date until* will be created. The variable *oHtmlcheck* displays a reference to the checkbox.

#### 5.10.1. Syntax of the Method setFullDays()

```
setFullDays(checkbox, Element1, Element2)
```

#### 5.11. Handling Tables via a JavaScript Object

With the expert attribute *jsobject*, values from columns and rows in tables can be accessed. If the expert attribute

**Name:** jsobject  
**Value:** true  
**Type:** boolean

is implemented for a view table or free layout table, an UpTable object will automatically be created. The UpTable object can be reached via the global object *oTableReg*. In addition to properties like ID and GUID of the table, it contains


- a list of the recIds of the data records shown on the browser page
- a list of the columns of the table

A column object contains properties like

- ID
- GUID
- displayName
- controlType

and offers methods for access to nodes (*<a>* or *<span>* tags) or to contents of text nodes. Here is an example for access to the object *oTableReg*:

```
oTableReg.getTableByGuid("GUID of the table").getColByGuid("GUID of the column").getNodeValue(recid)
```

 For the free table, the RecordID of the data record will be transferred here, for view tables the VelocityCounter. This method delivers the value from the column for the transferred column:

```
oTableReg.getTableByGuid("GUID of the table").aRecIds
```

An array of the RecordIDs for the page will be created here (new Array(1,2,5)).

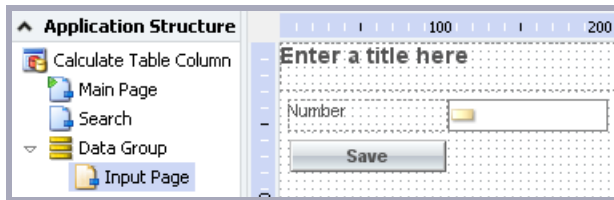
Additional methods are contained in the file *object.js*.

#### 5.11.1. Calculate Table Column

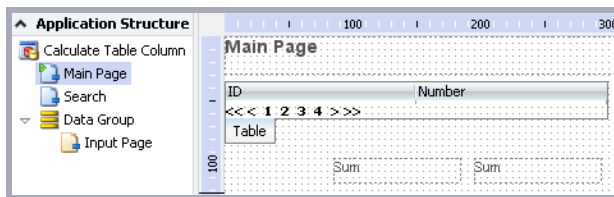
The value of a decimal number field in a table column should be summed and output in a sum field.

**Exercise**

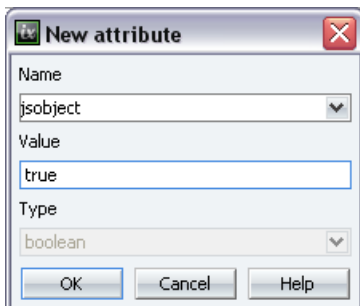
Create the application *Calculate Table Column* based on the *Empty Application* template. Define an edit field of type *Decimal Number* on the edit page, with the title *Number* and the button *Save*, with the action *Save* and link to the main page of the application.



On the main page, create a view table with the columns *Id* and *Number*. Below the table, define a view field of type *Static Text* with the title *Sum*.



Open the properties dialog of the table and set the new expert attribute here:



In the *onload* event of the main page, the following function will be called:

```
function calcTable()
{
    //Reference to the UP table object
    var oTable = oTableReg.getTableByGuid("1BB6...01C3");
    //Reference to the UP column object
    var oColumn = oTable.getColByGuid("E8DB...F9B3");

    var sum = 0;

    //Create sum of all columns
    for (var i=1; i <= oTable.aRecIds.length; i++)
    {
        var oValue = oColumn.getNodeValue(i);
        //Remove the formatting
        var jsValue = Helper.parseFloat(oValue);
        //Does jsValue contain a value???
        if ( isNaN(jsValue) == false)
            sum = sum + jsValue;
    }

    var oHtml = getElement("FBEC...3ECA"); /*Sum labelcontrol*/
    var oFloat = new upFloatControl();

    //Format and output Output
}
```

```


var output = oFloat.toLocalFormat(sum);
setTextValue(oHtml, output);
return true;
}

```

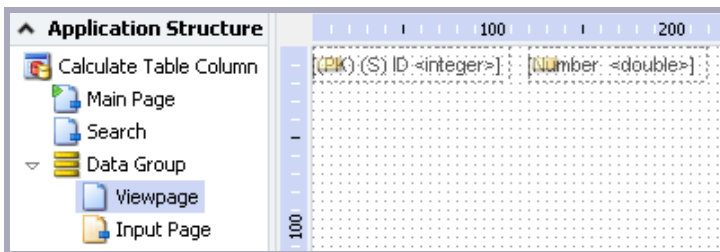
The GUID of the table column can be ascertained from the properties dialog of the view table by clicking the *Format* button for the marked field *Number*. Next, switch to the *Expert* tab and copy the GUID of the column.

Enter any number of number values in the browser on the edit page. Upon calling up the main page, the sum of the visible values will be output in the sum field.

ID ^	Number
1	1,00
2	14,00
3	12,30
Sum	27,30

 Only the values in the number column that are shown on the page will be calculated. If the table contains additional pages that are available via the navigational element at the foot of the table, these values that are not currently being shown will not flow into the calculation.

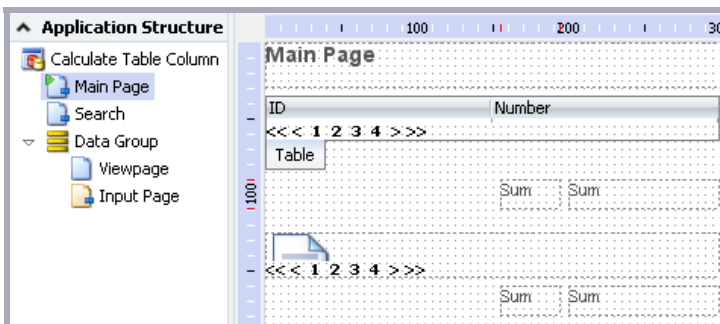
In order to calculate the sum of the column of a free layout table, define a view page with view fields for the data fields *Number* and *Id*.



On the main page, enter a free layout table for the newly created view page below the view table. Define the new expert attribute

**Name:** jsubject  
**Value:** true  
**Type:** boolean

here as well. For the output of the sum, a view field of type *Static Text* with the title *Sum* will be implemented.



