DesignBais

Responsive Design

Release 8

Table of Contents

Overview

This document describes the features of the DesignBais Responsive Design tool which forms part of the DesignBais toolset for creating browser-based application software.

The Responsive Design tool complements the existing DesignBais toolset. DesignBais Release 8 is completely compatible with DesignBais Release 7. There is no requirement for conversion or migration of DesignBais Release 7 forms and subroutines when upgrading to DesignBais Release 8.

Responsive Design

Responsive design is an approach to web page creation that makes use of flexible layouts, flexible images and cascading style sheet media queries. The goal of responsive design is to build web pages that detect the visitor's screen size and orientation and change the layout accordingly.

The DesignBais Responsive Design (RD) toolset is based on Bootstrap v3. Developers are encouraged investigate the Bootstrap Grid features, available on the web. A thorough understanding of these features will enhance your ability to design and implement responsive design forms.

🔒 https://getbootstrap.com/docs/3.4/examples/grid/

Bootstrap grid examples

Basic grid layouts to get you familiar with building within the Bootstrap grid system.

Three equal columns

Get three equal-width columns **starting at desktops and scaling to large desktops**. On mobile devices, tablets and below, the columns will automatically stack.

.col-md-4	.col-md-4	.col-md-4

Three unequal columns

Get three columns **starting at desktops and scaling to large desktops** of various widths. Remember, grid columns should add up to twelve for a single horizontal block. More than that, and columns start stacking no matter the viewport.

.col-md-3	.col-md-6	.col-md-3

Two columns

Get two columns **starting at desktops and scaling to large desktops**.

.col-md-8	.col-md-4

Notes for Developers using Responsive Design

DesignBais (not Responsive Design) sessions up to now, and ongoing, function in the following manner.

There are initially three hits on the database in order to establish a connection and rendering of a start form. The first hit initializes the new session. The second hit determines the form to be rendered. The third hit sends the data to populate the rendered form.

The ongoing activity involves the sending and receiving of xml packets containing changes to the rendered data, and change events to trigger actions. If the displayed form does not change then each server hit only passes data values that have changed. This minimizes traffic and therefore keeps response times as low as possible.

If a PROCESS.STACK command or a menu selection invokes a new form then the server hit on the database triggers the rendering and populating of the new form.

The DBISESSIONS file is used to hold the session state information.

Responsive Design is different in some ways. Interaction with the server involves the three hits that traditional DesignBais triggers only when a new session is started.

This allows Responsive Design forms to be rendered using a url containing the *dbpage* parameter that defines the page, or form, to be displayed. The display of the form framework, before it is populated with data, on hit two, is suppressed. The form and the data are rendered on hit three.
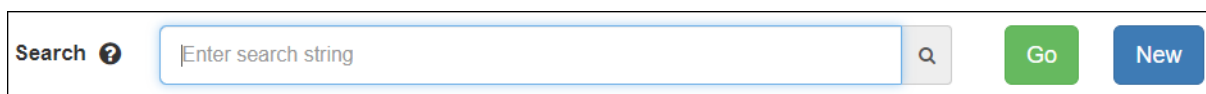
The start page of a Responsive Design application, if it captures data, must also include the *dbpage* parameter.

Responsive Design employs hit blocking, a feature that can be invoked in traditional DesignBais forms. All events after the first event are blocked.

In practice this means that if a change event is triggered by the change in the value of a field and a click event is generated by the user clicking a button then only one of these events will reach the database. Developers need to allow for this behavior in their basic subroutines.

Hit Blocking - implications

To demonstrate this consider the following Responsive Design form:

The *Search* field employs the *Lookup* form element. As shown in the following snip there is a read on this field. The read will be triggered by a change in the value of the *Search* field.

The *New* button has a click event associated with it. This button click event is intercepted by the basic subroutine and, in this example, displays fields to allow the user to enter a new record.

If the user enters a search string and selects a record the form will display as shown here. The change event, and therefore the read, will only happen when the user interacts with the form.

The interaction can be any of:
- Pressing the tab key
- Clicking either of the *Go* or *New* buttons
- Clicking anywhere on the form in fact!

The developer cannot implement this form and depend on their subroutine receiving a *button click* event when the *New* button is clicked. This is because the click of the *New* button will trigger the change event on the *Search* field (since it is now populated and was null) and the button click event will be blocked.

Note that if the *Search* field remains null and the *New* button is clicked then there will be a button click event as expected.

Date Formatting

Dates must be formatted as *yyyy-mm-dd* when sending a date value to the browser.

If using the ajax calls the date can be set as follows:

> rdSetVal("dateAdded","2019-06-26")

Where *dateAdded* is the form element id in the RD form.

Time Formatting

Time is based on 24-hour clock, therefore use value="08:56" for AM and value="20:56" for PM

DateTime-local Formatting

Uses the format: "YYYY-MM-DD\THH:NN:SS" . For example send " 2014-12-08T15:43:00". The time is in 24 hour format. In the format shown here "T" is just a delimiter (upper case). *DateTime* is currently not supported by the database component.

Month Formatting

Uses the format: "YYYY-MM". For example use "2015-01" to designate the month of January.

Populating DBRECORD, DBOTHER.RECORD, DBWORK

Read variables such as DBRECORD are built from the DesignBais common variable SCREENREC which is updated in the following ways:
- by values from the web page
- from DBVALUE if validating a field in the VALIDATE event
- by setting DBPASS.DBVALUE and DBPASS.DBVALUE.TO

The following example using DBPASS.DBVALUE demonstrates the preferred method of setting a default date in a field. This method allows DesignBais itself to format the date.

```
* set the default date in the Date Added field
 DBPASS.DBVALUE.TO = 'MUS.DATE.ENTERED'
 DBPASS.DBVALUE = DATE()
```

This second method will also work but relies on the developer to correctly format the date.

```
AJX.FUNC = 'SV'
AJX.ARG = 'dateAdded'
TODAY = DATE()
AJX.ARG<4> = OCONV(TODAY,"DY"):'-':OCONV(TODAY,"DM") "R%2":"-":OCONV(TODAY,"DD") "R%2"
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

```
006: <ajax><![CDATA[
007: document.getElementById("dateAdded").value="2019-06-26";
008: rdSetVal("search1","");
009: rdSetVal("title","");
010: rdSetVal("artist","");
011: rdSetVal("coverImage","");
012: rdSetVal("genre","");
013: rdSetVal("noOfPlays","");
014: rdSetVal("dateLastPlayed","");
015: rdShow("title","row",0);
016: rdShow("artist","row",0);
017: rdShow("dateAdded","row",0);
018: rdShow("coverImage","row",0);
019: rdShow("displayImage","row",0);
020: rdShow("genre","row",0);
021: rdShow("noOfPlays","row",0);
022: rdShow("dateLastPlayed","row",0);
023: rdHide("search1","row",0);
024: rdHide("play","element",0);
*--: EX
```

Tab Sequence requires DBRETURN.TO.FIELD

Unlike a classic DesignBais form the responsive design page contains field elements defined in an HTML string. These field elements do not have row and column values. For this reason the traditional tab index feature of classic DesignBais is not available.

The form data link process provides the ability to link a page field element to a field property definition. The sequence of field elements in the data link maintenance form does not necessarily correspond to the sequence of fields on the form.

Developers must use the common variable DBRETURN.TO.FIELD in their basic subroutine, to control the tabbing sequence within a responsive design page.

Page Flow

As stated above Responsive Design forms are rendered using a URL containing the *dbpage* parameter that defines the page, or form, to be displayed. This means that a user can reference any page by entering a URL such as:

http://192.168.0.0/websitename/dbnet.aspx?dbpage=dbdemo-musiccatalog

It is incumbent on the developer to ensure that users traverse pages in a valid sequence and are prevented from landing on a page that they are not permitted to run. There is a Basic Skeleton code example in DesignBais. Access this via the *Subroutine* button on the Code Editor form. Select from the *Select Code Segment* dropdown list.

For example, where a login page is part of the application, the developer must ensure that users are forced through the login page before they are permitted to enter any other page. The application should also ensure that pages are traversed in the correct sequence.

A suggested approach is include an After Display process on every page, including header and footer forms. The After Display processing must include checks to ensure that the user has validly logged in, and that the current page has been reached in a valid sequence.

The application could set up fields in DBSTORE(n) to track these actions:

DBSTORE(n)<11>   user logged in (record date and time for example)
DBSTORE(n)<12>   previous page
DBSTORE(n)<13>   valid next page(s)

The After Display code must then check that pages are traversed in a valid sequence.
For example the following code ensures that:
- the DBMUSIC_CATALOG form must be the first form
- if the user navigates directly to the header or footer form then they are directed to the first form in the sequence

```
*
CHECK.PAGE.FLOW:
*
   * first page must be CATALOG
   * cant land on HEAD or FOOT
   * can only land on STATS from CATALOG
   *
     BEGIN CASE
        CASE DBSTORE(1)<11> = ''
           PROCESS.STACK = 'DBMUSIC_CATALOG'
           DBSTORE(1)<11> = DATE():'*':TIME()
           DBSTORE(1)<12> = 'CATALOG'
           DBSTORE(1)<13> = 'STATS'
        CASE SCREEN.NO = 'HEAD' OR SCREEN.NO = 'FOOT'
           PROCESS.STACK = 'DBMUSIC_CATALOG'
        CASE SCREEN.NO = 'CATALOG'
           DBSTORE(1)<12> = 'CATALOG'
        CASE SCREEN.NO = 'STATS'
           IF DBSTORE(1)<12> # 'CATALOG' THEN
              PROCESS.STACK = 'DBMUSIC_CATALOG'
           END ELSE
              DBSTORE(1)<12> = 'STATS'
           END
     END CASE
     RETURN
```

Responsive Design Access

The Responsive Designer tool is accessed from the *Forms Designer* option on the *Responsive Design* menu option in the DesignBais tools menu displayed on the left hand side of the *Developer* Tools form (DBIFORMS_DEVELOP).

You must be in the target database account when you invoke the Responsive Design designer form. Check the bottom left of the Designer:

Account:192.168.199.9|PULSE.DEV7   Work Space:/indigolf/
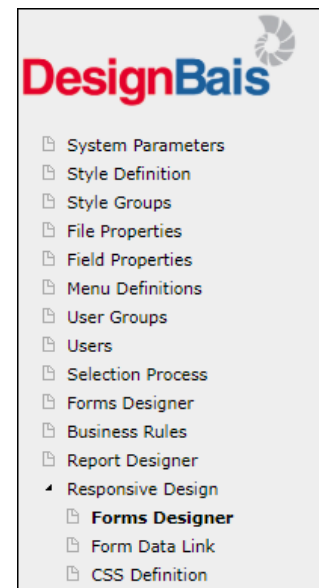
Responsive Design Structure

The DesignBais Responsive Design tool creates web pages that will run in any browser that supports the responsive design concept.

The web pages are stored in folders in a directory named *sites* within the website. In the example below the website is *v8a*. The release of DesignBais Responsive Design is *rdv1*.

Remote computer	C:\v8a\rdv1\sites	
Name		Object info
garb		File folder
garb2		File folder
garbrdfolder1		File folder
test		Local Disk
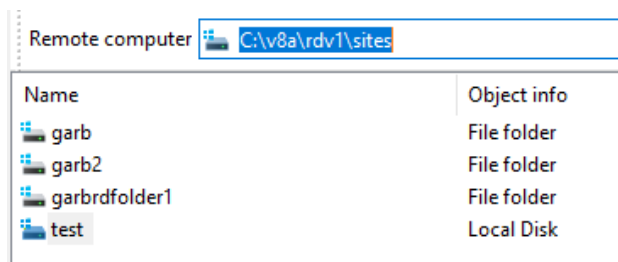
After creating a Responsive Design form and saving it in your work folder it must be published in order to run it.

We strongly recommend that the browser tab holding the Responsive Designer tool is closed after you have finished creating and publishing your RD page(s). Leaving the tab open, say overnight, and returning the next day can lead to problems due to the loss of session information when the originating session times out.

The *Publish* function relies on DesignBais being able to access the web server from your device using the IP address or server name (such as http://mywebserver/dbinetdemo/) because internally DesignBais attempts accessing its own functions using the IP address provided in the URL. The network config must allow that; i.e. a program running on the web server must be able to access that IP address.

DesignBais

- System Parameters
- Style Definition
- Style Groups
- File Properties
- Field Properties
- Menu Definitions
- User Groups
- Users
- Selection Process
- Forms Designer
- Business Rules
- Report Designer
- Responsive Design
  - **Forms Designer**
  - Form Data Link
  - CSS Definition

Selecting the Appropriate Responsive Design Component

Here is a guide to developers to assist in selecting the appropriate RD form element for a task.

- When you open a new page in RD Forms Designer you will see a single row. You can use CTRL+D to create additional rows.

- CTRL+I inserts a Form Row in a row. You need to create the Form Row as the container in which to place your form elements.

- Form elements are described in detail in following sections. This section is designed to give you a broad introduction.

Input Field
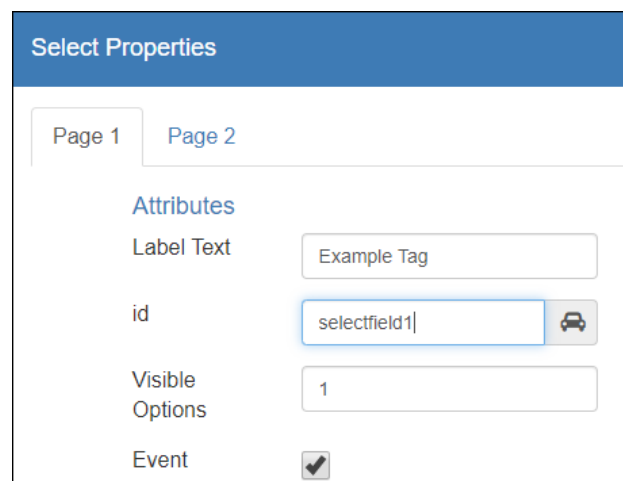
For input fields use CTRL+M to create a *Text Input* form element that will link to a field in your database. If your database field has a valid input list then use a *Select* form element. If it is an address then consider using the *Address* form element.

Select Field
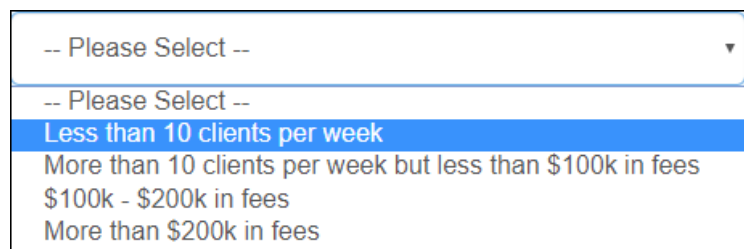
If your field has a valid input list then use CTRL+S to create a *Select* form element.

In general it is best to set the *Visible Options* to a value of 1. This is the default.

Click the *Event* checkbox if you require a change event.

That is all you need. Once you link the form element to a database field the valid input options will display.

Address Field

Use CTRL+A to create an *Address* form element for address fields.
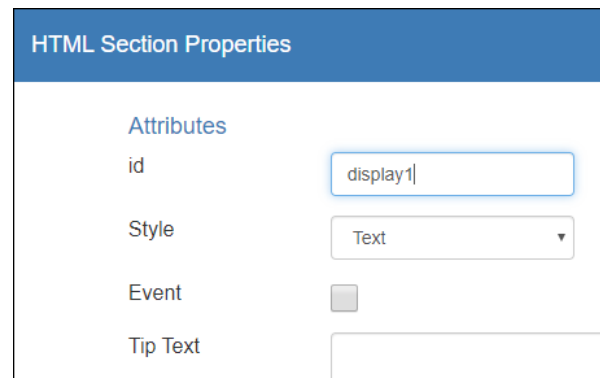
Button

Use CTRL+B to create a *Button* form element. This can also be used for tiles.

Output Field (Text, Image)

Use CTRL+K to create an *HTML* form element. Use these for placing text or images on a form.

Enter the id and tick if an *Event* is required. Then click Save.

HTML Section Properties

Attributes

id   display1

Style   Text

Event   ☐

Tip Text
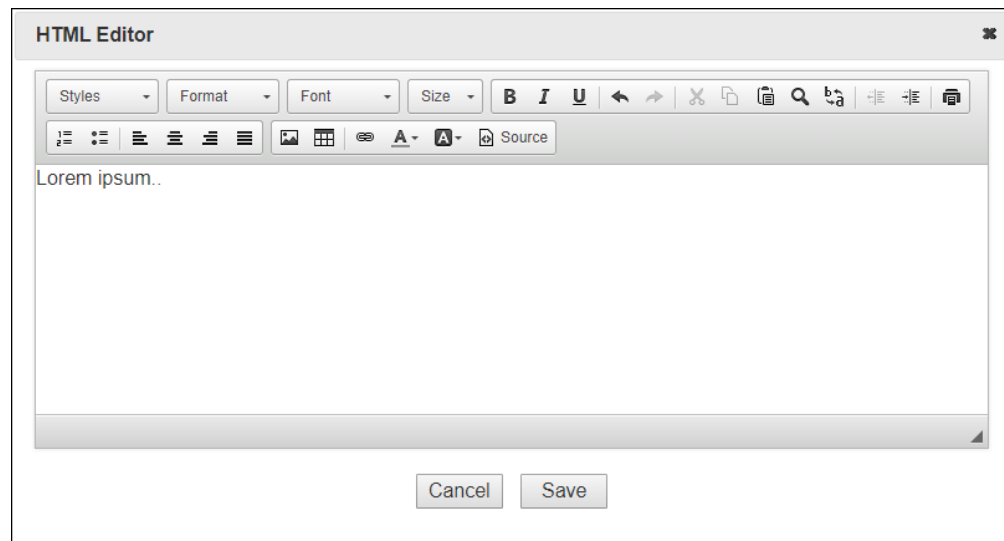
The resulting form element will look like this.

DesignBais Responsive

Lorem ipsum..

Focus on the element and click CTRL+Q to open the HTML Editor.

HTML Editor

Styles | Format | Font | Size | B *I* U | ↶ → | ✂ ⧉ ⧉ 🔍 | ⇤ ⇥ 🖨

Lorem ipsum..

Use the Editor to create text.

Cancel   Save

To insert an image click on the image button highlighted in yellow.

This opens the Image Properties window.

Insert the path to the required image. This will take the form of:

../res/foldername/images/imagename

The image will appear.

Refer to this manual for additional notes about inserting an image.
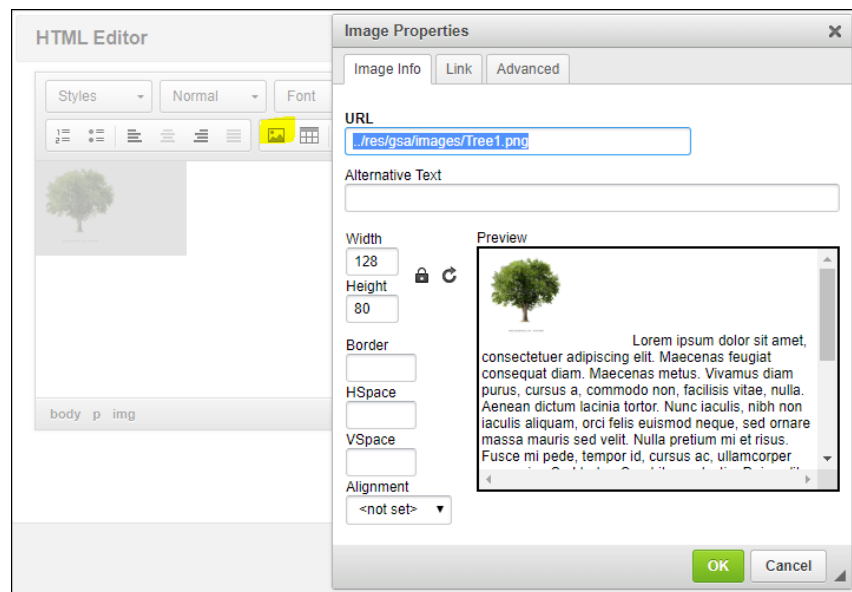
Radio Buttons

Use CTRL+R to create a Radio Button group. This will create a group containing two buttons. DesignBais assigns auto-generated ids to the group and the buttons, and sets the *Label Text* to "Radio".

You will have to change the *Label Text* since this is what will appear on your form. Enter CTRL+P to display the properties and change the *Label Text* to a meaningful description.

Alternatively, instead of using CTRL+R, you can focus on the form row, right click and select *Insert element* and then select *Radio [Ctrl+R]*. Using this method you will then have to create one button by entering values in the *Radio Options* fields and clicking *Add*. (Before you can save this form you must, as shown by the red error message in this image to the right, add one radio button.)

You will also have to assign an id to the group. Use the *auto* image to generate an id based on the *Label Text*. Avoid assigning an id that contains the text "radio" or "radioGroup" since DesignBais uses these strings to identify form elements.

In either case the default radio group button(s) generated by the Designer can remain. You do not have to match them to the actual buttons that you want to appear on your form.

When you link the form element to an actual database field, using the Form Data Link function, then DesignBais automatically sets up the radio buttons to match the valid input list settings on the linked field property.
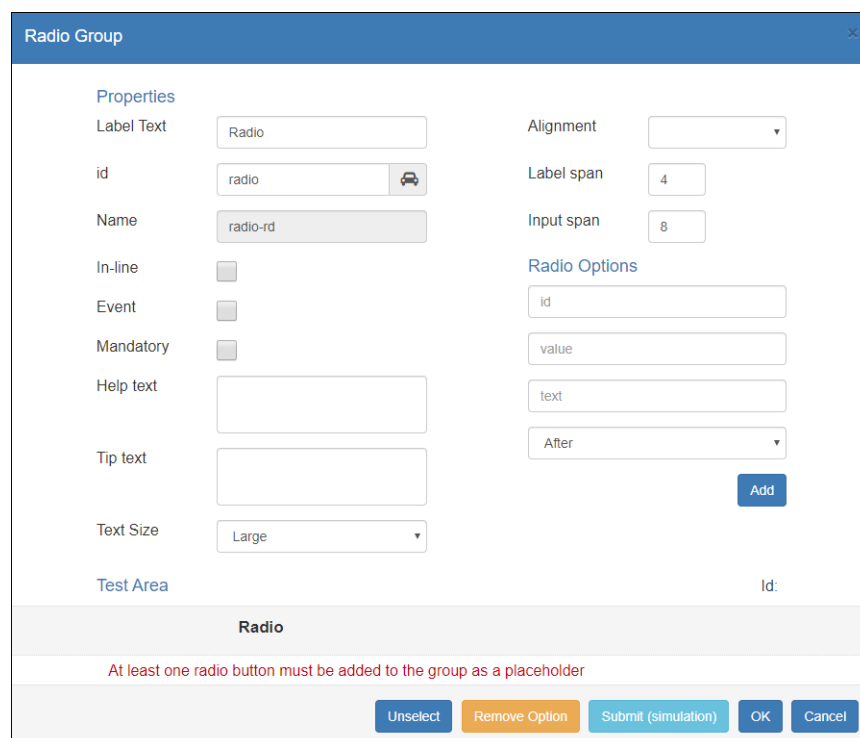
The *Label Text* is the only value that you need to set in Designer.

This example will clarify the above description. Press CTRL+R in a form row to create a default radio group, as shown here.

Press CTRL+P to open the Radio Group Properties. Amend the *Label Text* from the default value of "Radio" to "State". Click OK to save this change, then press Alt-L to save and publish the page. The name of the page is "state".

In the Form Data Link we link this page to a file with a field called GSA.STATE. This field has a valid input list containing the eight state and territory names.

Note field, that DesignBais builds the eight radio buttons based on the linked regardless of what exists on the RD Form as saved in the Forms Designer.

Checkbox

Use CTRL+H to create a checkbox. This will create a Checkbox Group with Group Properties as shown in the image to the right.

The Label Text defaults to the string *Checkbox*. This text will appear on your RD form unless you clear it in this form. It is a name for the checkbox group rather than a tag for actual checkbox so it is not essential to retain the Label Text.

When you link your checkbox to a database field the tag from the field property record on the database will, by default, display as the checkbox Field Text. This text can be modified as required and becomes the tag that is displayed on the RD form.

Two checkboxes are created as the default Checkbox Group. This builds the HTML construct that is used by the Form Data Link function when you link a checkbox form element to a database field and allows you to visualize how the checkboxes will appear on the form. Use the *In-line* option to display the form elements side by side as shown here.

The Checkbox Group must have an id. The RD Designer will provide a default but it is good practice to assign a meaningful id at this stage in order to facilitate the linking of database fields in the Form Data Link process. You can use the *auto* image to generate an id based on the *Label Text.* Avoid assigning an id that contains the text "checkbox" since DesignBais uses this string to identify form elements.

Note that the *Name* is generated by appending "*-cg*" to the *id*. This is not maintainable.

Alternatively, instead of using CTRL+R, you can focus on the form row, right click and select *Insert element* and then select *Checkbox [Ctrl+H]*.

Using this method you will then have to create one button by entering values in the *Add Checkbox to Group* fields and clicking *Add*. (Before you can save this form you must add a checkbox, as shown by the red error message in the above image.) So enter a meaningful id and text for the checkbox tag. Then click *Add*.

The text will be discarded in the Form Data Link process and replaced with a meaningful tag from the linked field. Click on the checkbox in the Test Area in order to display the id.

In Form Data Link you can link the checkbox to a field in the database.

Note that the concept of a group name for checkboxes, as implemented in standard DesignBais, does not exist in Responsive Design. Radio buttons provide this functionality.

Text Area

Select this form element when you need to allow text entry on a form.

The form element can be linked to a multivalued database field to store multiple lines of text.

At run time the text area displays like this:

Lookup

Use the Lookup form element when you want an input field driven by a lookup file.

In this example we require an input field for entering the Class.

There is a DBCLASS file containing records with the code as the key and containing the description.

Set the *Label Text* and *id* as required. Click the *Event* checkbox.

On *Page 3* select *Database Look up* from the *Type* dropdown.

Usually you would want to check the *Allow selection from the list items only*.

If checked, it does not allow the user to set the value as free text, the user must select one of the presented options. Currently all lookup elements are enforced as *listonly* by default. This is equivalent to checking this checkbox.

A future release of DesignBais will introduce a new element akin to a *datalist* which will allow entry of text regardless of the returned list.

Use the Form Data Link option to link the form element to the field in your database file that is to be populated with the Class Code.

In the *Lookup File* field enter the name of the file that will provide the lookup ids. This file must be a defined DesignBais file with a Dropdown Select Statement.

At run time the lookup displays like this:

The linked file must have a Dropdown Select Statement. The Dropdown Selection Field and Dropdown Description fields will determine the text that is displayed based on the characters entered in the lookup field at run time.

If the Lookup form element is linked to the id of the record being selected for display then you will need to have a button, such as "Add" or "New", which allows the user to create a new record. Typically this button will hide the Lookup field and display an input field.

Note that the lookup dropdown list is built when the form is loaded. The list is written to DBISESSIONS with a key like:

"48c92ab81d4c48e68d9db6cc8e2862d5*BNK.BSB*1*COMBO"

where the initial string is the session id and the second part of the key is the field name to which the lookup element is linked.

If new records are created in the form then they will not appear in the lookup dropdown list until the form is re-loaded. To get around this issue use the DesignBais DBDROPLISTADD function. For example in the BEFORE WRITE event, assuming the new record id is in DBKEY and the dropdown list description is in DBRECORD, the following code can be added to your basic subroutine:

```
*
BEFORE.WRITE:
*
    BEGIN CASE
        CASE SCREEN.NO = "your form name" AND PROCESS.PARAMETER = "RECORD"
            * add this record to the lookup list linked to file ABANK
            DBDROPLISTADD = "BNK.BSB"
            DBDROPLISTADD<2> = DBKEY
            DBDROPLISTADD<3> = DBKEY:' ':DBRECORD<BNK.NAME>
```
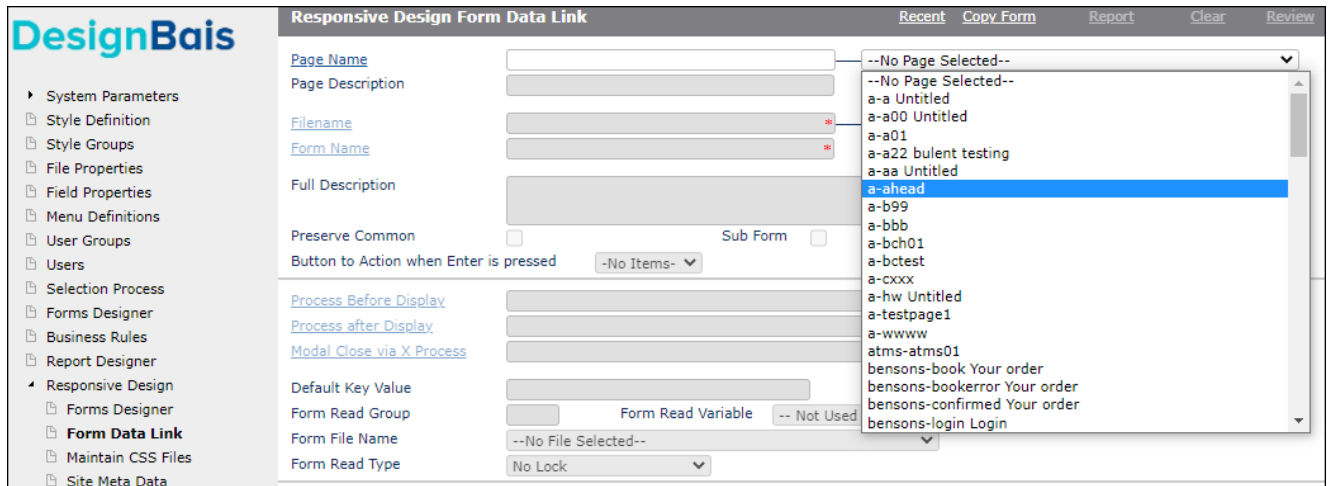
Substitute the name of your lookup field for "BNK.BSB" and the field name in the dropdown list description for "BNK.NAME".

If in the form update parameters the *Null After Write* parameter is set to *No* then this code can be placed in the AFTER WRITE event.

Linking the Responsive Design form to the database

Use the *Publish* [Alt L] option in the Responsive Design File menu to make a Responsive Design (RD) form available to the database component of DesignBais.

Use the *Form Data Link* option on the DesignBais tools menu displayed from the Developer Tools form (DBIFORMS_DEVELOP) to access the published RD form. Use either the *Page Name* link or the dropdown to select a published RD form.
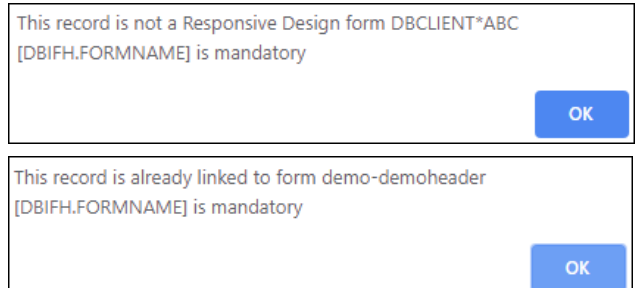
Then enter the database filename and form name to which the RD form is to be linked.

You must define a Filename and Form Name even if you do not intend to link any RD form elements to a field in the database. These two fields define the record id of a record stored in DBIFORMS. This record has the same layout as a traditional DesignBais form record and stores the form linking details, field names, field properties and the names of subroutines to be run from various slots such *Process After*. If no linking details are required then this DBIFORMS record must still exist but will be largely empty.

RD form records on DBIFORMS are flagged with the letter "H" in the DBIF.W3C.PDA field (attribute 178) of the form record.

You must link a new RD form to a new DBIFORMS record, that is, the record on DBIFORMS must not already exist. If you select an existing form name then either of these error messages will display. The first indicates that a traditional DesignBais form exists with this id. The second indicates that a different RD form has already been linked to this DBIFORMS record.

The Publish step creates a record in the DBIFORMHTML database file. The record id is formed by concatenating the work folder name, a dash (hyphen) and the form name. For example the record id for a form named ***contactDetails*** in a work folder called ***myFolder*** will be:

> ***myFolder- contactDetails***

The DBIFORMHTML record contains all the HTML code required to run the form as well as the name of the DBIFORMS record.

The Form Data Link option stores the linked database field details on the DBIFORMS record.

As stated above, you must use the Form Data Link option to create a record in the DBIFORMS file before a Responsive Design form can be run.

If this step is not done and you attempt to run the form the *Page not available error* will display.

Note however that header and footer pages no not need to be linked. All body pages must be linked. Header and footer pages that are linked to a body page will run as part of the body form.

Header and footer pages may be linked when required, such as if you want to link a form element on the header or footer to a database field, or if you want to define a process to be triggered by a click event.

GUID

The Responsive Page Designer creates a GUID to make the page unique across sites. Pages are stored in a site folder. Pages with the same name can exist in different sites.

For this reason the database stores the GUID in the DBIFORMHTML and checks it whenever a page is published.

A site page can only overwrite another published page of the same name (site-page) if it has a matching GUID.

The DBIFORMHTML file "site-page" record for the published page holds the GUID and the name of the linked form (file*form) in DBIFORMS. The form in DBIFORMS has a back-link to the DBIFORMHTML record.

If there is a GUID clash then the message to the right is displayed:

Use the "Save As" option to produce a new GUID.

If the original page is already linked you can clear the GUID attribute in the DBIFORMHTML record and publish will work.

The original links (form element to database field) will remain so long as XML ids do not change.
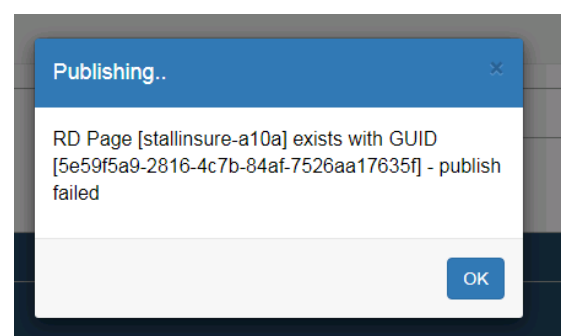
Responsive Design Form Data Link

The Responsive Design Form Data Link process allows you to link elements on the RD form to fields within the database. The linking process also creates a record in DBIFORMS on which the form elements and link data are held, in the same attributes as normal DesignBais forms. RD type forms are flagged with a value of "H" in the DBIF.W3C.PDA field (attribute 178) of the DBIFORMS record.

ID	Type	Unlink	File	Field	Variable	Prop	Event	Process After
search1	lookup	✕	DBMUSIC	MUS.ID	DBRECORD	LOOKUP	change	DBI.I.RD
searchGo	button	✕		B.SEARCHGO		BUTTON	click	DBI.I.RD
searchNew	button	✕		B.NEWRECORD		BUTTON	click	DBI.I.RD
title	text	✕	DBMUSIC	MUS.TITLE	DBRECORD	INPUT	change	DBI.I.RD
artist	text	✕	DBMUSIC	MUS.ARTIST	DBRECORD	INPUT		
dateAdded	date	✕	DBMUSIC	MUS.DATE.ENTERED	DBRECORD	INPUT	change	
coverImage	text	✕	DBMUSIC	MUS.COVER.IMG	DBRECORD	INPUT		
genre	select	✕	DBMUSIC	MUS.GENRE	DBRECORD	INPUT	change	DBI.I.RD
noOfPlays	number	✕						
displayImage	output	✕	DBMUSIC	MUS.DBKEY.WK	DBWORK	OUTPUT		
message1	output	✕						
dateLastPlayed	date	✕	DBMUSIC	MUS.DATE.LAST	DBRECORD	INPUT		
stats	button	✕		B.STATS		BUTTON	click	DBI.I.RD
clear	button	✕		B.CLEAR		BUTTON	click	DBI.I.RD
delete1	button	✕		B.DELETE		BUTTON	click	
save	button	✕		B.SAVE		BUTTON	click	
dupCheck	button	✕		B.DUPCHECK		BUTTON	click	DBI.I.RD
play	button	✕		B.PLAY		BUTTON	click	DBI.I.RD

Process After	Return To	Lookup File	Table/Group
DBI.I.RD			
DBI.I.RD	MUS.TITLE		
DBI.I.RD	MUS.TITLE		
DBI.I.RD			
DBI.I.RD			
DBI.I.RD			
DBI.I.RD	MUS.ID		
	MUS.ID		
	MUS.ID		
DBI.I.RD	MUS.TITLE		
DBI.I.RD	B.SAVE		

Test

Note the *Preserve Common* and *Sub Form* fields on the Form Data Link form.

If the RD Page is part of a series of pages, then you must check both of these fields on all RD pages in the series.

Responsive Design Form Data Link　　　　　　Recent　Copy Form

Page Name	gsa-testcb	── gsa-testcb
Page Description		
Filename	AGSADEMO	── AGSADEMO GSA RD Demo
Form Name	TESTCB	
		CSS Filenam
Full Description		Header
		Footer
Preserve Common	✔	Sub Form ✔

To put it another way, if you have *Next* and *Back* buttons on a form then it is almost certain that you will need to check the *Preserve Common* and *Sub Form* fields.

Fields

Page Name
The key of the record in the DBIFORMHTML file that is generated by the *Publish* command in the RD tool. It is formed from concatenating Foldername:"-":Formname.

Page Description
Description for this page. Free text.

Filename
The name of the database file to which the DBIFORMS form record will be linked.

Form Name
The name of the form on the DBIFORMS file. The id of the record on DBIFORMS will be *Filename*FormName.*

Full Description
The full description for the form record on DBIFORMS.

Preserve Common
Check this box if the RD form is to inherit common variables from the calling form. This means that any form that is the target of a *Back* button, say, must have this field checked. Typically both the calling form and the called form will require *Preserve Common* to be checked.

Sub Form
Always used in conjunction with the *Preserve Common* field. Check this box if the RD form is to inherit common variables from the calling form.

Process Before Display
Process to execute before display of the RD form. The common variable PROCESS.EVENT will be set to "AFTER DISPLAY". The common variable PROCESS.EVENTSOURCE will be set to name of the RD form.

Parameter
The value entered here will be passed to the subroutine in the common variable PROCESS.PARAMETER.

Process After Display
Process to execute after display of the RD form. The common variable PROCESS.EVENT will be set to "AFTER DISPLAY". The common variable PROCESS.EVENTSOURCE will be set to name of the RD form.

Parameter
The value entered here will be passed to the subroutine in the common variable PROCESS.PARAMETER.

Modal Close via X Process
This slot contains the name of the basic subroutine to call whenever the main form gains focus after a modal form is closed. Whenever a modal form closes then PROCESS.EVENTSOURCE will be set to contain the name of the closed modal form. SCREENROOT will contain the name of the currently active base form, the name of the form that called the modal form. PROCESS.EVENT will contain the event "MODAL RETURN". See further details below.

Default Key Value
Refer to the DesignBais Reference Manual. This group of fields performs the same function as in the normal Forms Designer.

The default key value is used to define a single record key for the form. The default key can be either a string value, or, if a variable key is required enter just V:. The V: will trigger the DERIVED KEY

	event which will be processed by the subroutine named in the *Process After* Display slot. The DERIVED KEY event must assign the required key to DBVALUE.
Form Read Group	The form read group is used when a default key is used. The value entered here defines a unique handle for the read that takes place when this form is entered. No read groups within a form or a form-set can be re-used. By convention, but not required, Read Group 1 is assigned to the DBRECORD Form Read Variable, Read Group 2 through 99 to the DBOTHER.RECORD(2) through (99) variables (DBOTHER.RECORD(1) is often not used in this scenario so that the Read Group can remain in sync with the Form Read Variable).
Form Read Variable	This field controls which of the 100 variables is used to contain the record that is read. You have the choice of DBRECORD or DBOTHER.RECORD(1) to DBOTHER.RECORD(99).
Form File Name	This field is used to determine which file the record is read from when a Default Key Value is used.
Form Read Type	All reads require the Read Type to be assigned from the dropdown list.
RD Form Element List	The on-form report displays the list of form element id and type from the RD form.
	Initially, for a new form, the other columns of the report a blank. By clicking the element id in column 1 the developer can define the link between the form element id and a field on the database.
Search for ID	Use this search field to locate a form element id. Enter a character string then press tab. Form element ids that contain this string will be displayed starting at row 1.
	To re-display in the original sequence clear the search string and press tab.

Buttons

Submit	Save the currently displayed DBIFORMHTML record and the linked DBIFORMS record.
Clear	Clear the form.
Delete	Delete the currently displayed record from DBIFORMHTML.
View HTML	Displays the RD form HTML code.
Updating	Opens the RD90 form to permit the developer to define the action to take when the update button is clicked.

	Variable to Update	Update Type	Process before Update	Param	Process After Update	Param	Update From Button	Update Override	Confirm Update	Null After Write
	DBRECORD ▼	Write / Release ▼					B.BUTTON1 ▼	No ▼	Yes ▼	Yes ▼

Clear From Button   -- Select -- ▼

Clear from field   -- Select -- ▼

[Submit]   [Cancel]

Table Fields	Toggles the display of table row fields in the on form report of form element ids. Click on any table id to open form RD180G to define actions associated with the table elements. Note that table fields behave in a similar way to multivalue grids in a traditional DesignBais form.

Table Definition   **Table Name** d557230

ID	Type	Unlink	File	Field	Variable	Prop	Process After	Return To	Lookup File
tablerg	text	✖	DBCLIENT	DBC.INV.DATE	DBRECORD	INPUT			
d636468	text	✖	DBCLIENT	DBC.INV.NUM	DBRECORD	INPUT			
d298932	text	✖	DBCLIENT	DBC.INV.AMT	DBRECORD	INPUT			

[Close]

Click any field ID in column 1 to open form RD160 which allows entry of the Input Field Definition.

Input Field Definition   Submit   Business Rules

Field ID   d636468   View Field Properties   View Help Text

File Name   DBCLIENT — DBCLIENT - Client Test & File ▼   Files Used

Field Name   DBC.INV.NUM — DBC.INV.NUM ▼   DBCLIENT

Variable to Use   DBRECORD - File = DBCLIENT - Read = DBC.CLIENT.CODE ▼

Field Type   ALPHA   Attribute   42   Multivalued   Y   Subroutine Calls from Field Properties

Field length   10

Lookup File

Button Only ☐

Process After   Parameter

	Read Group	Read Step	Prefix	Delimiter	Read to Variable	File To Read	Read Type	Process Before Read	Before Read Parameter	Process after Read	After Read Process Parameter
					READ STEP ONLY ▼	--No File Selected-- ▼	-- Select Read Type -- ▼				

Derived in Subroutine   Parameter

Default

Valid Input	Valid Display

[Submit]   [Cancel]

Table Updating	Click to open form RD70 which allows the entry of the Table Update Properties.

Test   Click to test the linked form.

Header Bar Buttons

Copy Form   Copy a responsive design page and linked form.

Clicking the *Copy Form* button opens form RD15. The copy from page will default to the page that is being maintained in form RD10.

The copy from form must already be linked to a database form.

If you are copying to a page and linking it to a different file then the list of linked fields will reflect the equates prefix of the new file.

You can add a second prefix in order to create a set of names that are grouped. See the example above where the second prefix of "NEW" is inserted to change BNK.BSB to BNK.NEW.BSB.

You can also use the grid fields to change *"Original Text"* to a new value. This will be applied to all text in the new form including subroutine names and field names. Note in the example above that the field name BNK.ADDRESS is to be changed to BNK.NEW.POST.ADDRESS.

Modal Close via X Process

Modal Close via X Process is the equivalent of the *Modal Form Return Process* in traditional DesignBais. It is invoked in RD when a modal form is closed via X.

In RDMODE it is only invoked when:

EVENTTYPE[1,10] = 'rdConfirm-' and FIELD(EVENTSOURCE,'-',1) = "dbModal"

This is when a modal form is closed via X and there is a database server hit in the rdShowModal. The *EVENTSOURCE* passed to the developer is the ID used in rdShowModal.

For consistency with non-RD the common variable DBMODAL.CLOSE.VIA.X is set to 1 before calling this routine. In addition PROCESS.CLOSE.MODAL and DBRESTORE.BEFORE.MODAL are set to 0 after the subroutine is invoked and therefore cannot be set by the developer.

The *Submit* can simply call rdHideModal.

192.168.199.194 says

modalEmail RETURN

OK

The Form Clear Button

Create a button in the RD forms designer to use as the *Clear From Button* in the DesignBais form setup.

Publish the RD form.

In *Form Data Link* display the form. The new button will appear as an unlinked form element. In the example below the form element Id is *btnClear*.

ID	Type	Unlink	File	Field	Variable	Prop	Process After
clientCode	text	✖	DBCLIENT	DBC.CLIENT.CODE	DBRECORD	INPUT	DB.I.MYFORM
btnRead	grpBtn	✖		B.BTNREAD		BUTTON	DB.I.MYFORM
clientCodeWk	text	✖	DBCLIENT	DBC.CLIENT.CODE.WK	DBWORK	INPUT	
btnReadWk	grpBtn	✖		B.BTNREADWK		BUTTON	DB.I.MYFORM
clientName	text	✖	DBCLIENT	DBC.CLIENT.NAME	DBRECORD	INPUT	
d735593	button	✖					
btnClear	button	✖					

If you now click the *Updating* button you will find that the clear button is not present in the dropdown.

Update parameters

	Variable to Update	Update Type	Proce
➜ ✖	DBRECORD ▾	Write / Release ▾	

Clear From Button   -- Select -- ▾

Clear from field

    -- Select --
    B.BTNREAD
    B.BTNREADWK

Submit   Cancel

It is necessary to click on the *btnClear* cell in column 1 of the *Form Data Link* form and then submit the *Button Definition* form even if no other changes are made. This will create the B.BTNCLEAR button in the DBIFORMS form linking record.

Button Definition

Field Xml Label	btnClear
Button Name	B.BTNCLEAR

Process After		Parameter
Return to Field		

Submit   Cancel

ID	Type	Unlink	File	Field	Variable	Prop	Process After
clientCode	text	✖	DBCLIENT	DBC.CLIENT.CODE	DBRECORD	INPUT	DB.I.MYFORM
btnRead	grpBtn	✖		B.BTNREAD		BUTTON	DB.I.MYFORM
clientCodeWk	text	✖	DBCLIENT	DBC.CLIENT.CODE.WK	DBWORK	INPUT	
btnReadWk	grpBtn	✖		B.BTNREADWK		BUTTON	DB.I.MYFORM
clientName	text	✖	DBCLIENT	DBC.CLIENT.NAME	DBRECORD	INPUT	
d735593	button	✖					
btnClear	button	✖		B.BTNCLEAR		BUTTON	

Click the *Updating* button and the *Clear From Button* dropdown list will display the button *B.BTNCLEAR* which can then be assigned as the *Clear From Button*.

Radio Button and Checkbox Form Data Link

When we link database fields to the Radio Button and Checkbox form elements DesignBais rebuilds the form HTML to match the database fields.

When you link the Radio Button form element to a field in the database that has a valid input list then DesignBais creates a radio button element for each valid input.

You have the choice to include or exclude values.

The linked field is then displayed in the Form Data Link form.

Search for ID						
ID	Type	Unlink	File	Field	Variable	Prop
d151721	radio	✖	AGSADEMO	GSA.STATE	DBRECORD	RADIO

The form displays the set of radio buttons derived from the valid input list.

State

- ○ NSW
- ● VIC
- ○ ACT
- ○ QLD
- ○ SA
- ○ WA
- ○ TAS
- ○ NT

Problem when updating existing Radio Button or Checkbox

If you change an existing Radio Button or Checkbox and find that the element no longer functions correctly, or you see an error like the one below, then follow these steps:

badev says

rPriceOpt1 original checkbox HTML not in the published page record

OK
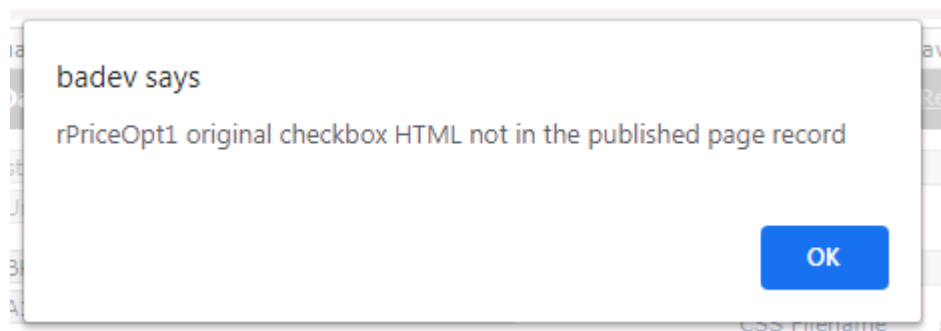
1. Go to Form Data Link and unlink the radio button or checkbox.
2. Go to forms designer and open the form.
3. Delete the radio button or checkbox.
4. Publish the form.
5. Reopen the form.
6. Re-create the radio button or checkbox.
7. Save, publish, and close the form.
8. Go to Form Data Link and link the checkbox.

Step 4 (Publish the form) saves the form in the designer (rdv2/site item) and updates the database. This removes any trace of the element from the DBIFORMHTML record in the database.

The HTML element id of the element (radio or checkbox) is stored in the DBIFORMHTML record so that any change to the HTML can be applied by searching and locating the element id when the form is next published. When you publish without the radio of checkbox in the page then all trace of the element is deleted from the database record.

If you use the *Edit Code* option, then the following notes may help:

Checkboxes and radio buttons created in the designer are used only as template and are replaced by the database in the Form Data Link process. This process attempts to leave line feeds and indenting for readability when the form is run. Removing them from the HTML string is not a precise science. Changing the style of checkbox elements in the edited code can also interfere with this process.

To avoid white space issues, as far as possible, the database replaces the checkbox id generated by the designer with the database generated id, which is unique.

When editing code Developers should use Do not use *id="fred"* rather than *id = "fred"* (i.e. with spaces). The correct input element id can also be used. For example, if name used in the designer is *rPriceOpt1* then the input id will become *checkbox-rPriceOpt1-1*. So, developers can use this id rather than a default value like d72033.

It is good practice, when editing code, to get the format from a standard element.

If there are problems, start fresh following the standard instructions:

- save the edited code
- remove the field from the designer
- publish to the database (link will be deleted)
- re-add the field in designer
- publish and re-data link

Running the Responsive Design form from the database

To run a responsive design form either set the start form for your DesignBais user as a responsive design form, or call a URL with a query component containing the RD form name:

"/?dbpage=test-clientMnt;newWindow"

This is best accomplished by setting up a new DesignBais user id with, say, an *rd* suffix. Then set up a login qcode querystring for this user to direct DesignBais to the start account and the responsive design start form.

The target account is determined by the querystring parameter "*ac*" in the DesignBais db.config file. The *?ac=string* is a qcode entryPoint in the db.config file which is explained in the Web Component Manual.

Example:
Normal DesignBais user id: dbfred
Set up an RD user id: dbfredrd

For user *dbfredrd* ensure that the Start Account and Start Form in User Maintenance are set up:

Entry Accounts/Forms		Account or Account Path	Start Form
Display Start Account List	↱ ✕	DB.NET	RD.START.FORM

In db.config

```
<entryPoint qcode=" dbfredrd ">
  <loginHost>192.168.199.9</loginHost>
  <BASUBROUTINE>BAWEBEXECNET</BASUBROUTINE>
  <loginHostType>UNIVERSE</loginHostType>
  <loginAccount>DB.NET</loginAccount>
  <loginUser>loginuser</loginUser>
  <loginPassword>password</loginPassword>
  <loginPublicUser>dbfredrd</loginPublicUser>
  <requestTimeoutSeconds>30</requestTimeoutSeconds>
  <debugUser>dotnetdev</debugUser>
  <enableXSSshield>false</enableXSSshield>
  <allowDomainNamesInLoginNames>false</allowDomainNamesInLoginNames>
  <convertLoginNamesToLowercase>true</convertLoginNamesToLowercase>
  <enableDetailedErrorMessages>true</enableDetailedErrorMessages>
 </entryPoint>
```

Use your DesignBais url and attach the querystring parameter. So, for example, if your website is called *dbnet* then the url would look like this:

http://localhost /dbnet/?ac= dbfredrd or with ip address: http:// 192.101.101.101/dbnet/?ac= dbfredrd).

Note also that Auto login is not applied to URLs containing *dbpage*. This is relevent to Responsive Design where *dbpage* is used to open a specific form such as in this example:

http://192.168.199.194/dbnet/?ac=ws&dbpage=dbweb-home

Note that, when published pages in DBIFORMHTML are transferred from a development system to a run-time system, you also need the *template-rdv2* record. This record tells DesignBais how to reconstruct your pages.

The *template-rdv2* record is created every time you publish a page. It will therefore be in your development environment but not necessarily in the run-time system.

The record in DBIFORMS created by the Form Data Link process are like any other non-Responsive Design DesignBais form except they are linked to the DBIFORMHTML record that is created from the Responsive Design page. The DesignBais engine simply reads the HTML required from DBIFORMHTML rather than constructing it from the raw form record which is the process that occurs in traditional DesignBais.

Forms, fields, programs, and styles all need to be available in the DBI files to which the account is linked.

In order for the Responsive Design Designer to run in an account the matching site records from the web site, the rdv2, site and res folders, must be available. Any new data account requires the DBIFORMHTML template-rdv2 record. It will be created when the first page is published in the data account.

Error due to Missing template-rdv2

If the template is missing an error similar to that shown below will display.
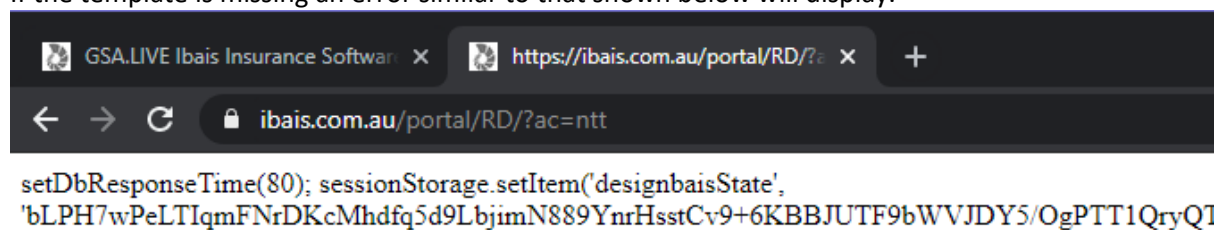
Deploying Responsive Design applications

Ensure that the resources for your site are set up in the target environment. These are stored in the *res* folder in the development site. The site folder within the *res* folder from the development site must be copied to the target site. In the following example:

- The website name is: bawebdevv7
- The work folder is: UnitedMonument
- Copy **bawebdevv7\res\UnitedMonument** to the target environment

OS (C:) ▸ bawebdevv7 ▸ res ▸

Name

- bulent
- css
- dbdemo
- default
- famous
- famousbikes
- fonts
- gsa
- images
- IRT
- Ka
- katrina
- meta
- stallinsure
- UnitedMonument
- placeholder.txt
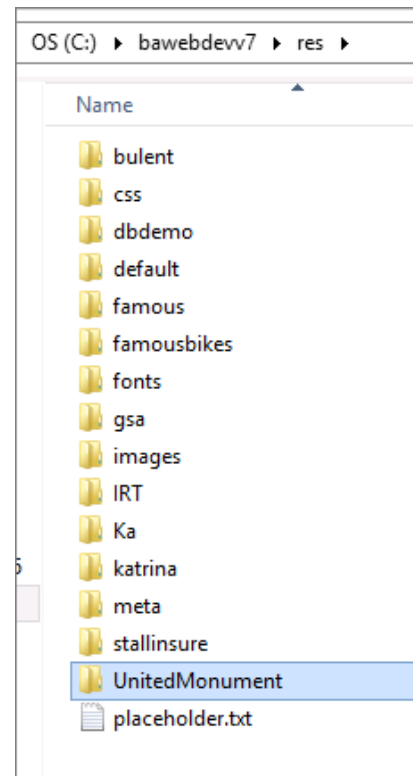
If you want to have use the Responsive Design forms designer in the target environment then you will need to also copy the *rdv2***UnitedMonument** folder.

Note that this folder is not required at runtime so forms can be deployed to run in a target environment without *rdv2* folder.

For runtime you need:
- Your application code in your basic library routines
- The DBIFORMS records (and all DBIPROP fields) that have been linked to the RD pages for the linked forms
- DBIFORMHTML records that contain the runtime HTML for your RD pages.

▸ OS (C:) ▸ bawebdevv7 ▸ rdv2 ▸ sites ▸

Name

- bulent
- dbdemo
- famousbikes
- gsa
- IRT
- Ka
- katrina
- stallinsure
- UnitedMonument
- gsatest4
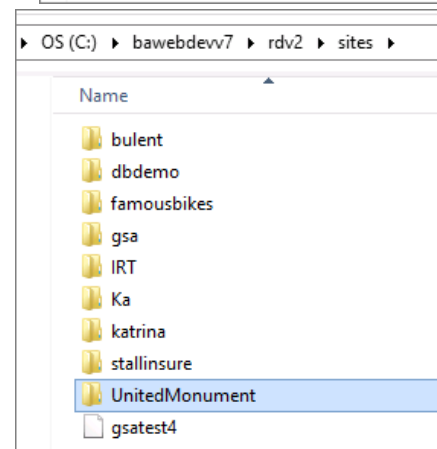
Maintain CSS Files

Use this option to create custom CSS files for use in DesignBais forms, both Responsive Design and traditional forms.

Responsive Design CSS definitions are held on the DBISTYLEGROUP file. They are distinguished from style group definition records by the *CSS Type* code *R.* Style Groups have a NULL type code.

Records created using this option are stored in the *css* folder within the *res/workfolder* folder on the website. In the example below the name of the workfolder is *dbdemo.* The name of the website is DBNET.

Use the *cssCustom Sub-Folder* to reference a sub-folder within the cssCustom folder in order to group and track your css files.

Removing the *cssCustom Sub-Folder* name from your definition will move the css record from the sub-folder to the main *cssCustom* folder.

Adding a *cssCustom Sub-Folder* name to your definition will move the css record from the main *cssCustom* folder to the designated sub-folder. The sub-folder will be created if it does not exist.

Use the *Style to Add* button to select a style from the DesignBais DBISTYLE file. Then click *Add* to add a copy the style properties to the css file being maintained.

Note that in order to maintain a css file that exists in your website but not in the database DBISTYLEGROUP file it is necessary to copy it manually using, say, notepad.

Create a new record in the Responsive Design CSS Definition maintenance form and call up this record. Then go to the website and open the website css file in notepad, copy it and paste it into the *Style Sheet Details* field on the database maintenance form. Adjust as necessary and save it.

Providing the *cssCustom Sub-Folder* field is populated with a path to the website *res* folder then any changes made on the database side will be update to the css folder, such as, for example:
    C:\DBNET\res\dbweb\css

Site Meta Data

This form allows maintenance of CSS, javascript and meta data for RD sites in the account or globally. The res folder may also hold meta data for the individual page.

The records containing the site meta data are held on DBIPARMS. The record id is composed of the prefix "RDMETA*" followed by the name of the site. The *All Sites This Account* option is held in a record with id "RDMETA*ALL".

The CSS links are held in attribute 1.
The Script and Meta Tags are held in attribute 2.
The Body Elements are held in attribute 3.

Fields

Site	Select the site from the dropdown list. Alternatively you may create site meta data for all sites in the current account.
CSS	The list of CSS Links to add for the selected site, or for all sites in the account.
Script and Meta Tags	The list of site script and meta data tags. Meta data will be applied to all pages in the selected site.
Body Elements	The list of links to add to the <body> of pages for the site.

Buttons

Submit	Save the currently displayed DBIPARMS record.
Clear	Clear the form.

Delete	Delete the currently displayed record from DBIPARMS.
Global	Opens the *Responsive Design Global Meta Data* form. The meta data is held on the DBIGLOBAL file in a record with id "RDMETA".

Site meta data files are added in the following order:
1. Global CSS
2. All Site CSS
3. Site CSS
4. Page Theme from Designer
5. Global Meta
6. All Site Meta
7. Site Meta
8. Page Meta Data res/site/meta/page.meta

The last occurrence of a style will be used unless the "*!Important*" setting is used.

If you have to move meta data down the list of locations shown above in order to invoke a particular style for example then you will need to republish the page(s) to pick up the change.

For example if a css file in the Site Meta is moved to the Page Meta Data in the res/site folder then effected pages will need to be republished.

Creating a Responsive Design Application

Log in to DesignBais and from the Developer Tools home page select the Account Selection option from the top menu. Select the account that you wish to develop in and click the Go button.

From the Developer Tools side menu select the Responsive Design option. This expands into 2 options.

Forms Designer opens the Responsive Design Forms Designer.

Form Data Link opens a form that allows you to link field elements on a Responsive Design Form or Web Page to database fields and processes.

We recommend Google Chrome for the RD Form Designer.
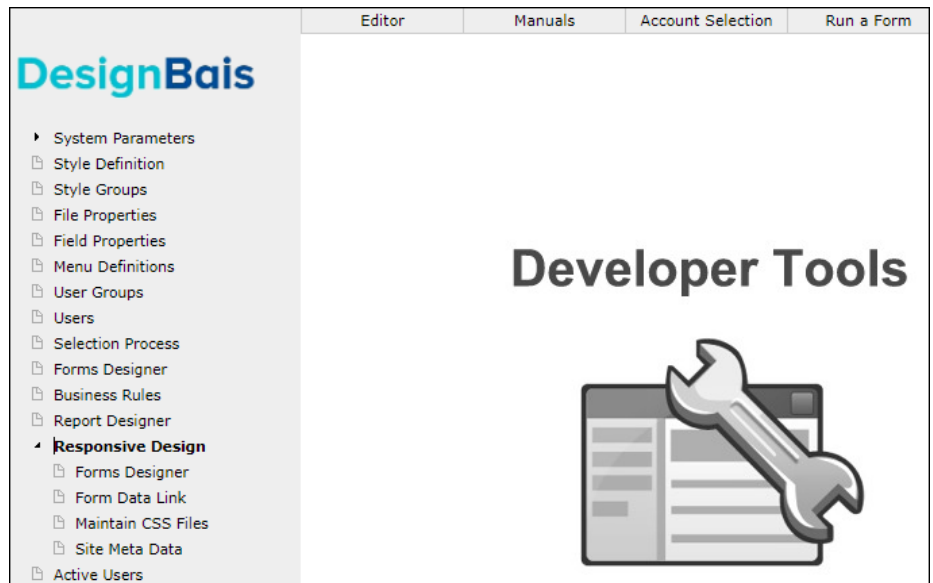
IMPORTANT: Please make sure that none of your logins or passwords are saved by browsers (e.g. Chrome) on the urls on which you do your development (e.g. http://localhost/dbnet). Otherwise any login or password field that you add to your page may be saved with your login or password on it and you don't want that.

Use the *Responsive Design* menu option and select the *Forms Designer* option.

This opens the Responsive Design (RD) tool where RD forms are created. Refer to the sections following in this manual.

After completing the form save and publish it. The *Publish* option runs a web service to link to the database and create a corresponding item in the DBIFORMHTML file. The key of the item is:

> Foldername:"-":Formname

For example if your work folder is called *bob* and your form is called *example1* then the key of the published record in DBIFORMHTML will be *bob-example1*. This is the equivalent to the Work Space name that is displayed in the status bar in the RD tool with / replaced by -.

Use the *Form Data Link* option on the *Responsive Design* menu to open the Responsive Design Form Data Link form (DBIFORMS_RD10). Refer to the following section which describes how to use this form.

Responsive Design - db.config File

The db.config file resides in the admin sub-folder within the website folder. DesignBais uses the db.config file to determine how to connect to the database.

The default connection path is defined in what we call the null qcode entry point. This is the entry point shown below: *<entryPoint qcode="">*. It is the recommended way to connect RD users. The *loginPublicUser "normaluser"* could be set up to have a login form in which each user enters their user id and password and are then directed to their start account and start form. The start form should be an RD form.