For the calculation of the sum in a free layout table, the script will be expanded in the *onload* event of the main page as follows:

```
function calcTable()
{
    // Calculate sum for free table

    var oTable = oTableReg.getTableByGuid("CC2B...CDC1");
    var oColumn = oTable.getColByGuid("E550...2508");
    var oRecids = oTable.aRecIds;
    var sum = 0;

    for (i in oRecids)
    {
        var recid = oRecids[i];
        var oValue = oColumn.getNodeValue(recid);
        var jsValue = Helper.parseFloat(oValue);

        if ( isNaN(jsValue) == false)
            sum = sum + jsValue;
    }

    var oHtml = getElement("6DCC...D0A8"); /*Sum labelcontrol*/
    var oFloat = new upFloatControl();
    var output = oFloat.toLocalFormat(sum);
    setTextValue(oHtml, output);
    return true;
}
```

The GUID of the free layout table and the column to be calculated can be determined from the application structure. Show the elements of the page on which the table is placed and open the subordinate tree until you can see the element *shapedtablebase*.



When the expert options are activated you can now determine the correct GUID by pressing the key F4. As GUIDs of the several columns the GUIDs of the corresponding view elements on the view page will be used. Save the application and test it in the browser.

ID	Number
1	1,00
2	14,00
3	12,30
	Sum 27,30
1	1,00
2	14,00
3	12,30
	Sum 27,30

With the following commands, table cells can also be overwritten for output:

```
for (i in oRecids)
{
    var oNode = oColumn.getNode(i);
    setTextValue( oNode, "value");
}
```

### 5.12. Automatic Reload of a View Table

With the function *reload\_[tablerecords]()*; a view table should be refreshed in regular intervals. Assign the function to the *onload* event of the page on which the view table to be refreshed is found.

```
window.setTimeout('reload_tablerecords12...()',200); // Execute the reload
function in 200 millisecond interval...
```

Replace *tablerecords12...* with the actual name of the tablerecords in your application. The name can be ascertained, if you

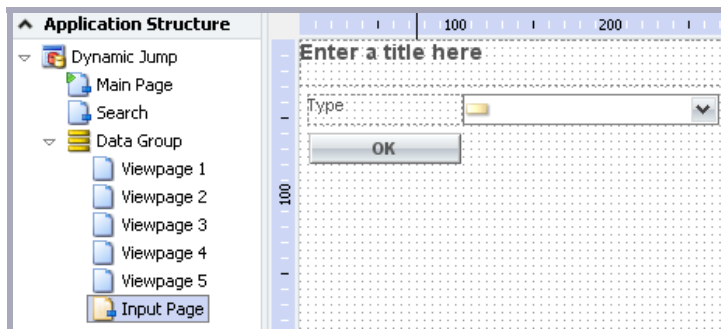
- highlight the view table,
- with the *F4* key (expert mode must be activated)
- switch to the XML dialog,
- search for the tag *<tablerecords* and the attribute *name=""*, highlight this value, and copy it.

### 5.13. Dynamic Jump from View Table

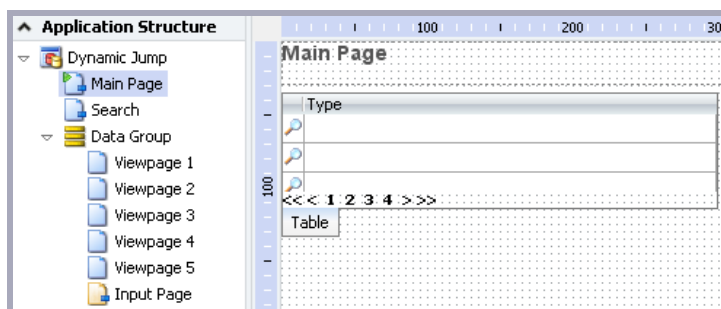
Depending on a column value in a view table, a column should jump to various view pages.

#### Exercise

Create the application *Dynamic Jump* based on the *Empty Application* template. Define a selection list on the edit page with the title *Type*, the data type *Text* and the user-defined entries *Type 1*, *Type 2*, *Type 3...* - *Type 5* and a button with the action *Save*, the title *OK*, and a jump to the main page. Underneath the data group, create 5 view pages for the various types and name them correspondingly.



Now create a view table on the main page with the columns *Type* and jump target to one of the *View Pages* in a tooltip.




Open the script editor via the menu *Edit / Edit JavaScript* and insert the following script:

```
function getTypeandJump(oHtml, p_strType)
{
    if(p_strType == "Type 1")
        oHtml.oUp.oTarget.rq_TargetGuid = "7069A...0A6A9";
    else if(p_strType == "Type 2")
        oHtml.oUp.oTarget.rq_TargetGuid = "85D3B...37511";
    else if(p_strType == "Type 3")
        oHtml.oUp.oTarget.rq_TargetGuid = "4BEF...E52FD1";
    else if(p_strType == "Type 4")
        oHtml.oUp.oTarget.rq_TargetGuid = "7ED6...E8255";
    else if(p_strType == "Type 5")
        oHtml.oUp.oTarget.rq_TargetGuid = "B473A...110D4";
}
```

```

    return true;
}

```

 Please note that the values to be checked in the JavaScript function must be the same as the values of the *Type* selection list.

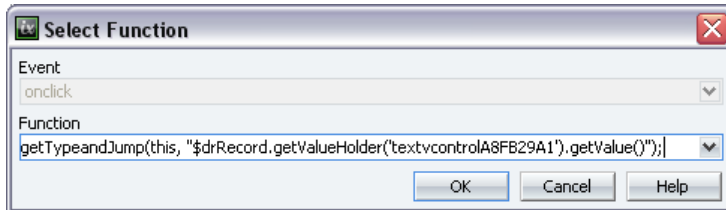
Replace the individual GUIDs with the actual GUIDs of the *View Pages 1-5*. These can be found in the details dialog of each page.

Assign the function to the *onclick* event of the column that is defined as jump as follows:

```

getTypeandJump( this, "$drRecord.getValueHolder('textvcontro...').getValue()" );

```




In the value of the first entry, replace *textvcontrolD87AD347* with the actual name of the *Type* column. This can be ascertained in the properties dialog of the column on the *Expert* tab.

Save the application. In order to test the results, enter 5 types via the edit page and save it. If an existing data record is selected from the view table, the JavaScript function will be linked to the view page belonging to it.

## 6. Velocity

### 6.1. What is Velocity?

Velocity is an Open Source project of the Jakarta project group of Apache. Velocity is a Java-based template engine. It enables the direct call of Java classes from websites. VTL is the abbreviation of *Velocity Template Language*.

 VTL references begin with the character \$ and will be used for requests (such as variable definitions or class requests that return data). VTL commands begin with the character # and will be used for the execution.

### 6.2. Where will Velocity be used?

Velocity calls can be exclusively processed on VM pages. Velocity commands can be directly entered to a static VM or integrated via the XSL into dynamic pages.

The Application Designer offers the possibility of editing Velocity commands in

- Expert mode or
- in VTL controls.

In addition, the properties dialogs of the elements in the elements in the Application Designer allow information to be queried from Java classes (such as information about the current user or parameters from the request object).

### 6.3. References

#### 6.3.1. Variables

```

$foo
$a

#set( $a = "Velocity" )
$a is the name of the variable, Velocity the value.

```

```

Beispiel
<html>
<body>#set( $foo = "Velocity" )
        Hello $foo World!
</body>
</html>

```

### 6.3.2. Properties

```

$customer.Address
$purchase.Total

```

### 6.3.3. Methods

```

customer.getAddress()
$purchase.setTitle("My Home Page")
$person.setAttributes(["large", "small", "medium"])

```

#### Formal Reference Notation

```

${customer.address}

```

The formal notation is necessary for the following text:

This is a `${type}`case. --- `${type}` will now be handled as a variable and replaced.

#### Silent Reference Notation

When Velocity encounters an undefined .Reference, the name of the reference will simply be output.

Example:

```

<input type="text" name="email" value="$email"/>

```

If `$email` has no value, the *value* would also be better to remain empty. This can be achieved with the following notation: `<input type="text" name="email" value="$!email"/>`

Combined with the formal notation, the following notation is also possible:

```

<input type="text" name="email" value="${!email}"/>

```

#### Escaping References

```

#set($email = "foo")
$email
\$email
\\$email
\\\ $email

```

The output would look like the following:

```

foo
$email
\foo
\$email

```

If `$email` is not defined:

```

$email
\$email
\\$email
\\\ $email

```

### 6.4. Comments

Comments are not visible in the results file.

```

## This is a one-line comment.
#*
This is a multiline comment, which will not be shown on the web.
*#

```

**6.5. #set**

The #set command assigns a value to a reference.

```
#set ( $property = "friendly" )
#set ( $customer.behavior = $property )
```

A variable or property reference must be entered to the left.

On the right-hand side, the following is possible:

- Variables reference
- Character string
- Property reference
- Methods reference
- Number
- ArrayList
- Map

Example for number: #set ( \$number = 123 )  
 ArrayList : #set ( \$array = [ "Not", \$my , "fault" ] )  
 Map : #set ( \$map = { "banana" : "good", "roast beef" : "bad" } )

Hint: access to the ArrayList via  
       \$array.get(index), i.e. \$array.get(0) returns the value 'Not'.  
 Access to the map via  
       \$map.get(name), i.e. \$map.get("banana") returns the value 'good'.

Calculations may exist to the right:

```
#set ( $value = $foo + 4 )
```

The return value null does **not** overwrite the contents of a reference, i.e.

```
#set ( $result = $query.criteria("name" ) )
#set ( $result = $query.criteria("address" ) )
```

If \$query.criteria("address") returns null as a value, \$result will retain the value from the first #set, meaning the value that \$query.criteria("name") returned, such as Maier.

Example: for a for-each, the following should be used because of this

```
#set( $criteria = ["name", "address" ] )
#foreach( $criterion in $criteria )
    #set( $result = false )
    #set( $result = $query.criteria($criterion) )
    #if( $result )
        Query was successful
    #end
#end
```

**For String Literals, the Following Applies:**

Using double quote marks ""

```
#set( $directoryRoot = "www" )
#set( $templateName = "index.vm" )
#set( $template = "$directoryRoot/$templateName" )
$template
```

The output is:  
 www/index.vm

Using simple apostrophes , ' – no parsing will occur, meaning that references will not be canceled.

```
#set( $foo = "bar" )
```

```
$foo
#set( $blargh = '$foo' )
$blargh
```

The output is:

```
bar
$foo
```

## 6.6. #if / #else / #elseif

### 6.6.1. #if

```
#if( $foo )
  <strong>Velocity!</strong>
#end
```

### 6.6.2. #else

```
#set ( $foo = "deoxyribonucleic acid" )
#set ( $bar = "ribonucleic acid" )

#if ( $foo == $bar )
  In this case it's clear they aren't equivalent. So...
#else
  They are not equivalent and this will be the output.
#end
```

### 6.6.3. #elseif

```
#if( $foo < 10 )
  <strong>Go North</strong>
#elseif( $foo == 10 )
  <strong>Go East</strong>
#elseif( $bar == 6 )
  <strong>Go South</strong>
#else
  <strong>Go West</strong>
#end
```

## 6.7. Relational and Logical Operators

Conditions

Equivalence operator: #if( \$foo == \$bar )

Greater than: #if( \$foo > 42 )

Lesser than: #if( \$foo < 42 )

Greater than or equal: #if( \$foo >= 42 )

Lesser than or equal: #if( \$foo <= 42 )

Same number: #if( \$foo == 42 )

Same character string: #if( \$foo == "bar" )

Logical NOT: #if( !\$foo )

### logical AND

```
#if( $foo && $bar )
  <strong> This AND that</strong>
#end
```

### logical OR

```
#if( $foo || $bar )
  <strong>This OR That</strong>
#end
```

### logical NOT

```
#if( !$foo )
```

```
<strong>NOT that</strong>
#end
```

**6.8. #foreach**

ArrayList:

```
#set ( $allProducts = [ "Glove", "Hat", "Boot" ] )
<ul>
#foreach( $product in $allProducts )
  <li>$product</li>
#end
</ul>
```

Automatic number variable: \$velocityCount (begins with 1)

Map:

```
#set ( $allProducts = { "banana" : "good", "roast beef" : "bad" } )
<ul>
#foreach( $key in $allProducts.keySet() )
  <li>Key: $key -> Value: $allProducts.get($key)</li>
#end
</ul>
```

**6.9. #include**

Allows the import of a local file.

```
#include( "one.txt" )
```

**6.10. #parse**

Allows the execution of a Velocity file.

```
#parse ( "static.vm" )
#parse ( $varfürDateiname )
```

**6.11. #macro**

Allows the definition of VTL partial sections that can be repeated any number of times.

**Defining the Macro**

```
#macro( d )
<tr><td></td></tr>
#end
```

**Calling the Macro**

```
#d()
```

**Defining the Macro with Parameters**

```
#macro( tablerows $color $somelist )
#foreach( $something in $somelist )
  <tr><td bgcolor=$color>$something</td></tr>
#end
#end
```

**Calling the Macro with Parameters**

```
#set( $greatlakes = [ "Superior", "Michigan", "Huron", "Erie", "Ontario" ] )
#set( $color = "blue" )
<table>
  #tablerows( $color $greatlakes )
</table>
```

**Results**

```
<table>
  <tr><td bgcolor="blue">Superior</td></tr>
```

```

<tr><td bgcolor="blue">Michigan</td></tr>
<tr><td bgcolor="blue">Huron</td></tr>
<tr><td bgcolor="blue">Erie</td></tr>
<tr><td bgcolor="blue">Ontario</td></tr>
</table>