```
<entryPoint qcode="">
   <loginHost>192.168.199.9</loginHost>
   <BASUBROUTINE>BAWEBEXECNET</BASUBROUTINE>
   <loginHostType>UNIVERSE</loginHostType>
   <loginAccount>DBINETWEBSITE</loginAccount>
   <loginUser>dbnetuser</loginUser>
   <loginPassword>pwxxxxxxx</loginPassword>
   <loginPublicUser>normaluser</loginPublicUser>
   <requestTimeoutSeconds>30</requestTimeoutSeconds>
   <debugUser>dbiwebsite</debugUser>
   <enableXSSshield>false</enableXSSshield>
   <allowDomainNamesInLoginNames>false</allowDomainNamesInLoginNames>
   <convertLoginNamesToLowercase>true</convertLoginNamesToLowercase>
   <enableDetailedErrorMessages>true</enableDetailedErrorMessages>
 </entryPoint>
```

The db.config file may contain multiple entry points. By using a "?ac=name" query string in the url used to access DesignBais you can direct DesignBais to commence in a particular account. See the example below: *<entryPoint qcode="website">*.

```
<entryPoint qcode="website">
   <loginHost>192.168.199.9</loginHost>
   <BASUBROUTINE>BAWEBEXECNET</BASUBROUTINE>
   <loginHostType>UNIVERSE</loginHostType>
   <loginAccount>DBINETWEBSITE</loginAccount>
   <loginUser>dbnetuser</loginUser>
   <loginPassword>pwxxxxxxx</loginPassword>
   <loginPublicUser>salesuser</loginPublicUser>
   <requestTimeoutSeconds>30</requestTimeoutSeconds>
   <debugUser>dbiwebsite</debugUser>
   <enableXSSshield>false</enableXSSshield>
   <allowDomainNamesInLoginNames>false</allowDomainNamesInLoginNames>
   <convertLoginNamesToLowercase>true</convertLoginNamesToLowercase>
   <enableDetailedErrorMessages>true</enableDetailedErrorMessages>
 </entryPoint>
```

A url like:

> *192.111.123.122/dbnet/?ac=website*

will find this entry point and connect to the database account named DBINETWEBSITE using the login credentials of user dbnetuser with password pwxxxxxxx.
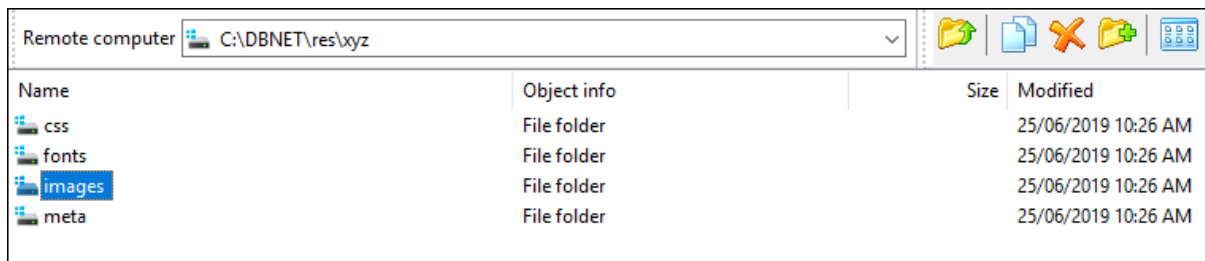
The DesignBais user for the session, however, is taken from the loginPublicUser. So the WEBLOGON for the session will be, in this example, *salesuser*.

The rule to be followed by RD form designers is to ensure they are in the same database account as the database files they plan to use.

When publishing an RD form make sure that this is done in the account in which the application is to run. The publish step will create, or update, a record in the DBIFORMHTML file. In environments where there is more than one set of DesignBais accounts the developer should make sure that they are referencing the correct DBIFORMHTML file. The status bar displayed at the base of the forms designer tool shows the account name and the work folder name.

Creating a Work Folder

Creating a new work folder triggers the creation, in the website res (for Resources) folder, a folder with the name of the newly created work folder. Within this folder there are four sub-folders.

Name	Object info	Size	Modified
css	File folder		25/06/2019 10:26 AM
fonts	File folder		25/06/2019 10:26 AM
images	File folder		25/06/2019 10:26 AM
meta	File folder		25/06/2019 10:26 AM

Remote computer: C:\DBNET\res\xyz

- css   holds the css files for the work folder.
              Create a css file here. The best way is to copy one of the existing preset css files in that folder, paste into the same folder under a different name and edit it as needed. When designing an RD page the css file can be selected as the theme for that page.

- fonts   holds any required special fonts.
              Save a custom font file here (e.g. grafiti.ttf) and refer to it from the css file as follows:
              ```
              @font-face{
                      font-family: grafiti;
                      src: url('fonts/grafiti.ttf');
                              }
              h3 {
                      font-family: grafiti;
                      }
              ```

- images   this is where you must place images that are used by your RD forms.
              All images used in the RD pages for a given site should be saved in this folder. In RD HTML Editor refer to images using a relative path like this:
                      "../res/xyz/images/tree1.png".

- meta   holds http responses headers and javascript metadata.
              All meta tags (i.e. references to external JS files, or to external CSS locations or any other HTML meta tag) for a given page (e.g. "mypage") should go into a file named mypage.meta saved in this folder.

Responsive Design Custom CSS

For a website called DBNET DesignBais creates a resource folder containing a sub-folder called *css*.

- DBNET URL: http://localhost/dbnet
- DBNET Physical folder: c:\inetpub\wwwroot\dbnet
- Create new RD Work folder: "sales"
- DesignBais creates themes folder: c:\inetpub\wwwroot\dbnet\res\sales\css
- This folder holds a number of preset theme files *.css

Copy any preset css file (e.g. navy.css) and paste with a new name in the same folder (e.g. sales.css): c:\inetpub\wwwroot\dbnet\res\sales\css\sales.css
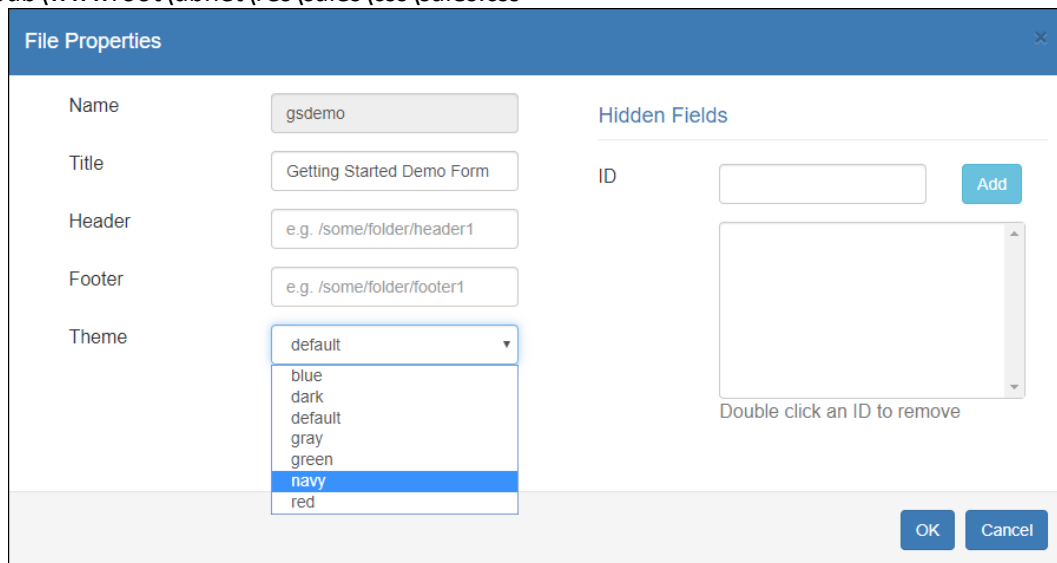
Select "sales.css" as your theme using the menu File > Properties > Theme in RD Forms Designer. You can then edit/modify sales.css (in Notepad) as necessary to make up your own theme.

Responsive Design Meta Tags

Responsive Design Meta tags are used to add:
- custom javascript
- custom css
- other custom html meta tags to individual DesignBais RD pages.

This is an example of how Meta tags are used:

We want users to see the prompt "hello world!" when they navigate to
http://localhost/dbnet/?dbpage=sales-fred

- DBNET URL: http://localhost/dbnet
- DBNET Physical folder: c:\inetpub\wwwroot\dbnet
- RD Work folder: sales
- RD page: fred
- URL of fred: http://localhost/dbnet/?dbpage=sales-fred

- Resource folder of "sales":   c:\inetpub\wwwroot\dbnet\res\sales
- Meta folder of "sales":   c:\inetpub\wwwroot\dbnet\res\sales\meta
-

Create new meta file for page "fred": c:\inetpub\wwwroot\dbnet\res\sales\meta\fred.meta
You can do this using Notepad.

The content of fred.meta can be a single line pointing to the location of a JS file as follows:
<script src="res/sales/meta/somescript.js"></script>
Note the src attribute starts with res/.

Create script file using Notepad: c:\inetpub\wwwroot\dbnet\res\sales\meta\somescript.js
Script file content: $(document).ready(function(){alert('hello world!')});

Users will see the prompt "hello world!" when they navigate to
http://localhost/dbnet/?dbpage=sales-fred

This is an example only as there is no need to use custom javascript to display a simple "hello world!" prompt in DesignBais.

This example shows how third party javascript libraries can be included and used in DesignBais if necessary.

Responsive Design Custom Fonts

Developers can use an existing font (ttf) files or download a new font file (ttf) from https://fonts.google.com/.

This is an example of using the custom font file named "grafiti.ttf" on a DesignBais RD page:

- DBNET URL: http://localhost/dbnet
- DBNET Physical folder: c:\inetpub\wwwroot\dbnet
- RD Work folder: sales
- RD page: fred
- URL of fred: http://localhost/dbnet/?dbpage=sales-fred
- Resource folder of "sales":   c:\inetpub\wwwroot\dbnet\res\sales
- CSS file of "sales": c:\inetpub\wwwroot\dbnet\res\sales\css\sales.css
- Fonts folder of "sales": c:\inetpub\wwwroot\dbnet\res\sales\fonts

Save the grafiti.ttf file in the fonts folder: c:\inetpub\wwwroot\dbet\res\sales\fonts\grafiti.css

Edit the custom css file: *c:\inetpub\wwwroot\dbnet\res\sales\css\sales.css* to add the following at the top of the document:

```
@font-face{
    font-family: grafiti;
    src: url('../fonts/grafiti.ttf');
}
 h3 {
     font-family: grafiti;
```

```
        }
```

The above method sets the designer's own elements to that font as well (in general this should not be a problem but it may be confusing for the developer). To avoid this the following method can be used:

```
#dmain *{
    font-family: scylla;
}
#dnav *{
    font-family: scylla;
}
#dfooter *{
    font-family: scylla;
}
```

Open RD page *fred*. Use menu *File > Properties > Theme* to select "sales.css".

Press CTRL+K to create an HTML segment. Click the text "Lorem Ipsum" and press CTRL+Q to edit.

In the HTML editor, click the text "Lorem Ipsum" and set it to "Heading 3" using the drop down. The text font will turn to grafiti.

Responsive Design – Inserting an Image

An image can be inserted into a HTML form element.

In Forms Designer insert a form row and press *Ctrl+K* to insert an HTML form element.

Right-click the row and highlight *Form Row* then select *Edit HTML [Ctrl+Q]*.

This will open the HTML Editor.
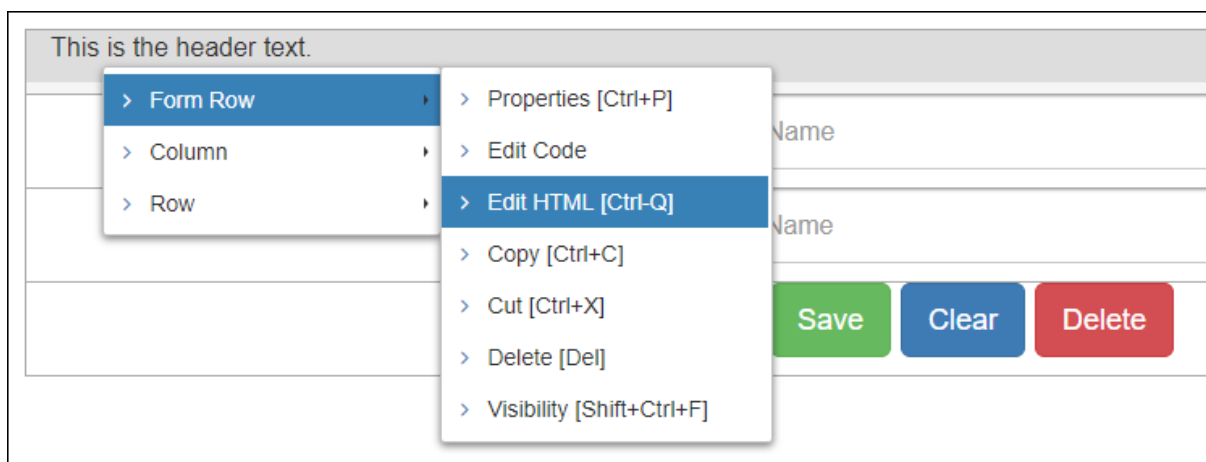
Click the *Source* button.
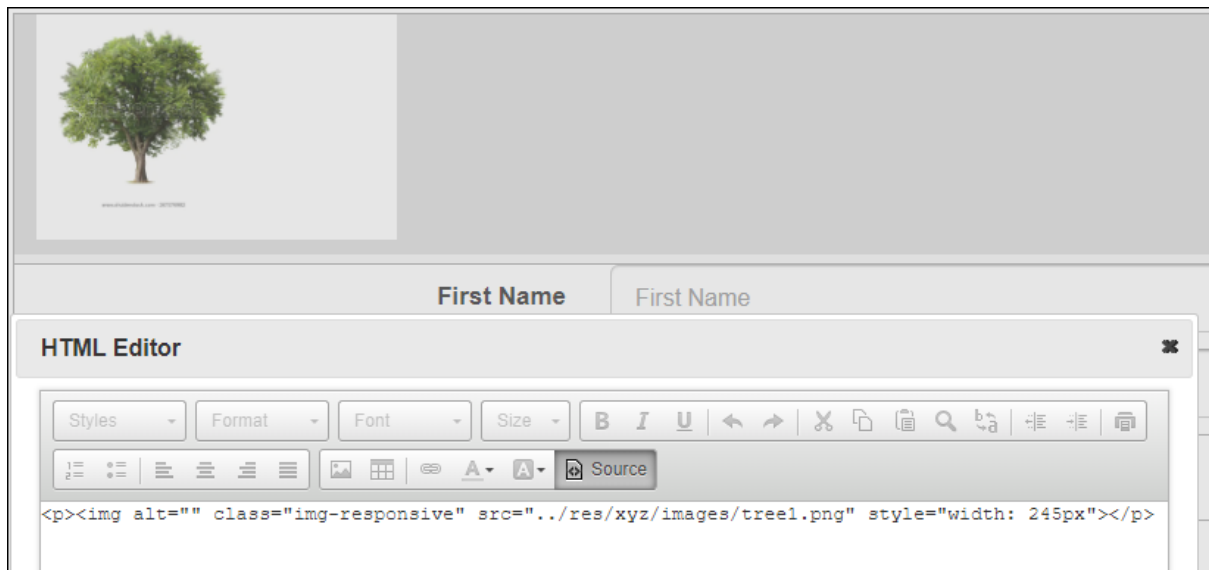
Enter the html string shown above. Note that:

- The name of your image must be prefixed with "../".
- In this example the name of the image is *tree1.png* .
- So the path to the image is "../res/xyz/images/*tree1*.png ".

<p><img alt="" class="img-responsive" src="../res/xyz/images/tree1.png" style="width: 525px"></p>

- The width should be set as required.
- *alt* can be set as required. This defines what will be displayed if the specified image cannot be found.

Images can be placed directly in the *images* sub-folder within the *res* folder.

It is good practice to create sub-folders within the *images* folder so that images for a particular project or application can be grouped together. This makes the task of moving an application to a different environment easier, since all required images can be moved by copying the sub-folder.

In the example shown to the right the images are placed in a sub-folder named *gs*.

Remote computer	C:\DBNET\res\dbweb\images\gs	
Name		**Object info**
gs1.PNG		PNG File
gs2.PNG		PNG File
gs2open.PNG		PNG File
gsButton.PNG		PNG File
gsButton2.PNG		PNG File
gsButton3.PNG		PNG File
gsButton4.PNG		PNG File
gsButtonGroup.PNG		PNG File
gsButtonGroup2.PNG		PNG File
gsFirstName.PNG		PNG File
gsFirstName2.PNG		PNG File
gsHTML.PNG		PNG File
gsHTML2.PNG		PNG File
gsLastName.PNG		PNG File
gsLastName2.PNG		PNG File
gsNewRow.PNG		PNG File
gsNewRow2.PNG		PNG File
gsTopMenu.PNG		PNG File
gsWorkFolder.PNG		PNG File
gsWorkFolder2.PNG		PNG File
gsWorkFolder3.PNG		PNG File

Responsive Design String Encoding

In order to include a string such as "../res" on, say, an output HTML element use the following:

```
<p style="font-size:0px;"><span style="font-size:16px;">.</span><span style="font-size:16px;">./res/WORK_FOLDER/images/IMAGE_NAME</span></p>
```

This gives:

../res/WORK_FOLDER/images/IMAGE_NAME

Responsive Design Top Menu

The top menu comprises:
- File
- After
- GridOff

File Options

File ▾	After ▾	GridOff

Open [Alt-O]

New Footer [Alt-F]

New Header [Alt-H]

New Page [Alt-P]

Save [Alt-S]

Save As [Alt-A]

Close [Alt-C]

Properties [Alt-R]

Refresh [Alt-E]

Preview [Alt-V]

Work Folder [Alt-W]

Publish [Alt-L]
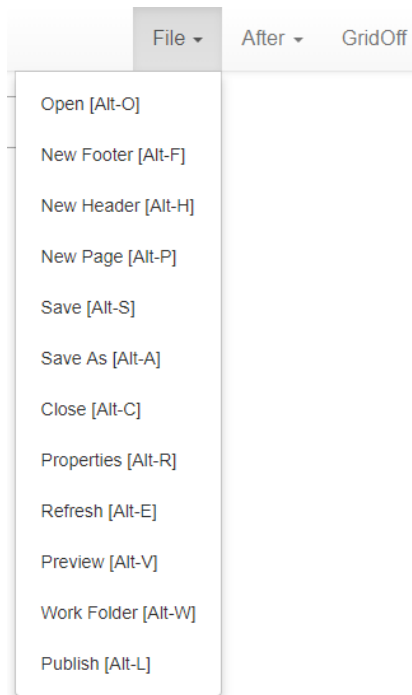
After Options
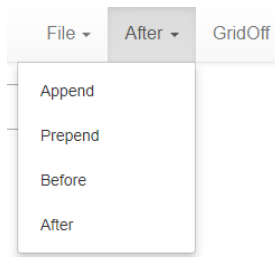
GridOff Option

This allows you to quickly preview the web page you are working on without the grid lines. As you design a page, you'll want to see how the page looks without the grid lines. Press *Esc* to exit this mode.

The File Option *Preview* [Alt V] can also be used but it's slower as it opens a new window. *Preview* is needed, however, if you've made changes to the header or footer and you wish to display the latest version of the entire set.

Top Menu Appearance with Narrow Browser Width

The top menu will contract to the *Hamburger* icon shown below when the browser width is reduced since DesignBais Responsive Designer is built as a responsive design web page.
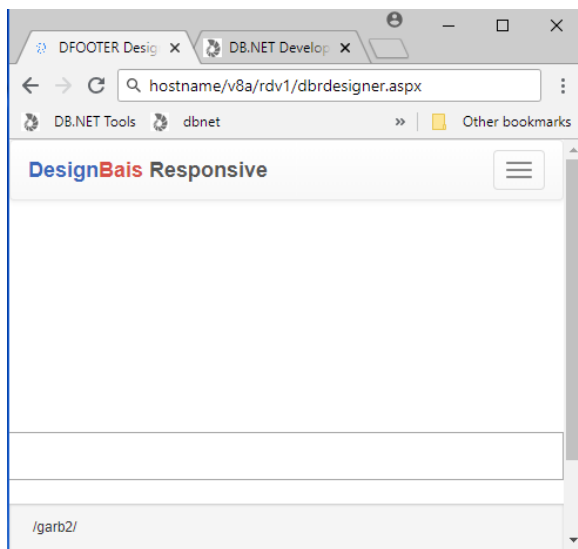
Clicking the menu icon displays the menu options down the left hand side of the page.

DesignBais Responsive ☰

File ▾

New Page [Alt-P]

Open [Alt-O]

New Footer [Alt-F]

New Header [Alt-H]

Save [Alt-S]

Save As [Alt-A]

Close [Alt-C]

Properties [Alt-R]

Refresh [Alt-E]

Preview [Alt-V]

Work Folder [Alt-W]

Publish [Alt-L]

After ▾

GridOff
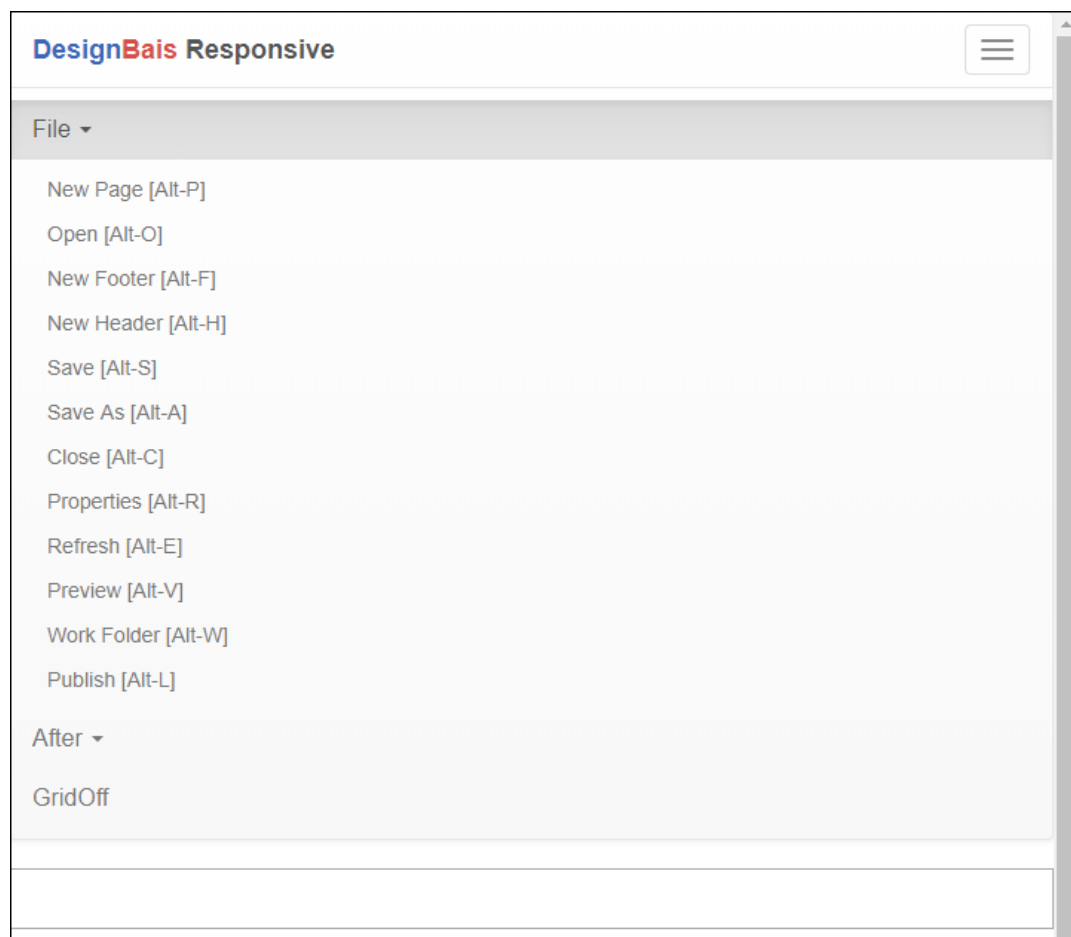
Responsive Design Components, Parts and Functions

The basic construct in a Responsive Design form is a row. So when the Designer is entered the initial display is one row.

The Designer is driven by positioning the mouse and right-clicking. The available options displayed by the right-click will vary based on position.

Initially it is possible to add new rows, or to insert columns but before any form elements can be created it is essential to create one or more *Form Rows*. A form row is the container in which a form element lives.

Multiple *Form Rows* can be created within one *Column*.
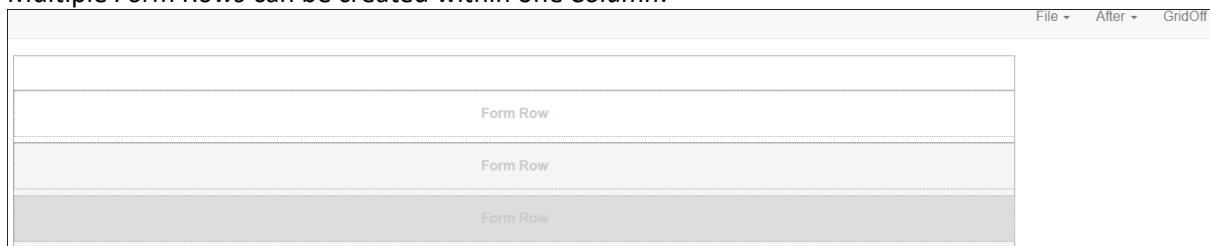
Form elements can be added to any of these form rows.
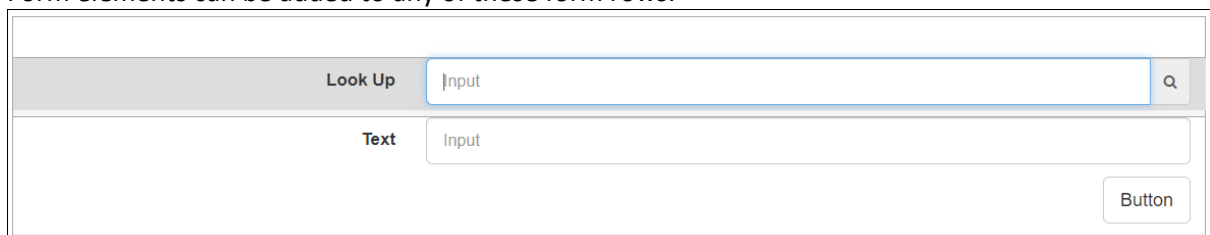
All the above form elements are in a single column. Therefore if two columns are required for the form row containing the button the developer might think to insert a column in the button form row. So right-click and the button form row, select *Row* then select *Insert Column*.
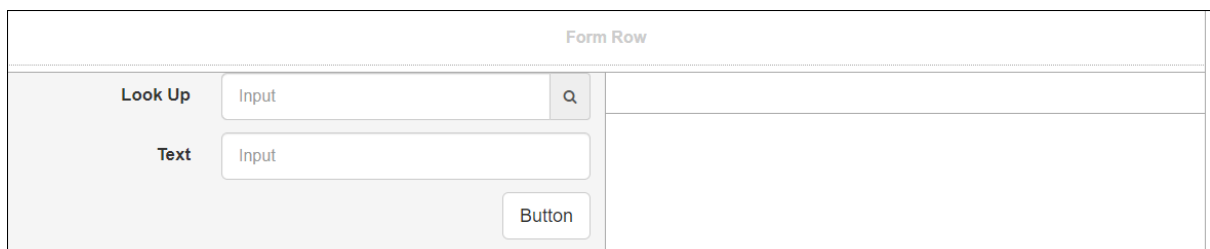
This results in ALL form elements in the *row*, not just the *form row*, being moved into the left column of two columns in the row.

In order to achieve the desired result it is necessary to add a new *row*, not a *form row*, and copy the button into the new row.
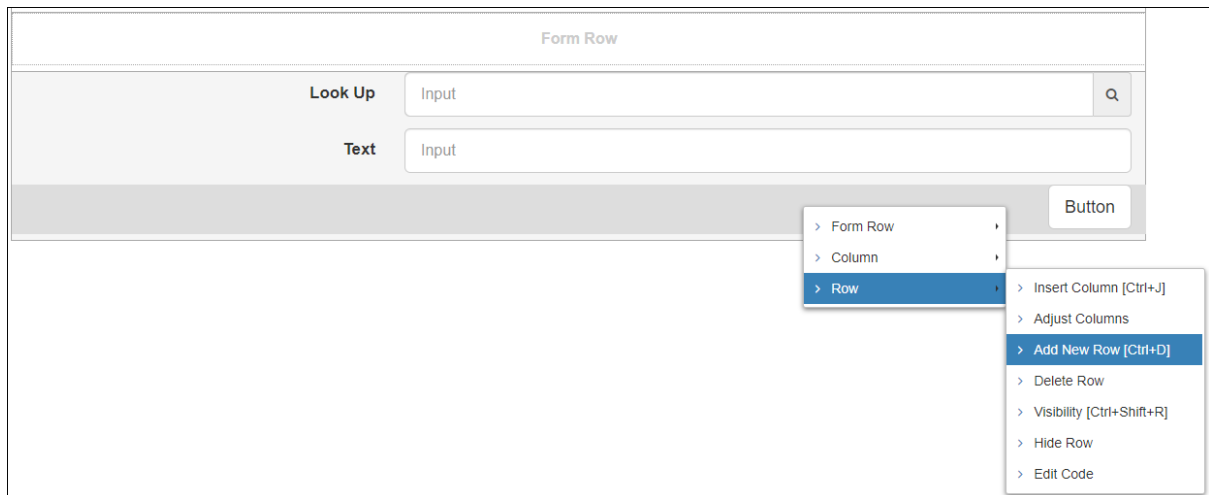
After adding the new *row* then insert a *form row* into the new row. Then copy the button element.
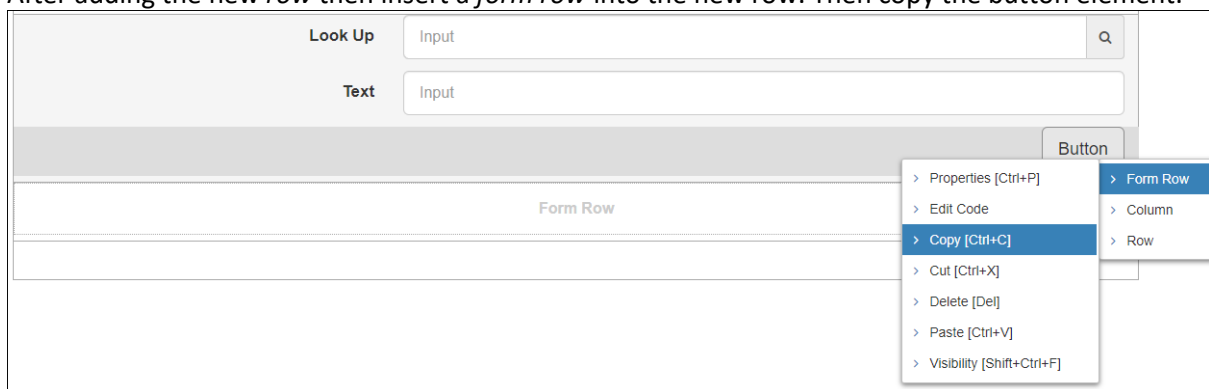
Then paste the button element into the new *form row*.
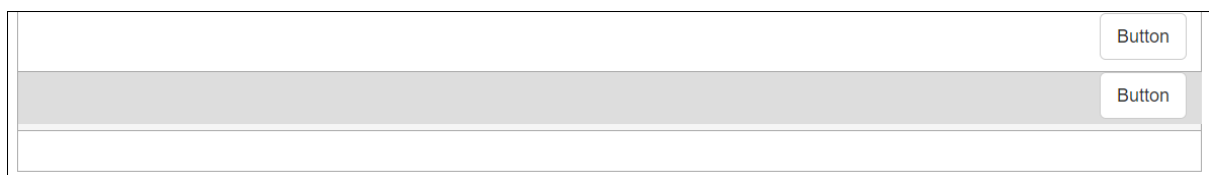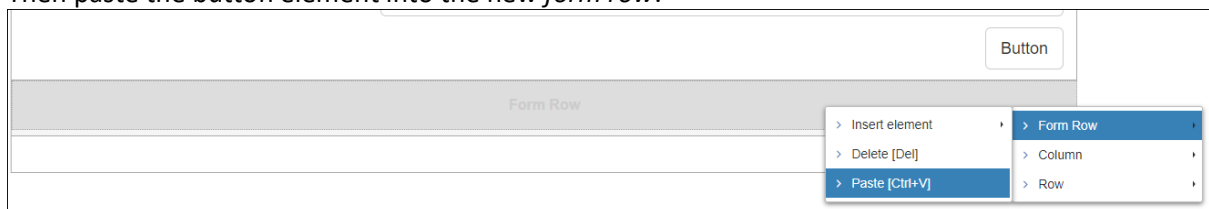
Now insert a *column* into the *form row* in the new *row.*
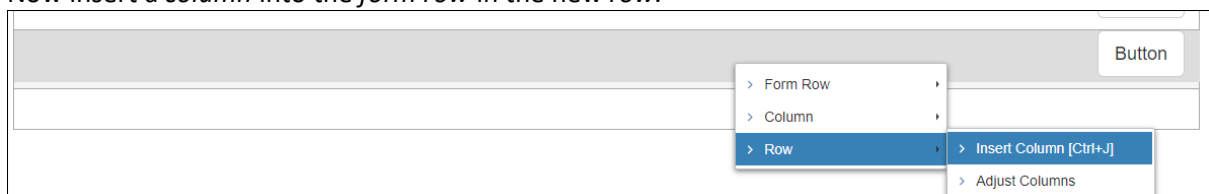
Look Up	Input	Q
Text	Input	
		Button
	Button	

You can now insert a new button element into the right-hand column of the new row so that there are two buttons on the same row.

Look Up	Input	Q
Text	Input	
	Button	Button 2

Keep in mind that you can cut and paste in order to move form elements.

Client Code	Client Code	
Amount	Amount	
Amount 4 dec pl	Amount 4 dec pl	
	Lorem ipsum..	
		Save

If you want, for example, to move the *Client Code* and *Amount* fields to the right hand side then simple focus on the form element, press Ctrl X, focus on the empty column to its right and press Ctrl V.

		Client Code	Client Code
Amount	Amount		
Amount 4 dec pl	Amount 4 dec pl		
	Lorem ipsum..		
			Save

		Client Code	Client Code
		Amount	Amount
Amount 4 dec pl	Amount 4 dec pl		
	Lorem ipsum..		
			Save

COMPONENTS

- File
- Lookup
- Address
- Checkbox
- Radio
- Textarea
- Select
- HTML
- Button
- Textbox (can be one of the following types):
    - Text
    - Number
    - Date
    - Time
    - Datetime-local
    - Month
    - Email
    - Phone
    - Password
    - Search
    - Url

COMPONENT PARTS (targets)

Each RD component may have one or more of these "parts" as applicable.

- row
- col
- formRow
- element
- label
- option
- optionlabel
- error
- tipicon
- tiptext
- help
- segment

Diagrams for Textbox and Radio are shown below. Other RD elements follow a similar pattern:

TEXTBOX targets
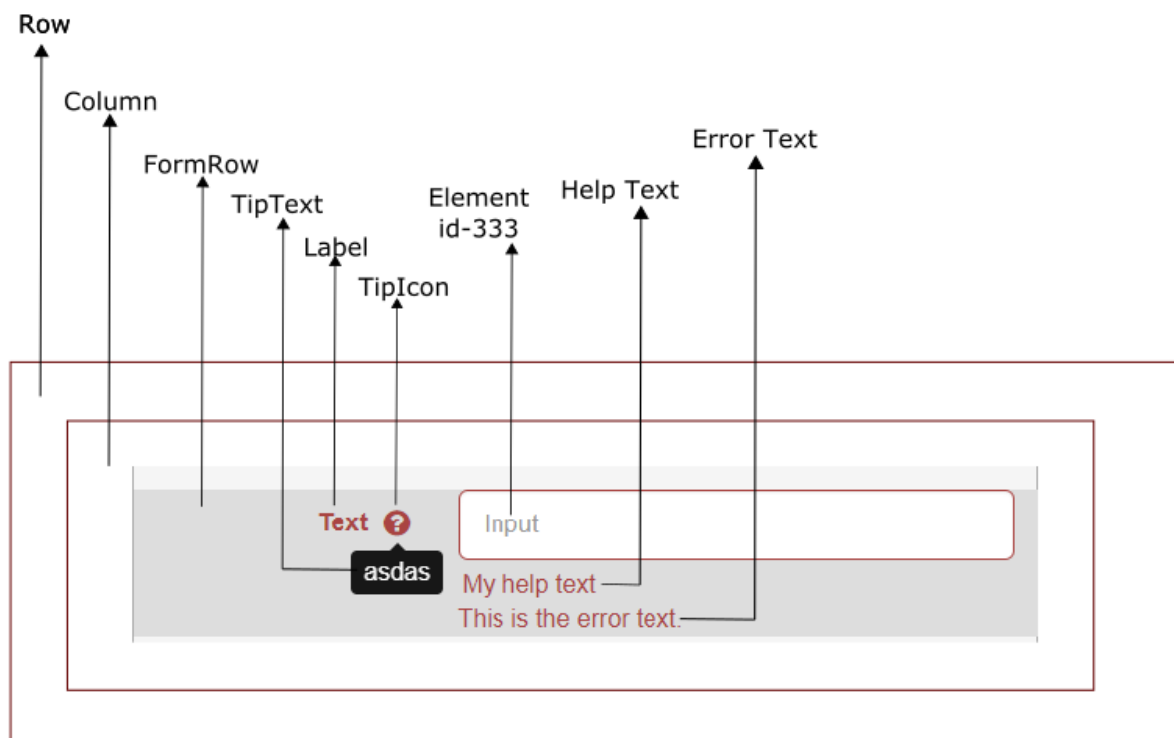
RADIO targets
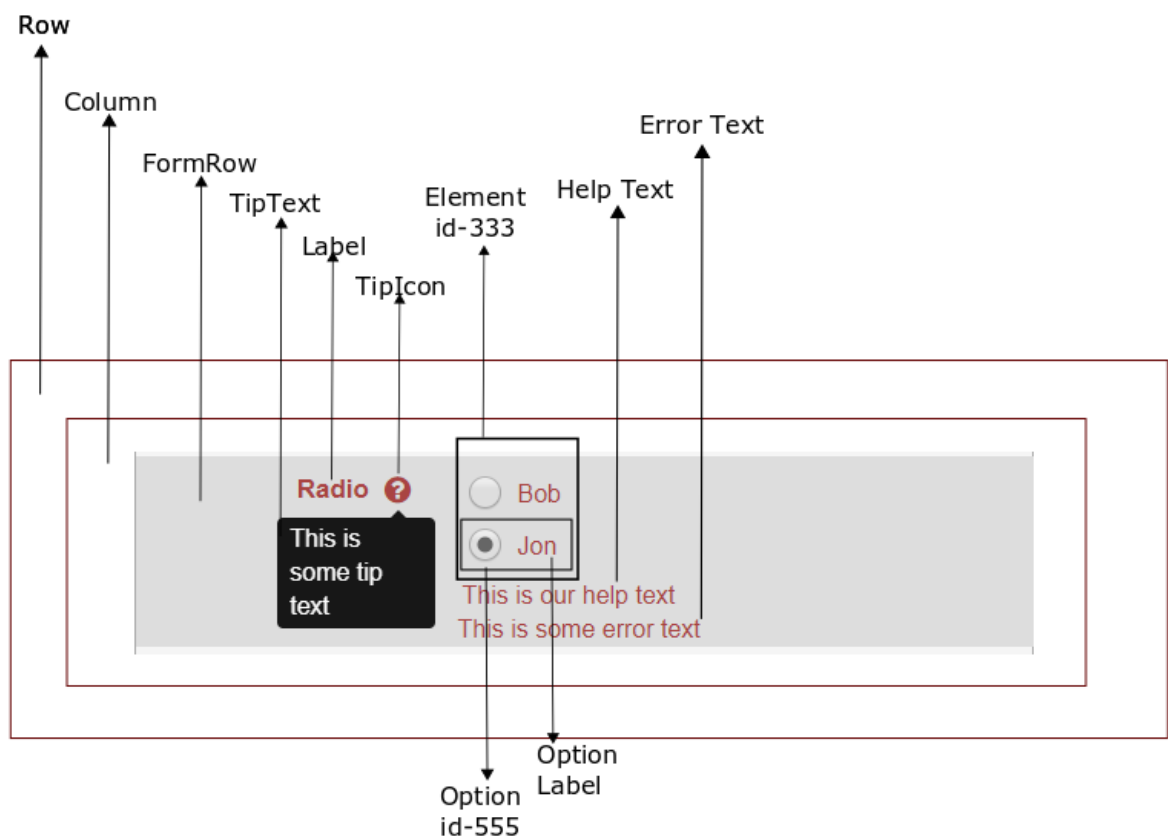
FUNCTIONS

Refer to the following section *Modifying a Responsive Design Form at Runtime* which describes the method of programmatically setting RD form elements.

Modifying a Responsive Design Form at Runtime

AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page. Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.

Use the DesignBais subroutine DBI.G.AJXCMD to modify form elements at runtime. This subroutine can be called from the event processing routine. It has two arguments.

DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)

Note: this subroutine and the common variable DBAJXCMD are documented in the DesignBais Reference Manual.

AJX.FUNC – either the full Ajax command or the abbreviation is valid:

Command	Abbreviation	Description
rdHide	HD	Hide an element on an RD form
rdShow	SH	Show an element on an RD form
rdSetText	ST	Set text in an element on an RD form
rdSetVal	SV	Set the value in an element on an RD form
rdSetStyle	SS	Set the style of an element on an RD form
rdSetAttribute	SA	Set an attribute
rdRemoveAttribute	RM	Remove an attribute
rdLookupJSON	LJ	JSON lookup
rdShowConfirm	SC	display a "confirm" dialog
rdShowModal	SM	displays an RD row as a modal form
rdHideModal	HM	hides an RD row displayed as a modal form
rdAlertBox	AB	display message using the browser's native alert box
rdShowMessage	SMSG	display a standard message panel on the page
rdHideMessage	HMSG	hide a standard message panel on the page
rdHideMessageAll	HMSGA	hide all message panels on the page
rdScrollToID	STID	scroll page to bring the row of the element into view
rdShowAllRows	SAR	display all rows of header, footer or body if hidden
rdHideAllRows	HAR	hide all rows of header, footer or body if displayed
rdShowSection	SSN	show header, footer or body
rdHideSection	HSN	hide header, footer or body
rdSetElementValue	SEV	directly set the value of any element
rdNavigate	NAV	navigate to the specified page
rdDeleteTableRow	DTR	delete row from table (implement as shown below)
jqsv	JQSV	jquery set value
jqst	JQST	jquery set text
jqshtml	JQSHTML	jquery set html
jqsh	JQSH	jquery show()
jqhd	JQHD	jquery hide()
jqsa	JQSA	jquery set attribute
jqra	JQRA	jquery remove attribute
jqss	JQSS	jquery set css
addCode	AC	add code
addScript	AS	add script

addCSS	CSS	add CSS to the DesignBais style sheet
setSlider	SL	create a slider control
dbslidePanel	SP	create a slide panel
dbCarousel	CAR	create an image carousel
addEvent	AE	add event
customAttribute	CA	set custom attribute (HTML attribute) at run time
removeCustomAttribute	RCA	remove custom attribute at run time
dbDayPicker	DP	day (date) picker (RD forms only)
dbDayPickerSetHTML	DPSH	day (date) picker (RD forms only)
showSpinner	SSPN	show spinner text (can be used in non-RD forms)
hideSpinner	HSPN	hide spinner text (can be used in non-RD forms)
authenticated	AUTH	set a user as authenticated (allow access to uploads) (can be used in non-RD forms)
setSelectionRange	SSR	set the start and end selection for input field (can be used in non-RD forms)

AJX.ARG

For the following functions only attributes 1 through 5 are required:

rdHide	HD
rdShow	SH
rdSetText	ST
rdSetVal	SV
rdSetStyle	SS
rdSetAttribute	SA
rdRemoveAttribute	RM
rdLookupJSON	LJ

Attribute	*Content*	*Format*
1	RD form element id	multi-valued
2	Type of RD Form element (see list below)	multi-sub valued
3	CSS property	multi-sub valued
4	CSS property setting or value	multi-sub valued
5	mSec delay time	multi-sub valued

Type of RD Form element

row
col
formRow
element
label
option
optionlabel
error
tipicon
tiptext
help
segment   (use for HTML output element)

For other functions the attributes from 5 upwards contain function-specific values:

Attribute	Content	Format
1	RD form element id	multi-valued
2	Type of RD Form element (see list above)	multi-sub valued
3	CSS property	multi-sub valued
4	CSS property setting or value	multi-sub valued
5	mSec delay time	multi-valued
6	button events	multi-valued
7	close events	multi-valued
8	response handling subroutine	multi-valued

Example of Usage rdSetStyle:

```
AJX.ARG = ''
IF DBVALUE = 'N' THEN
  CLR1 = 'thistle'; CLR2 = 'yellow'; WID3 = 'thick'
END ELSE
  CLR1 = '#8e578e'; CLR2 = '#ff4d4d'; WID3 = 'thin'
END
AJXCMD = 'SS'                        ;* rdSetStyle
AJX.ARG<1,-1> = 'd948937'
AJX.ARG<2,-1> = 'element'
AJX.ARG<3,-1> = 'background-color':SVM:'border-width'
AJX.ARG<4,-1> = CLR1:SVM:WID3
AJX.ARG<1,-1> = 'd978206'
AJX.ARG<2,-1> = 'element'
AJX.ARG<3,-1> = 'color':SVM:'background-color':SVM:'border-width'
AJX.ARG<4,-1> = 'red':SVM:CLR2:SVM:WID3
CALL DBI.G.AJXCMD(AJXCMD,AJX.ARG)
```

Developers should note the effect of using this method to modify a form at runtime. Referring to the following example:

Developers should not change the first 4 lines (dbrow, dbcol, dbform, dbformgroup) unless there is a specific reason to do so. In order to just set the colour for the 5th line (segment) use:

```
AJX.ARG = ELEMENT.ID
AJX.ARG<2, 1> = 'element'
```

which will set the color of the div with ID = ELEMENT.ID

This code changes the HTML of the row, and should be avoided:

```
AJX.ARG = ELEMENT.ID
AJX.ARG<2, 1> = "row"
AJX.ARG<3, 1> = "color"
AJX.ARG<4, 1> = CSS.COLOR
CALL DBI.G.AJXCMD('SS', AJX.ARG)
```

This code only changes the color for the html segment with the specified id, and is the recommended method:

```
AJX.ARG = ELEMENT.ID
AJX.ARG<2, 1> = "element"
AJX.ARG<3, 1> = "color"
AJX.ARG<4, 1> = CSS.COLOR
CALL DBI.G.AJXCMD('SS', AJX.ARG)
```

Example of Usage customAttribute:

Set the multi-select property of a select element:

```
AJX.ARG = 'BKT.PROP.Q47A'
AJX.ARG<3> = "multiple"
AJX.ARG<4> = "multiple"
CALL DBI.G.AJXCMD("CA",AJX.ARG)
```

Example of Usage of the common variable DBAJAXCMD:

MYCLASS="dotempty"
ID="step1"

THIS.CMD = 'document.getElementById("':ID:'").className="':MYCLASS:'"'
DBAJAXCMD<-1> = THIS.CMD:';   '

Note that the final few characters of the string are the DesignBais end command flag:
    ";" + 3 spaces.

Example of Usage showSpinner:

Use the 'SSPN' (or 'showSpinner') command to display text above the server busy spinner

```
AJX.ARG = ""                               (element ID is not required)
AJX.ARG<2>     = text to display
AJX.ARG<3>     = "true" or "false" (used to optionally blur and block the background)
CALL DBI.G.AJXCMD("SSPN",AJX.ARG)
```

Example of Usage authenticated:

Authenticate a user to access the website *uploads* folder.

AJX.FUNC = "AUTH" or "authenticated"
AJX.DATAIN = "Y" or "y" or "TRUE or "true" or 1

```
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.DATAIN)
```

Will set SESSION.REC<75> = "Y". Set AJX.DATAIN to any other value will clear it.

msec

In the following function descriptions the argument *msec* is the animation speed.

- msec = slide up/down animation speed

Any function that has an "msec" parameter can do sliding up or sliding down animation.

e.g. You can use *rdHide* and *rdShow* to hide or show a row. If you set the *msec* value to say 300, the row will disappear or appear by sliding up or down in 300 milliseconds. That is obviously a slow animation speed but demonstrates the usage of *msec*.

msecDelay

This is a delay period in milliseconds. The default is 0 (no delay).

The delay may be needed in rare cases such as delaying the disabling of an element until after other cascaded functions have been completed.

e.g. (cascaded function calls):
rdShow("myInstructionsID", "row", 3000);   rdRemoveAttribute("mySubmitButtonID", "element", "disabled", 4000)

A row is displayed in 3 seconds. 1 second after that an element is enabled in another row.
This could be a scenario where an element is not enabled before displaying the content of a row which may contain a set of instructions.

Key to Function Descriptions

id   This is the *id* assigned to the form element in the RD form.

target   The form element target such as row, column or tiptext. Refer to the section *Responsive Design Components, Parts and Functions* for the list of targets.

underscore Arguments with an underscore are optional.

The following functions can be executed as javascript or using the DesignBais DBI.G.AJXCMD subroutine.

rdHide(id, target, msec)

Hide a form element.
Msec recommended values: 0-1000 msec, default is 0

Example:
```
AJX.ARG = 'utility'            ;* form element id
AJX.ARG<2>='element'           ;* target
AJX.ARG<1,-1> = 'testHTML';* another form element
AJX.ARG<2,-1>='element'        ;* another target
CALL DBI.G.AJXCMD('HD',AJX.ARG)
```

Or using subvalues to define multiple targets for the one form element:

```
AJX.ARG = 'bsbNumber'          ;* form element id
AJX.ARG<2,1>='element'         ;* target
AJX.ARG<2,1,-1>='label'        ;* target
CALL DBI.G.AJXCMD('HD',AJX.ARG)
```

Developers should note that an attempt to "show" a row that is partly hidden will fail.

Consider this example where in the AFTER DISPLAY event the form element *bsbNumber* is hidden by hiding the row containing the field.

In the snip below the *bsbNumber* field on the second row is not visible. The field labeled *BSB Number*, on the first row, is visible. It is lookup field used to search for an existing record.

Clicking the *Add* button hides the *BSB Number* lookup field and shows the *bsbNumber* field.

```
*
AFTER.DISPLAY:
*
      BEGIN CASE
         CASE SCREEN.NO = "MAINT" AND FILENAME = "ABANK"
    AJX.ELEM = 'bsbNumber'
            AJX.TARG = 'row'
            AJX.FUNC = 'HD'; GOSUB SHOW.HIDE

*
BUTTON:
*
      BEGIN CASE
         CASE SCREEN.NO = "MAINT" AND FILENAME = "ABANK"
            BEGIN CASE
               CASE EVENTSOURCE = "B.ADD"
                  * hide the lookup and reveal the input field
                  AJX.ELEM = 'lookupBsb'
                  AJX.TARG = 'row'
                  AJX.FUNC = 'HD'; GOSUB SHOW.HIDE
                  AJX.ELEM = 'bsbNumber'
                  AJX.TARG = 'row'
                  AJX.FUNC = 'SH'; GOSUB SHOW.HIDE

*
SHOW.HIDE:
*
      AJX.ARG = AJX.ELEM        ;* form element id
      AJX.ARG<2,1>=AJX.TARG     ;* target
      AJX.ARG<5,1> = 900        ;* msec delay
      CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
      RETURN
```

If, instead of hiding the **row** in the AFTER DISPLAY, only the visible components of the field are hidden (the *element* and the *label*), using this code below for example:

```
AJX.ARG = 'bsbNumber'          ;* form element id
AJX.TARG   ='element'          ;* target
AJX.TARG<1,1,-1>='label'       ;* target
AJX.FUNC = 'HD'; GOSUB SHOW.HIDE
```

then using the code below to show the field again will not work:

```
AJX.ELEM = 'bsbNumber'
AJX.TARG = 'row'
AJX.FUNC = 'SH'; GOSUB SHOW.HIDE
```

This is because the row is not fully hidden, hence attempting to show the row will not have any effect.

rdShow(id, target, <u>msec</u>)

Display a form element.
Msec recommended values: 0-1000 msec, default is 0

rdSetText(id, target, some_text)

Provides a means to set or change the text of a form element.

```
AJX.ARG = 'address'
AJX.ARG<2>='tiptext'
AJX.ARG<4> = 'New set text for tip'
CALL DBI.G.AJXCMD('ST',AJX.ARG)
```

rdSetVal(id, value)

Provides a means to set or change the value of a form element.
Example:
> Set the text of a button: rdSetVal("b-button1", "Click Here")
> Where 'b-button1' is the form element id of the button.

```
AJX.ARG = 'b-button1'
AJX.ARG<4> = Click Here'
CALL DBI.G.AJXCMD('SV',AJX.ARG)
```

rdSetStyle(id, target, style_attribute, value)

Provides a means to set or change the value of a style property of a form element.
Example (setting both Style and Attribute):
```
RD.FUNC = 'SS':VM:'SA'
RD.INP = 'name':VM:'name'
RD.INP<2> = 'element':VM:'tiptext'
RD.INP<3> = 'backgroundColor':VM:'title'
RD.INP<4> = 'lightblue':VM:"Bob\'s tip"
CALL DBI.G.AJXCMD(RD.FUNC,RD.INP)
```

rdSetAttribute(id, target, element_attribute, value, <u>msecDelay</u>)

Set the value of a target attribute in a form element.
msecDelay recommended values: 0-1000 msec, optional argument, default is 0

Example:
```
RD.FUNC = 'SA'
RD.INP = 'clientCode'
RD.INP<2> = 'tiptext'
RD.INP<3> = 'data-original-title'
RD.INP<4> = "Bobs tip"
CALL DBI.G.AJXCMD(RD.FUNC,RD.INP)
```

Setting the *element_attribute* to a value of "nofilter" provides a means of controlling the filtering of the results from an onload event in a lookup element or dropdown list. Setting "nofilter" to "true" results in all elements being returned rather than just those that match the filter.

> rdSetAttribute(elementID,"element","nofilter", "true");

Example:

Setting text overlay associated with a form element. Use this feature to, for example, display overlay text for the DesignBais server-busy spinner when a button is clicked. The text is displayed only when the event source (e.g. a button) has the attribute "loadingtext".

rdSetAttribute(element_id, "element", " loadingtext", "Loading your delivery calendar");
(Note: only the parts shown in red are variables)

Example:
Disabling a button. The id of the button is 'payment'.

```
AJX.CMD = 'SA'
AJX.PARMS = 'payment'
AJX.PARMS<2> = 'element'
AJX.PARMS<3> = 'disabled'
AJX.PARMS<4> = 'disabled'
CALL DBI.G.AJXCMD(AJX.CMD,AJX.PARMS)
```

Setting the mandatory property using rdSetAttribute

```
IS.REQD = (DBWORK<GCL.ACCRED.WK> = 'Y' AND DBRECORD<GCL.ACCRED.DOC> = "")→ this is the condition
AJX.ARG = ''
AJX.ARG<1, 1> = 'docAccrAttachFileDUpload'                        → this is the element id
AJX.ARG<2, 1> = 'element'
AJX.ARG<3, 1> = 'filemandatory'
IF IS.REQD THEN WORD = 'true' ELSE WORD = 'false'
AJX.ARG<4, 1> = WORD
CALL DBI.G.AJXCMD('SA', AJX.ARG)
```

rdRemoveAttribute(id, target, element_attribute, msecDelay)

Remove the value of a target attribute in a form element.
msecDelay recommended values: 0-1000 msec, optional argument, default is 0

Example:
Enabling a button. The id of the button is 'payment'.

```
AJX.CMD = 'RM'
AJX.PARMS = 'payment'
AJX.PARMS<2> = 'element'
AJX.PARMS<3> = 'disabled'
CALL DBI.G.AJXCMD(AJX.CMD,AJX.PARMS)
```

rdLookupJSON(id, keys/values, descriptions/labels)

Create a JSON array of name-value pairs.
List of dropdown keys/values in argument 3 and descriptions/labels in argument 4.
Values can contain up to a maximum 100 double-pipe separated ids.

```
AJX.CMD = 'LJ'
AJX.ARG = ''
AJX.ARG<3> = 'key1.1||key1.2'     :VM:'key2.1||key2.2'
AJX.ARG<4> = 'label1.1||label1.2':VM:'label2.1||label2.2'
CALL DBI.G.AJXCMD(AJX.CMD,AJX.ARG)
```

rdShowConfirm(confirmEventId, message, header, buttons, buttonStyles, buttonEvents, closeEvent, subroutineName)

Displays a "CONFIRM DIALOG" with the following parameters:

confirmEventId	Create a hidden input field with an ID in your RD form designer (File > Properties > Hidden Fields). Link that hidden field in DesignBais. Feed the ID of that hidden field as *confirmEventId* to this function. This way, you'll be able to receive the button click events from this dialog box. The event source will be the ID of your hidden field. The event type will be "rdConfirm-n" where n is the button order from left to right. e.g. for YES||NO, the event types will be rdConfirm-1 and rdConfirm-2 respectively. rdConfirm-0 means the dialog has been closed using the (X) button on the top right corner.
message	message text for the dialog
header	header for the dialog
buttons	Button captions delimited using two pipes (||). e.g. "YES||NO". Default: a single "OK" button.
buttonStyles (optional)	One of the following constants: "default", "primary", "danger", "info", "warning", "success" , "link" for each button separated using pipes "||". e.g. "primary||danger". If missing or set to "" then the button style defaults to "default" style.
buttonEvents	1=serverHitAndClose or 0=JustClose. || delimited for each button; e.g. "0||1". Default: 1
closeEvent	1=serverHitAndClose or 0=JustClose. A single value for the dialog box (X) button. Default: 1
subroutine name	the name of the subroutine that is to process the response

Example:
rdShowConfirm("dHidden001","Please let us know what you think. Is Bulentium an element?", "Is it an element?", "YES\|\|NO\|\|WHAT?", "primary\|\|warning\|\|danger", "1\|\|1\|\|0", 1);

buttons, *buttonStyles*, *buttonEvents*   VM and SVM characters are converted to "\|\|"

The developer must use the Form Data Link process to link the hidden form element to a database field and this field must have a Process After specified. The subroutine named in the Process After will be called to handle the DIALOG event generated when a button is clicked. In the DIALOG event the EVENTSOURCE will be set to the element id of the hidden field, PROCESS.PARAMETER will contain the sequence number of the button that is clicked ordered from left to right.
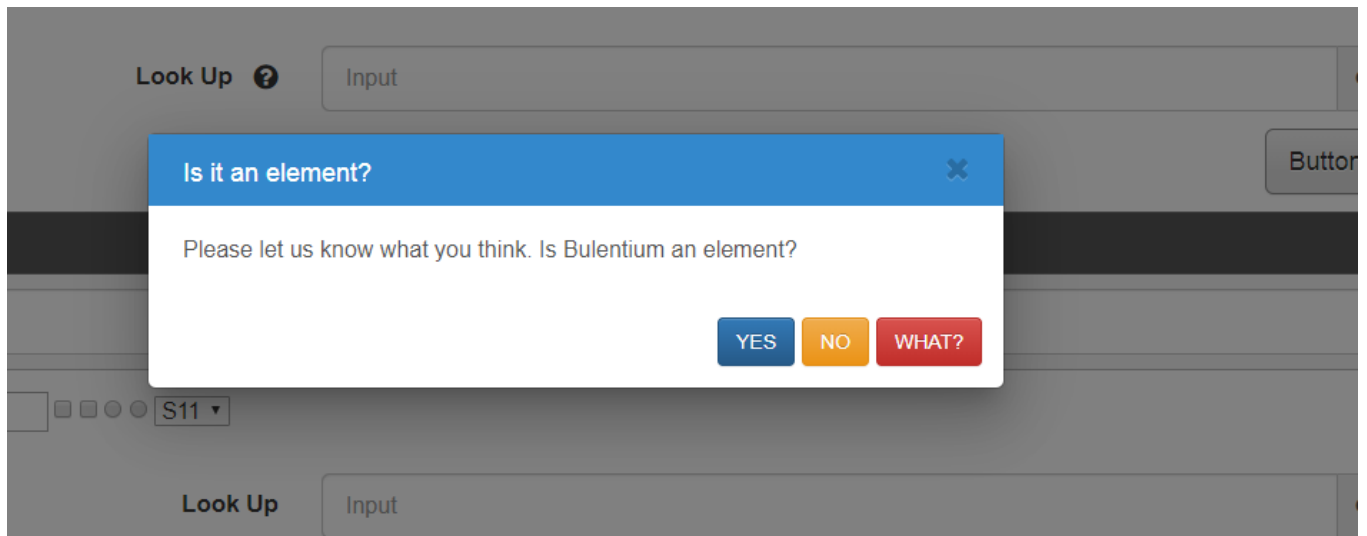
Example of basic code for *rdShowConfirm*:

```
AJX.FUNC = 'SC'
AJX.ARG = ''
AJX.ARG<1> = EVENTSOURCE
AJX.ARG<2> = 'Are you sure that the name is to be changed?'
AJX.ARG<3> = 'Client Name Change Request'
AJX.ARG<4> = 'Yes||No'
AJX.ARG<5> = 'primary':VM:'danger'
AJX.ARG<6> = '0||1'
AJX.ARG<7> = '0'
AJX.ARG<8> = 'SUBR.NAME'
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

This is another example that demonstrates a method of handling the display of a confirmation message in different ways depending on whether the user is in traditional DesignBais or in Responsive Design:

```
CASE DELETE.TYPE = "D"
    IF RDMODE THEN
        AJX.ARG = ''
        AJX.ARG<1> = 'dbDialog'
        AJX.ARG<2> = "Are you sure you would like to delete row [":DBMVCOUNT:"]?"
        AJX.ARG<3> = 'Table Row Delete Request'
        AJX.ARG<4> = 'Yes||No'
        AJX.ARG<5> = 'danger':VM:'primary'
        AJX.ARG<6> = '1||1'
        AJX.ARG<7> = '1'
        AJX.ARG<8> = 'SUBR.NAME'
        CALL DBI.G.AJXCMD('SC',AJX.ARG)
    END ELSE
        DBDIALOG.TEXT = "Are you sure you would like to delete row [":DBMVCOUNT:"]?"
        CALL DBI.G.GLOSSARY(DBDIALOG.TEXT)
        DBDIALOG.BUTTONS = 4
        DBDIALOG.PROGRAM = "DBDELETEMVPROCESS"
        DBDIALOG.PARAMETER = "DBDELETEMVPROCESSRESPONSE"
    END
    GOSUB …

END
```

The following example shows how to process the response in the DIALOG event:

```
RDSHOWCONFIRM:
    AJX.ARG = 'confirmSave'
    AJX.ARG<2> = 'Do you wish to save the current record':SVM:'Record Id:<b>':DBKEY:'</b>?'
    AJX.ARG<3> = 'Confirm Save of Current Record'
    AJX.ARG<4> = 'Yes||No'
    AJX.ARG<5> = 'primary':VM:'danger'
```

```
                AJX.ARG<6> = '1||1'
                AJX.ARG<7> = '1'
                AJX.ARG<8> = 'SUBR.NAME'
                *
                CALL DBI.G.AJXCMD('SC',AJX.ARG)
                RETURN

        DIALOG:
            BEGIN CASE
                CASE SCREEN.NO = 'CATALOG'
                    *
                    * In the DIALOG event the EVENTSOURCE contains the name of the
                    * hidden field used in the rdShowConfirm call eg: confirmSave
                    * PROCESS.PARAMETER contains the option that was clicked eg: 1
                    *
                    BEGIN CASE
                        CASE EVENTSOURCE = 'confirmSave'
                            BEGIN CASE
                                CASE THIS.PARAMETER = 1;* Yes was clicked
                                    NULL;* no action - remain as you were!
                                CASE THIS.PARAMETER = 2;* No was clicked
                                    DBKEY = ''
                                    EVENTSOURCE = 'B.CLEAR'
                                    GOSUB BUTTON
                            END CASE

                    END CASE

            END CASE
            RETURN
```

rdShowModal(id, header, closeEvent)