```

**6.12. Range Operator**

```

[n..m]

#foreach( $foo in [1..5] )
$foo
#end

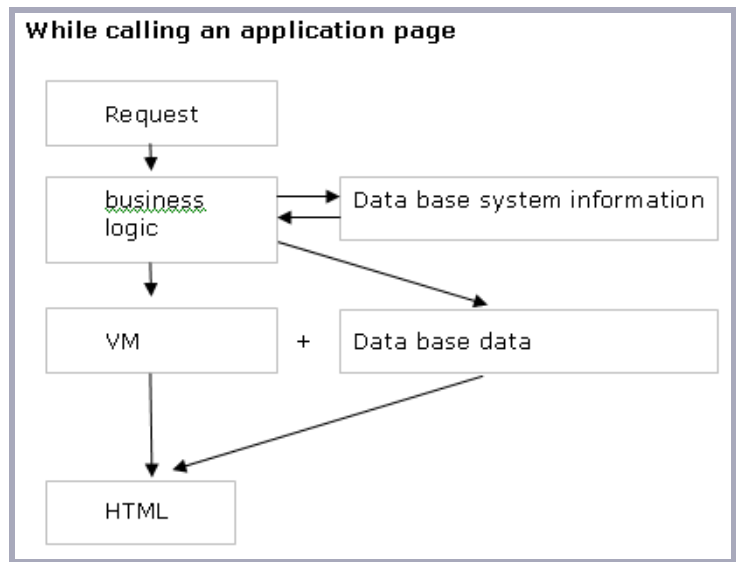
#foreach( $bar in [2..-2] )
$bar
#end

#set( $arr = [0..1] )
#foreach( $i in $arr )
$i
#end

```

**7. Calling a Page (2nd Transformation)**

When calling an application page



**8. Context Objects in Velocity**


The `$User` object provides all information for use that affects the currently logged in user. Sample call: `$User.getLoginName()`

**8.1. \$User**

The `$User-Object` provides all information for use that refers to the currently logged on user. The following methods are available:

Method	Information about current user
<code>int getId()</code>	ID
<code>java.lang.String getGuid()</code>	GUID
<code>java.lang.String getEmployeeNo()</code>	Personnel number
<code>java.lang.String getFirstName()</code>	First name
<code>String get2ndName()</code>	Middle name
<code>java.lang.String getTitle()</code>	Title
<code>java.lang.String getFullName()</code>	First and last name
<code>java.lang.String getLastName()</code>	Last name
<code>java.lang.String getStreet()</code>	Street

java.lang.String getCountry()	Country
java.lang.String getZipCode()	Postal code
java.lang.String getCity()	City
java.lang.String getEmailBiz()	eMail – business
java.lang.String getPhoneBiz()	Telephone – business
java.lang.String getPhoneMobileBiz()	Mobile telephone – business
java.lang.String getPhoneFax()	Fax – business
java.lang.String getEmailHome()	eMail – private
java.lang.String getPhoneHome()	Telephone – private
java.lang.String getPhoneMobileHome()	Mobile telephone – private
java.lang.String getDescription()	Description
java.util.Date getBirthday	Birthdate
java.util.Date getEnterDate()	Date of employment start
int getGender()	Gender
byte[] getImageData()	ID photo
java.lang.String getLoginName()	Login name (without domain)
java.lang.String getQualifiedLoginName()	Login name with domain
Java.lang.String getLoginDomain()	Domin
Java.util.TimeZone getTimeZone()	Time zone
int getBoss()	Manager (ID)
Java.util.Map getCustomMapVH	Returns a map of additional fields
Java.lang.String getPhonePager()	Pager
boolean isAnonymous()	Login status
boolean isDisabled()	Account disabled
Java.lang.String getExternalLogin(0)	Login name 1 under external logins
Java.lang.String getExternalPassword(0)	Password 1 under external logins

 With the method `$User.getLoginName()`, the login name of the current user will be read out. Additional fields can be read out with the method `getCustomMapVH()`, whereby `XX1` is the database field name of the additional field: `$User.getCustomMapVH().getValue("XX1").getValue()`

The database field name of the additional field can be ascertained from the *User* application if it is open in the Application Designer. By right-clicking on the *User* data group, all data fields can be shown via the context menu.

## 8.2. \$Request

The request object is a Java object. It contains all data that are required for request processing. For a request to the server (such as upon clicking on a button), this object will be created in the method `processRequest()`, filled with the required data (such as the values of the controls that are to be saved) and sent to the server. The method `processRequest()` is contained in the file `object.js`. With this object created in such a way, values can also be sent from one page to the next, which will then be queried via Velocity and can be further processed. The following methods are available in the request for use:

**java.lang.String \$Request.put("rq\_myParam", "value")**

This method transfers a value to the request object, here with the name `rq_myParam`.

**java.lang.String \$Request.get("rq\_myParam")**

This method returns the value that has been saved under the name `rq_myParam`.

The values from edit controls will be saved under the name of the edit control. With this, the values can be queried on the following page under this name (see chapter *JavaScript / Automatic Update*).

## 8.3. \$Loader (BusLogicCaller)

With the *DataCollection*, the following values can be queried. The corresponding business logic object can be addressed within Velocity as follows:

`$Loader.getDataCollection()`

Or in short:

`$DC`

Method	Information
IDataRange getDataRange()	Data range object for the corresponding control
IDataRange getDataRange().getQuery	SQL-Select, which for the construction of the data range object was set for the corresponding control.
IDataRange getDataRangeBySysident()	Data range object for the corresponding control
java.lang.String getParentId()	Returns the ID of the parent set (ParentID). If no ParentID has been set, the value -1 will be returned.
java.util.Map.getPropertiesVH()	Properties from property data group
java.lang.String getRecId()	Returns the current ID of the data record
java.lang.String getUserId()	Returns the ID of the user (UserID) of the data record, set by the field that defined the sysident <i>UserID</i> .
IValueHolder getValueHolder()	Returns the valueholder of the control with the name entered in <i>Controlname</i> .
IValueHolder getValueHolderBySysident()	
java.lang.String getValueStrByFieldGuid()	
java.lang.String getValueStrBySysident()	

`$Row in DC().getDataRange(Name of Control)`

`$Row.getAttribute('Column Name')`

`$Row.getRecId()`

List element for each line: contents of a column in the datarange object

### Velocity Debug Class

#### **\$DEBUG.inspect( \$reference )**

Overview of methods and properties of the object transferred as parameter.

#### 8.4. **\$null**

Has the value null.

#### 8.5. **\$charset**

Contains the value of the currently valid character set. This value will also be written to the HTTP header.

#### 8.6. **\$lang**

Contains the language identifier, as it was determined at the start of the request processing from the query string, cookies, and the system settings (de.uplanet.lucy.util.UpLocaleFactory).

#### 8.7. **\$UriBuilder**

Class in order to construct Intrex URLs. de.uplanet.lucy.server.composer.UriBuilder

#### 8.8. **\$ObjectHelper**

Helper class. Is currently used to check objects for nulls. de.uplanet.lucy.server.auxiliaries.ObjectHelper

#### 8.9. **\$Factory**

Factory object for the creation of objects that do not need to exist in every Velocity context. Mainly to be used to improve performance and resource consumption on the server.

de.uplanet.lucy.server.auxiliaries.ObjectFactory

#### 8.10. **\$Response**

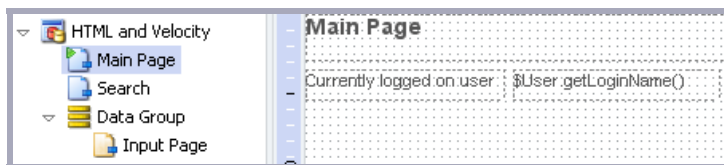
The response object, similar to HttpServletResponse.

de.uplanet.lucy.server.connector.web.HttpResponseWrapper

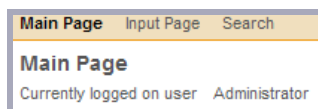
- 8.11. **\$Portal**  
The global web application object (corresponds to the application object in ASP).  
de.uplanet.lucy.server.auxiliaries.Portal.
- 8.12. **\$Session**  
The current session.  
de.uplanet.lucy.server.session.ISession
- 8.13. **\$DbConnection**  
The database connection assigned to the context  
de.uplanet.jdbc.JdbcConnection.
- 8.14. **\$Request**  
The request object.  
de.uplanet.lucy.server.connector.IServerBridgeRequest
- 8.15. **\$Loader**  
de.uplanet.lucy.server.composer.BuslogicCaller
- 8.16. **\$Chat**  
Proxy object fo chat and notes.  
de.uplanet.lucy.server.auxiliaries.ChatProxy
- 8.17. **\$OMUC**  
Wrapper class for various usage cases of the organizational management.  
de.uplanet.lucy.server.auxiliaries.UserManagerWrapper
- 8.18. **\$VelocityContext**  
The Velocity context object (as initialization parameter for other objects in the context).  
org.apache.velocity.VelocityContext
- 9. **Direct Velocity Call**  
Velocity commands can be called up directly in a view control. Via a direct call, the login name of the user should be output.

**Exercise**

Create the application *HTML and Velocity* based on the *Empty Application* template. On the main page, create a *Static Text* view element and set the option *Only default language (such as for programming purposes)* on the *Options* tab. In the title of the view element, enter the Velocity command `$User.getLoginName()`. From the context menu item *Create Title*, create for the view element the title *Currently logged on user*.



The Velocity code contained in the view element will be executed and shows the login name of the currently logged in user in the browser.



- 10. **Static VMs in Velocity**  
With the view element *VTL Include*, static VMs can be directly called up. Static VMs do not run through the XML/XSL transformation. They will be directly integrated into the VM.



The VM can be entered in the properties dialog with a path. The table example above can also realize a *VTL Include*.

## 11. Query from Velocity

### 11.1. Example for PreparedQuery

As an example for PreparedQuery, you see here the *sample.vm* from the directory `[xtreme]\org\[portal name]\internal\system\vm\html\include`.

```
##this is a sample file for the integration of vm code to an application
##you can easily add a VTL Include to your Application, if you install the
VTL Include Patch available from United Planet

#####
##After you find a sample to connect to a database and get the results as an
iterator object
#####

#####
## output macro
#####

#macro(output_data)

<td nowrap class="SCUP_Datarange_RowText_BG">
  <span class="SCUP_Datarange_RowText_normal">
    $!l_row.getValueHolder($!l_counter).getValue()
  </span>
</td>
#set($!l_counter = $!l_counter+1)
#if($!l_counter <= $!l_arglength)
  #output_data()
#end
#end

#####
##define your query and execute it
#####

#set($!l_stmt = $PreparedQuery.prepare($DbConnection, "SELECT LID as
id,STRLOGIN AS Login,STRFULLNAME as Name,STRMAILBIZ as Mail FROM DSUSER"))
#set($!l_rs = $!l_stmt.executeQuery())

#####
## write the header to the html output
#####
<table cellpadding="0" cellspacing="0"
class="SCUP_Datarange_Container_normal_BG" style="empty-cells: show;">
  <tr>
    <td nowrap class="SCUP_Datarange_Header_BG" align="right"><span
class="SCUP_Datarange_Header">ID</span></td>
    <td nowrap class="SCUP_Datarange_Header_BG" align="right"><span
class="SCUP_Datarange_Header">Login</span></td>
    <td nowrap class="SCUP_Datarange_Header_BG" align="right"><span
class="SCUP_Datarange_Header">Name</span></td>
    <td nowrap class="SCUP_Datarange_Header_BG" align="right"><span
class="SCUP_Datarange_Header">Mail</span></td>
  </tr>
```

```
#####
## get the content
#####
    #foreach($l_row in $l_rs.iterator())
        #set($l_arglength = $l_row.size())
        #set($l_counter = 1 )
        #if($l_even)
            #set($l_even = false)
            #set($l_style="SCUP_Datarange_RowOdd_normal_BG")
        #else
            #set($l_even = true)
            #set($l_style="SCUP_Datarange_RowEven_normal_BG")
        #end
        <tr class="$l_style">
            #output_data()
        </tr>
    #end
</table>

#####
##close the recordset
#####
$l_rs.close()
$l_stmt.close()
```

## 11.2. Simple Database Query

The following example executes a query on the user table and delivers as return value a resultset with all entries of this table. The query will, in this example, be executed without parameters within the statement.

```
#set($l_statement = $PreparedQuery.prepare($DbConnection, "SELECT
STREMPLOYEENO AS No, STRFULLNAME as Name, STRPHONEBIZ as Tel, STRMAILBIZ as
Mail FROM dsuser"))
```

Creates the statement; in this case, to the system database (*\$DbConnection*).

```
#set($l_resultset = $l_statement.executeQuery())
```

Runs the query and gives back a resultset as return value.

```
<table>
<tr>
    <th>Number</th><th>Full Name</th><th>Tel.</th><th>Mail</th>
</tr>

#foreach($l_row in $l_resultset.iterator())
Loop through all data records of the resultset.
    <tr>
        <td>${!l_row.getValueHolder(1).getValue()}</td>
        <td>${!l_row.getValueHolder(2).getValue()}</td>
        <td>${!l_row.getValueHolder(3).getValue()}</td>
        <td>${!l_row.getValueHolder(4).getValue()}</td>
```

The return value of the methods *getValueHolder(int p\_idx)* are ValueHolders, which can be brought via renderers into a formatted output format. In this example, for clarification purposes we have chosen not to, and called up the method *getValue()* of the ValueHolder object. This method returns a string with the lexicographical correspondent to the data type.

```
    </tr>
#end
</table>

$l_resultset.close()
$l_statement.close()
```

## 12. Velocity and Parameters

### Exercise

In the User Manager, under the menu item *User / Schema Manager*, the additional field *Country Code* will be defined.

For each user, one of the three country codes *de*, *en*, or *fr* will be entered.

Attribute	Type	Size	Value
Country Codes	string	2	en

Now, enter two test customers into the application *HTML and Velocity* via the edit page. For one customer, enter the country code *de*, and for the other, the country code *en*.

Customer No.	Company	Country Code
10000	ABC Engineering Ltd.	en
10001	Polygram Inc.	de

In a VM file, the SQL query that filters by country code of the current user will be run. Switch to the directory

`xtreme\org\[portal name]\internal\system\vm\html\include` and create a copy of the file *sample.vm*. Rename the VM to *countrycodes.vm*. Open the file and search for the comment *define your query and execute it*. Enter the following code into this section:

```
## country codes
#set($lkz = $User.getCustomMapVH().get("STR_LAND").getValue())
#set($l_stmt = $PreparedQuery.prepare($l_conn, " SELECT
T0.L_CUSTOMERNONINTEGER_32334E86 as CNo,
T0.STR_COMPANYSHORTTEXT_586A0F18 as Company,
T0.STR_COUNTRYCODE_BE378880 as Countrycode FROM datagroupD349D2AE T0
WHERE T0.STR_COUNTRYCODE_BE378880='$lkz' ORDER BY
T0.STR_COMPANYSHORTTEXT_586A0F18 ASC"))
$l_stmt.setString(1, $lkz)
```

```
#set($l_rs = $l_stmt.executeUpdate())

$l_rs.close()
$l_stmt.close()
```

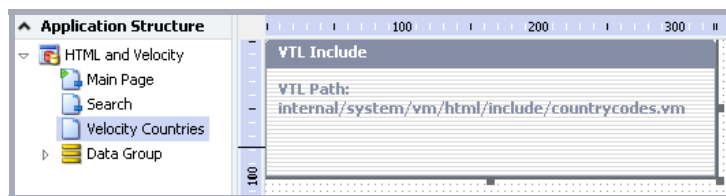
### #set(\$lkz = \$User.getCustomMapVH().get("XX2").getValue())

Replace the entry *XX2* as required with the correct column name of the additional field from the User Manager. Additional fields will be named upon creation in increasing order with *XX1*, *XX2*...*XXn*. The column name can be determined at the database level in the table with *OMUSER*.

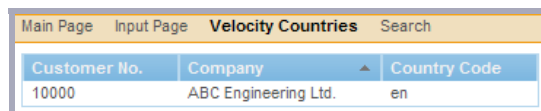
### #set (\$l\_stmt = \$PreparedQuery.prepare(\$l\_conn, " SELECT...)

Replace the data field names entered here as required, and the name of the data group with the current name from your application. The field names can be found in the case of an open application via the context menu item *Show data fields* for the data group *Customers*. Select the corresponding data field in the application structure. With the *F4* key, the name can be copied and inserted from the *Details* dialog.

Create the new view page *Velocity Countries* in the application. On the page, a view element *VTL Include* will be connected to the new VM *countrycodes.vm*.



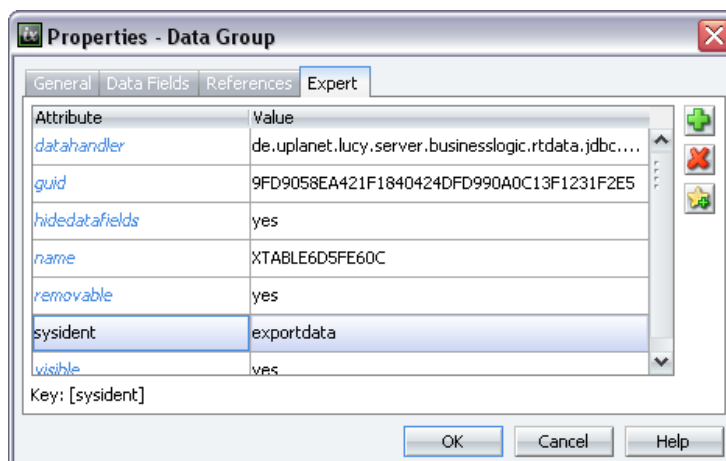
Add the new view page to the application menu, save the application, and log in to the portal with one of the users for whom you have assigned a language abbreviation. Test the results in the browser.



## 12.1. Variably Querying Table Names

When copying the *File Export* application, the table name will change. In order for the same VM file to be able to be used, the table name can be determined with the following procedure:

In the Application Designer, a value will be assigned to the expert attribute *sysident* via the properties dialog of the data group (such as *exportdata*).



With the following select statement, the table name can be ascertained:

```
#set ($l_strCardTablename =
$AppWalker.getTableByNameBySysident($Request.get('rq_AppGuid'),
'exportdaten'))
```

On the view page *Table Name*, the select statement will be inserted into a view field of type *static text* as its title. On the *Options* tab of the properties dialog, the option *Only default language (such as for programming)* will be set.

In the browser, the current table name of the data group will be output.

### 13. Buttons

Buttons execute actions in Intrex Xtreme, such as saving, deleting, changing, and selecting data records, loading other pages, sending eMails, etc. The display in the browser is possible as

<b>Button</b>	<i>(upButtonControl)</i>
<b>Image</b>	<i>(upImageActionControl)</i>
<b>Text</b>	<i>(upTextActionControl)</i>

This means that the functionality, regardless of the display of the button, is always the same. An *upTextActionControl* has, therefore, the same action as an *upButtonControl*. An action control contains an instance of the *upSource* object and an instance of the *upTarget* object. In these instances, the relevant information is contained for the request. The *upSource* object defines properties of the source page of the request. The *upTarget* object defines the requirements of the next page to be loaded, such as which page should be loaded with which data record.

All requests that will be sent to the server will be assembled in the method *processRequest()*, from the file *Object.js*.

#### Example: Button type button, save, and jump to another page

```
<input id="ID_buttoncontrol1" name="buttoncontrol1" type="submit" style=""
class="SCUP_Button_Standard_normal"
onmouseover="if(this.oUp){this.oUp.setStatus(true);}return true;"
onmouseout="if(this.oUp){this.oUp.setStatus(false);}return true;"
value="Save">

<script type="text/javascript">
obuttoncontrol1 = new upButtonControl();
obuttoncontrol1 = obuttoncontrol1.biDirectUpHtml(obuttoncontrol1,
'ID_buttoncontrol1');
obuttoncontrol1.description = "OK";
obuttoncontrol1.oFup = oeditpageB598206B;
obuttoncontrol1.linkType = "16";
obuttoncontrol1.oSource = new upSource();
obuttoncontrol1.oSource.fr_ActionId = "1";

obuttoncontrol1.oTarget = new upTarget();
obuttoncontrol1.oTarget.rq_TargetId = "1000";
obuttoncontrol1.oTarget.rq_AppId = "1013";
obuttoncontrol1.userName = "Save";
</script>

<script type="text/javascript">
obuttoncontrol1.oHtml.onclick = onclickHandlerbuttoncontrol1;
function onclickHandlerbuttoncontrol1(e)
{
var rv = true;
this.oUp.processRequest(e);
if(!bProcessFormAction)return false;
}
</script>
```

**Example: Button type text, login**

```

<a id="id_textactioncontrollogin" href="#"
class="SCUP_Login_Standard" onMouseover="this.oUp.setStatus(true);return
true;" onMouseout="this.oUp.setStatus(false);return true;">Text</a>

<script type="text/javascript" language="javascript">
    otextactioncontrollogin = new upTextActionControl();
    otextactioncontrollogin =
otextactioncontrollogin.bIDirectUpHtml(otextactioncontrollogin,
'ID_textactioncontrollogin');
    otextactioncontrollogin.linkType = "14";
    //Ex.: for submit action, which form should be submit
    //otextactioncontrollogin.oHtml.form = oformgroup7f27ab17.oHtml;
    otextactioncontrollogin.oTarget = new upTarget();
    otextactioncontrollogin.oTarget.newWindow = "1";
    otextactioncontrollogin.oTarget.rq_TargetFrame = "wndLogin";
    otextactioncontrollogin.oTarget.rq_Template =
"internal/system/vm/html/login/login.vm";

    otextactioncontrollogin.oHtml.onclick =
onclickHandlerotextactioncontrollogin;
    function onclickHandlerotextactioncontrollogin(e)
    {
        this.oUp.processRequest(e);
        return false;
    }
</script>

```

**13.1. Properties and Methods of the Action Control Object**

Parameter	Values	Meaning	Attribute	Mand-atory
<b>General</b>				
.oMyAction	new upTextActionControl();	Define TextActionControl Object		yes
.oMyAction = .oMyAction.bIDirectUpHtml(oMyAction, 'ID_button');		Create relation to HTML Element (bidirectional association)		yes
.description	= description	Text for status line and tooltip		
.linkType	See values list			
.oHtml.form	"[FORM NAME]"	Reference to HTML form (for button, already exists)		
.userName	Name of user	Display name for the user, such as when validating the form in error messages.		
.oHtml.onclick	onclickHandlerbutton	Handler that should be called when clicking on the control		
<b>oTarget</b>				
.oTarget	new upTarget();	Create target object		
.oTarget.newWindow	"1" / "0"	Open new window	New window	
.oTarget.rq_Template	VM-File, if the target is a static VM			
.oTarget.addParam	Helper.setQsValueByParam("Name", "value", oMyAction.oTarget.addParam);	Set additional request parameter		
.oTarget.windowSettings	"400,300,0,0,1";	For a new window: set size, transmit Fupid of the requested window (width, height, offset x, offset y, boolean positioning on ActionControl)		yes, for popups
.oTarget.props	"dependent=yes,menubar=no,toolbar=no,scrollbars=yes,resizable=yes"	For a new window: properties		
.oTarget.rq_TargetId	FupId of target, when no rq_Template			
.oTarget.rq_AppId	AppId of target, when no rq_Template			

.oTarget.rq_ReclId	Data record to be loaded			
.oTarget.rq_SId	SessionId, if cookies have been deactivated			
.oTarget.rq_ClientType	Currently only <i>html</i>	Type of client: determines which VM processes the data		
.oTarget.rq_Lang	Language code according to ISO	Controls language to be output		
.oTarget.rq_TargetFrame	Frame Name in which target should be opened			
<b>oSource</b>				
.oSource.fr_ActionId	See values list			
.oSource.rq_SourceId	FupId of current page			
.oSource.rq_SourceAppId	AppId of current page			
.oSource.rq_SourceReclId	RecordId of current page			
.oSource.rq_SourceParentId	Data record ID of superordinate data record			

### 13.2. LinkType

Value	Description
0	Jump to existing data record
1	Request an external URL; in rq_TargetUrl, the defined URL will be written
2	Jump to a page of the parent data group OR save a new data record with jump to the same data record
4	Switch language (changeLang () instead of processRequest())
5	Initiate datapicker
6	Add new data record from popup
7	Open Datarange in new popup
8	Datepicker
9	Jump to new data record
10	Text export from datarange
11	eMail from Datarange
12	Download from FileControls / ImageFileControls (still active?)
13	Categories selection
14	Login
15	Flip Flop
16	Jump to main page

### 13.3. ActionId

Value	Description
1	Update / Insert (depends on whether recld = -1)
2	Delete

## 14. Ajax

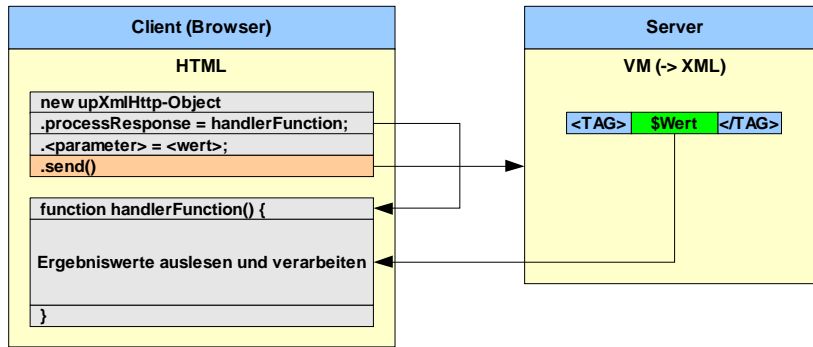
### 14.1. What is Ajax?

**Ajax** stands for „Asynchronous JavaScript and XML“, and means the asynchronous communication between the server and the browser. It is primarily used to create more performance in the portal, as well as to make the user interface for the user more intuitively constructed. Performance means that unnecessary reloads of entire pages will be reduced, and data will be queried in a targeted way in the background and directly changed at the affected area (DHTML). The realization of “Rich Applications”, meaning here web-based applications, which now acts as a compiled, locally installed piece of software (such as office software, groupware, ...).

### 14.2. Ajax in Intrex Xtreme

For the upXmlHttp object, a new instance must first be create at the appropriate Intrex page. For this object instance, various parameters will now be defined, including the function, which reads out the reply to the request and processes the VM file as well, which handles the request on the server side and prepares the results in XML format.

If the server delivers a reply to the request, the process-response event will be executed in the browser and will run the handler function. Using the oRequest object, the results values can now be read out and processed.



**i** In Internet Explorer 6, the XmlHttpRequest object will be realized via an ActiveX control. For this reason, this Ajax functionality is only usable with "activated" ActiveX. As of version 7, this is no longer the case.

### 14.2.1. The upXmlHttp Object

The following table shows all properties and methods of the upXmlHttp object.

Method/Property	Description
.bProcessResponse = <true/false>	Initialization
.processResponse = <Handler function>	Assignment of the handler function, which will be processed when initiating the request-reply process.
.bAsync = <true/false>	Asynchronous communication (default = true) Note: for false = synchronous communication, endless loops are possible (browser freeze)
.strMethod	Sender method: „POST“, „GET“
.bAddFormData	
.strURL	URL for VM file that will process the request
.send()	Method for sending the request

Example:

```
var meinXmlHttp = new upXmlHttp;
meinXmlHttp.bProcessResponse = true;
meinXmlHttp.processResponse = myHandler;
meinXmlHttp.bAsync = true;
meinXmlHttp.strURL = "";
meinXmlHttp.send();
```

### 14.2.2. ActionControls

Executing an Ajax request can also take place over ActionControls in order to transmit values via request parameters to the Velocity file. In this, the values can be correspondingly processed.

```
var oAction = new upActionControl();
oAction.oHtml = new Object();
oAction.oTarget = new upTarget();
oAction.oTarget.rq_Template =
"internal/system/vm/html/chat/getnotifymessage.vm";
oAction.requestType = 2;
oAction.oXmlHttp.bAsync = true;
oAction.oXmlHttp.bProcessResponse = true;
oAction.oXmlHttp.processResponse = processPull;
oAction.processRequest();
```

RequestType	Description
2	xmlHttp "Get"
3	xmlHttp "Post"

### 14.2.3. Request Processing

Processing a request takes place with the help of a Velocity file. This can ascertain and prepare information, for example, via a prepared query. In this it is important that the Velocity file renders and prepares the result in XML format. To do this, the header must be correspondingly generated. The tags for the return parameters are to be combined using a superior tag with the name <response>.

```

$Response.setIgnoreWrite(true)

:
: Here, with Velocity and Java classes, the results will be determined
: and/or actions run!
:

$Response.setIgnoreWrite(false)
$Response.setHeader("Cache-Control", "no-cache")
$Response.setHeader("Content-Type", "text/xml; charset=$charset")
$Response.setIgnoreWrite(false)<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>

<response>
:
  <tag>$Variable</tag>
  : Results always in XML tags
:
</response>

<response>

<tag>Entry1</tag> -> reference.getElementsByTagName('tag-
name')[0].firstChild.data;
<tag>Entry2</tag> -> reference.getElementsByTagName('tag-
name')[1].firstChild.data;

<tag>$ArrayToString<tag>

```

### 14.2.4. oRequest Object

The oRequest object makes possible the reading and/or evaluation of results of a request. With this object, in the first step the document that contains the result of the request query can be referenced (XML from VM). Next, using the DOM, the individual contents can be accessed via TAG.

Property	Description
.responseXML.documentElement	Reference to the XML document. Further reading via the DOM (following properties)
.responseText	XML reply
.readyState	= 4 (complete), 1,2,3 (incomplete)
.status	= 200 (ok), <> 200 error
.statusText	Status text (for status <> 200 = error message)

### 14.3. Returning the Ajax and XML Response

#### 14.3.1. Returning Values

##### Exercise

Create a new application based on the template *Empty Application*, then rename it to *Ajax and XML Response*. Insert a *static view element* on the main page and delete the title. Change the display to *HTML* (*Options* tab) and select the style class *Standard* from the *View* tab.

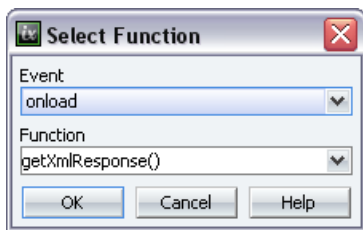
Create a button with the title *Output* without an action, and switch to the script editor:

```
function getXmlResponse()
{
    /*
    * Generate new object of type upSimpleAjax
    */
    var mySimpleAjax = new upSimpleAjax();

    mySimpleAjax.oProcessFunc = myXmlHandler1; // ResponseHandler

    /*
    * Sends a request to a given vtl-file,
    * which produces an xml response.
    */
    mySimpleAjax.loadXmlVm("internal/system/vm/custom/xml.vm");
}
```

Assign the function in question to the *onclick* event of the button.



Create the JS function *myXmlHandler1* as follows.

```
function myXmlHandler1(oXMLDocumentElement)
{
    var oLabelOutput = getElement("BD5E...D5CE"); /* labelcontrol*/
    oLabelOutput.innerHTML = oXMLDocumentElement.firstChild.data;
}
```

Save the application.

In the next step, create the file *xml.vm*, called by the function *getXMLResponse*, in the directory `<xtreme>/org/<portal>/internal/system/vm/custom/`. For the contents, we will enter the following line with the help of a text editor:

```
This text is <b>bold</b>.
```

Test the application in the browser.

#### 14.3.2. Sending with Request Values and Form Elements

##### Exercise

On the main page, create the two edit fields *Date* (type date and time, no linking) and *Price* (type currency, no linking). Also, enter a *static view element*, delete the title, switch it to *HTML* in the options, and under view, to the style class *Default*.

Next, create a button called *Output* without an action, and switch to the script editor:

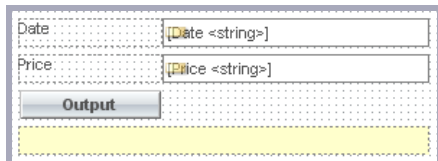
```
function getXmlResponse4()
{
    var mySimpleAjax = new upSimpleAjax();
    mySimpleAjax.oProcessFunc = myXmlHandler4;

    // Post DateTime and Float values
    var oLocalDateTime = getElement("1747...ABF5"); /*Date
datetimecontrol*/
    var oJsDateTime = getDateObject(oLocalDateTime);
    var strISODatetime =
oLocalDateTime.oUp.toISODatetimeString(oJsDateTime);

    var strLocalFloat = getElement("A930...4403").value; /*Price
currencycontrol*/
    var strFloat = Helper.getFloatStringByLocal(strLocalFloat);

    mySimpleAjax.loadXmlVm("internal/system/vm/custom/xml4.vm",{rq_param1
:strISODatetime, rq_param2:"value2"},{fr_param1:strFloat});
}
```

Assign the function to the *onclick* event of the button.



The screenshot shows a web form with two input fields. The first field is labeled "Date" and contains the text "[Date <string>]". The second field is labeled "Price" and contains the text "[Price <string>]". Below these fields is a button labeled "Output". Below the button is a large, empty rectangular area, likely intended for displaying the output of the function.