Displays an RD row as a modal form.

id	ID of an arbitrary element within the row designated as modal
header	header text for the modal row when it displays
closeEvent	1=serverHitAndClose or 0=JustClose.
	A single value for the dialog box (x) button.
	Default 1

In RD Designer, right click a row and select visibility > Display Mode > Modal.
This flags the row to be displayed as a modal form in run time. In RD Designer, the modal row is
indicated with a light blue color when another row is clicked.
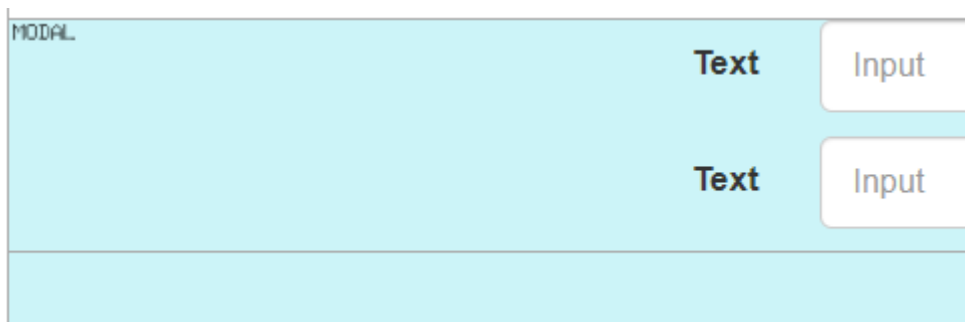
rdShowModal("modalFields", "This is My Title!!", 1)

```
        CASE EVENTSOURCE = "B.BTNSHOWMODAL"
            AJX.FUNC = "SM"
            AJX.ARG = "modalFields"
            AJX.ARG<2> = "This is My Title!!"
            AJX.ARG<3> = 1
            CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

The modal form in run time:
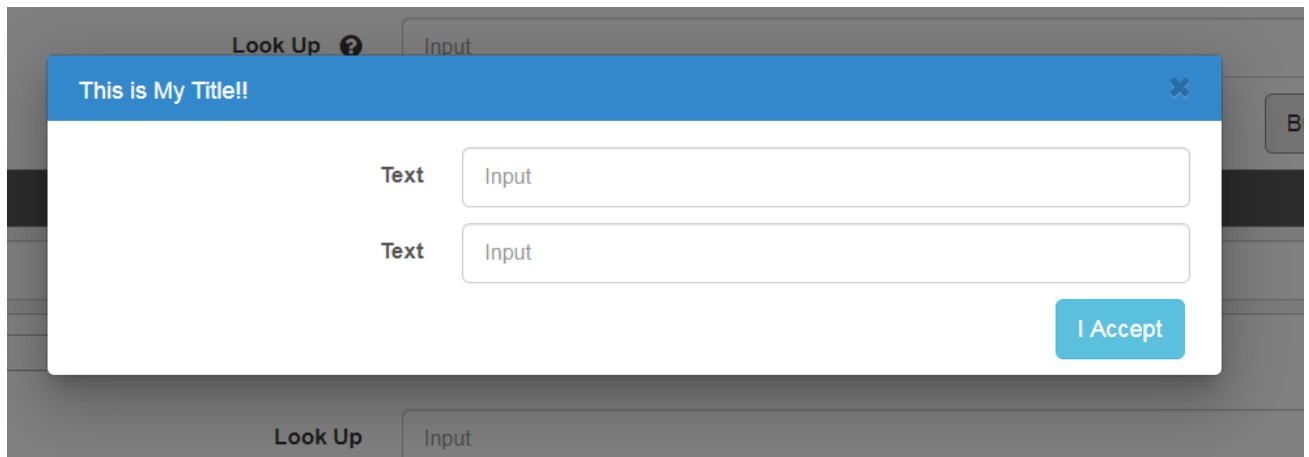
rdHideModal(id)

```
AJX.FUNC = "HM"
AJX.ARG = "modalRow"
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

rdAlertBox(inText)

Display a message using the browser's native alert box.
The native alert box has drawbacks:
- blocks the thread; it cannot be closed programmatically.
- some popup blockers disable alert boxes
- Looks different on different platforms/browsers and can be ugly.

Note that characters within the message text are converted as follows:
subvalue mark char(252) → break row
apostrophe ' → break row
char(13) → break row

```
AJX.FUNC = "AB"
AJX.ARG = 'Test New rdAlertBox.':SVM:SVM:'In modalEmail close via X'
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```
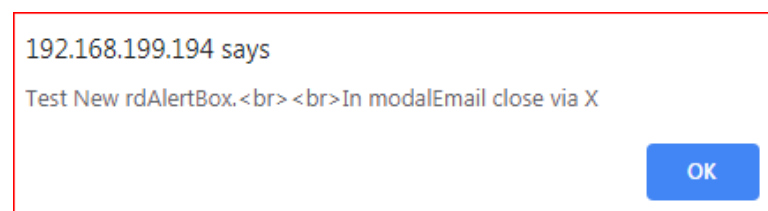

rdShowMessage(messageEventId, message, header, messageList, showClose, closeEvent, position, sBefore, style, msec)

Displays a standard message panel on the page at the required row position. Page scrolls to the message panel when displayed. Note that to link to a null id set messageEventId to $BLANK$.

messageEventId	ID of a field created in RD Form Designer (an HTML output element will do)
message	the message text.
header	the header text.
messagelist	||delimited strings. Optional. Displayed as bullet points if provided.
showClose	0=DontShow 1=Show; sets the visibility of the panel's (x) button. Default: 1
closeEvent	1=serverHitAndClose or 0=JustClose. A single value for the (x) button. Default: 0
position	the string "page" or the ID of any element on the form. Default: page
sBefore	"before" or "after". The message panel is displayed before or after the element specified in the position parameter. If the element is "page" then the message is displayed at the top or at the bottom of the page depending on this value. Default: before
Style	"primary", "warning", "info", "danger", or "success"; style of the panel. Default: danger
msec	the animation parameter. Recommended values: 0-1000 Default: 0

Example:
rdShowMessage("XYZ", "Please correct the following entries:" , "A few errors have been detected!", "First Name||Last Name||Age||Gender||Team||Sports", 0, 1, "page", "before", "info", 300);

A few errors have been detected!
Please correct the following entries:
- First Name
- Last Name
- Age
- Gender
- Team
- Sports

Example usage:

rdShowMessage(messageEventId,"Please correct the following entries:" , "A few errors have been detected!", "First Name||Last Name||Age||Gender||Team||Sports",1,0, "page", "before", "info", 300);

There can be multiples of these on a page and these can be shown/hidden as needed using the messageEventId. It doesn't have to be connected to a hidden field, any ID will do as long as it doesn't cause a problem in the DATACOMP.

Example usage in a subroutine:
```
AJX.ARG = ''
AJX.ARG<1> = 'formMessages'                          ;* form element id
AJX.ARG<2> = 'Are you sure that the name is to be changed?'   ;* message text
AJX.ARG<3> = 'Client Name Change Request'            ;* header text
```

```
AJX.ARG<4> = 'Bullet Point 1||Bullet Point 2'        ;* optional message list
AJX.ARG<5> = 0                                        ;* show close
AJX.ARG<6> = 0                                        ;* close event
AJX.ARG<7> = 'page'                                   ;* position near name field
AJX.ARG<8> = 'after'                                  ;* before or after field
AJX.ARG<9> = 'success'                                ;* style
AJX.ARG<10> = 0                                       ;* milliseconds delay
CALL DBI.G.AJXCMD('SMSG',AJX.ARG)
```

The *Form Data Link* treats RD Designer "hidden" fields as a button.

The developer will get a *BUTTON* event in their linked program code if there is a close event associated with the displayed message.

message text may contain HTML e.g. text (for bold), <br> for new lines. VM and SVM characters are converted to <br>.

header can contain <b> and <u>.
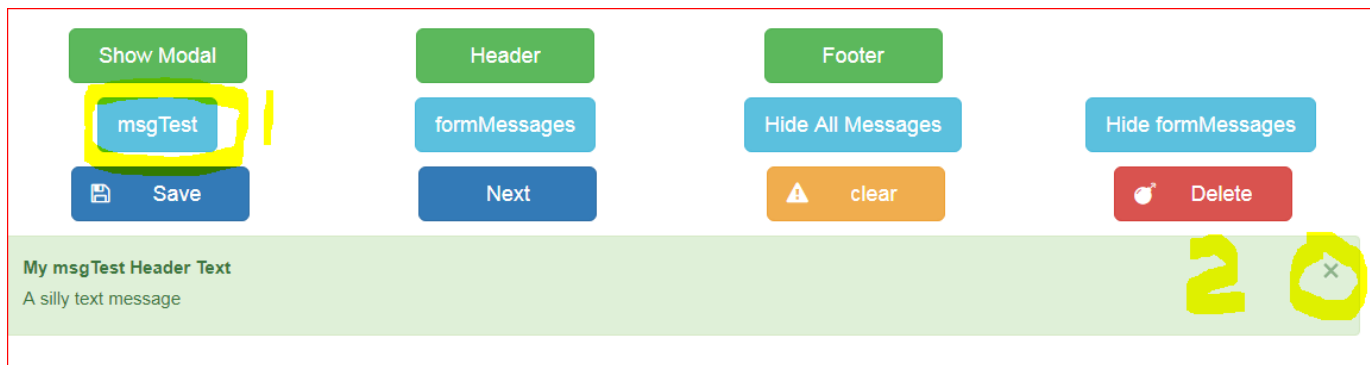
rdHideMessage(messageEventId, msecDelay)

Hide a standard message panel displayed using rdShowMessage.

messageEventId   ID of a hidden field created in RD Form Designer
msecDelay   Recommended values 0-1000 msec, default 0

Example usage in a subroutine. Consider the following code from the validation event. Note that the error message is displayed by the call to SHOW.MESSAGE using the *SMSG* function. The message must be cleared by using the *HMSG* function otherwise the message will remain even after the field validation rules have been passed.

```
IF DBVALUE # '' AND CMP.FLD # '' THEN
    IF NOT (DBVALUE = CMP.FLD) THEN
        AJX.FUNC = 'SMSG'
        THE.ID = 'email'
        THE.MSG = 'Email address mismatch'
        THE.MSG.HEADER = 'Email Address'
        THE.MSG.FIELD = 'email'
        THE.MSG.LIST = ''
        THE.MSG.SPOT = 'after'
        GOSUB SHOW.MESSAGE
    END ELSE
        AJX.FUNC = "HMSG"
        AJX.ARG = 'email'
        CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
    END
```

```
                END

        *
        SHOW.MESSAGE:
        *
                AJX.ARG = ''
                AJX.ARG<1> = THE.ID
                AJX.ARG<2> = THE.MSG
                AJX.ARG<3> = THE.MSG.HEADER
                AJX.ARG<4> = THE.MSG.LIST
                AJX.ARG<5> = 0
                AJX.ARG<6> = 0
                AJX.ARG<7> = THE.MSG.FIELD
                AJX.ARG<8> = THE.MSG.SPOT
                AJX.ARG<9> = 'success'
                AJX.ARG<10> = 0
                CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
                RETURN
```

rdHideMessageAll(msecDelay)

Hides all message panels on the page.
msecDelay   Recommended values 0-1000 msec, default 0

rdScrollToID(id, msec)

Scrolls the page to bring the row of the element into view. Only a single Id can be specified.

Id   Form element id
msec   Recommended values 0-1000 msec, default 300

rdShowAllRows(sectionTarget, msec)

Displays all rows of a major section (header, footer or body) if hidden.

sectionTarget   "header", "footer" or "body"
msec   Recommended 0-1000 msec (default 0)

rdHideAllRows(sectionTarget, msec)

Hides all rows a major section (Header, footer or body) if visible.

sectionTarget   "header", "footer" or "body"
msec   Recommended 0-1000 msec (default 0)

rdShowSection(sectionTarget, msec)

Displays a major section if hidden.

sectionTarget   "header", "footer" or "body"
msec   Recommended 0 msec

## rdHideSection(sectionTarget, msec)

Hides a major section if visible.

sectionTarget	"header", "footer" or "body"
msec	Recommended 0 msec

rdSetElementValue(id, value)

Directly set the value of any element, usually custom / injected elements.
The element can be of type input-text, output-text, select, checkbox, radio or textarea.

Id	ID of the element
Value	This can be can integer, string, boolean or numeric.

The following example shows how to change the name of a button, in this case it is a Group Button on an input field.
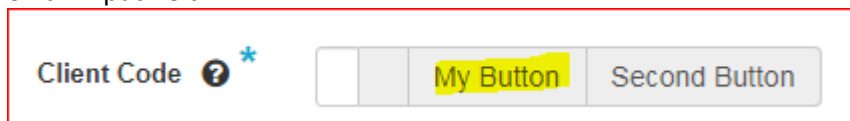

```
AJX.ARG = 'clientBut1'
AJX.ARG<2> = 'My Button'
CALL DBI.G.AJXCMD('SEV',AJX.ARG)
```

rdNavigate(id)

This function navigates to the specified page.
Example usage:

        rdNavigate('dbweb-helloworld')

This will open a new page called *helloworld* on folder *dbweb*.

This function can also be called using the subroutine:

```
AJX.FUNC = 'NAV'
AJX.ARG = 'dbweb-helloworld'
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

The *rdNavigate* method avoids a server hit since it navigates directly to the required page.

To implement this in a RD form use the HTML element with a click event:

rdNavigate; Usage in RD Form Designer
- Create a HTML "Lorem ipsum" form element (CTRL+K)
- Click CTRL+Q to edit.
- Go into Source view and replace:
        <p>Lorem ipsum</p>
- with
        <p>Button Text</p>
- Tick the event checkbox

In this example the hyperlink will display as Button Text

rdDeleteTableRow(id)

This function deletes a row from a table.
Example usage:

      rdDeleteTableRow("tr-gridTest-3");

This will delete the row with ID of *tr-gridTest-3*. Implemented as follows:

      DBAJAXCMD<-1>='rdDeleteTableRow("tr-grid-test-3"); '

Jqsv(id, value)

This function uses jQuery to set the value of an element. It can also be called using the subroutine.

To set the value of an element:

```
AJX.FUNC = 'JQSV'
AJX.ARG = id
AJX.ARG<2> = 'element'
AJX.ARG<4> = 'value'
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

To clear a value send a null in AJX.ARG<4>.
This function is translated as follows in the subroutine DBI.G.AJXCMD:

```
THIS.CMD = '$("#':ID:'").val("':VALU:'")'
```

Jqst(id, value)
Sets or returns the text content of selected elements.
Translated as:

```
THIS.CMD = '$("#':ID:'").text("':VALU:'")'
```

Jqshtml(id, value)
Sets or returns the content of selected elements (including HTML markup).
Translated as:

```
THIS.CMD = '$("#':ID:'").html(':"'":VALU:"')"
```

Jqssh(id, value)
Show HTML elements with the show() method.
Translated as:

```
THIS.CMD = '$("#':ID:'").show()'
```

Jqhd(id, value)
Hide HTML elements with the hide() method.
Translated as:

```
THIS.CMD = '$("#':ID:'").hide()'
```

Jqsa(id, value)
Translated as:

```
THIS.CMD = '$("#':ID:'").attr("':TARG:'","':VALU:'")'
```

Jqra(id, value)
Translated as:

```
THIS.CMD = '$("#':ID:'").removeAttr("':TARG:'","':VALU:'")'
```

Jqss(id, value)
Translated, where *value* is assigned to the variable *ATT* and can be subvalued, as:

```
THIS.CMD = '$("#':ID:'").css({'
FOR YY=1 TO ATTMAX
   ATTSV = ATT<1,1,YY>
   IF ATTSV = '' THEN ATTSV = ATT<1,1,1>
   VALUSV = VALU<1,1,YY>
   IF VALUSV = '' THEN VALUSV = VALU<1,1,1>
   THIS.CMD := '"':ATTSV:'":"':VALUSV:'"'
   IF YY#ATTMAX THEN THIS.CMD := ","
NEXT YY
THIS.CMD := '})'
```

addCode

```
AJX.FUNC = 'addCode'
AJX.DATAIN = 'ewayScript2'                  ;* Unique script ID to prevent reloading
AJX.DATAIN<2> = "var ewayKey='":EWAY.REC<DBIPM.EWAY.EKEY>:"'"
                                            ;* javascript variable holding the encryption key
AJX.DATAIN<3> = ''
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.DATAIN)
```

addScript

```
AJX.FUNC = 'addScript'
AJX.DATAIN = 'ewayScript1'                  ;* Unique script ID to prevent reloading
AJX.DATAIN<2> = EWAY.REC<DBIPM.EWAY.SCRIPT> ;* javascript file path
AJX.DATAIN<3> = ''                          ;* optional function to run once script is loaded
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.DATAIN)
```

addCSS

```
JMAX=DCOUNT(RESULTS<1>,VM)
FOR J=1 TO JMAX
   LOCATE 'href' IN RESULTS<2,J>,1 SETTING VPOS ELSE CONTINUE
   AJX.DATAIN = DBIACCOUNT:"_link_":J
   AJX.DATAIN<2> = RESULTS<3,J,VPOS>
   CALL DBI.G.AJXCMD('addCSS',AJX.DATAIN)
NEXT J
```

Another example:

```
AJX.DATAIN = ''
AJX.DATAIN<1>  = 'dbspScript1'
AJX.DATAIN<2>  = 'js/dbslidePanel-min.js'
CALL DBI.G.AJXCMD('AS',AJX.DATAIN)
```

setSlider
Refer to the documentation here: *https://www.designbais.com/documents/ionrangeslider/options.html*
Refer to the DesignBais Demo Form DBDEMO_SLIDER which has a link to the above documentation.

Can be either a single or double slider.

```
IF DBL THEN
    AJX.DATAIN = THIS.FIELD.XML
    AJX.DATAIN<10,-1> = VAL1
    AJX.DATAIN<11,-1> = VAL2
    CALL DBI.G.AJXCMD("SL",AJX.DATAIN)
END ELSE
    AJX.DATAIN = THIS.FIELD.XML
    AJX.DATAIN<10,-1> = VAL1
    CALL DBI.G.AJXCMD("SL",AJX.DATAIN)
END
```

slidePanel

Refer to the documentation here: *http://192.168.199.194/dbnet/js/docs/dbslidePanelAPI.html*

Refer to the DesignBais Demo Form DBDEMO_SLIDEPANEL which has a link to the above documentation.

```
*Set up the attributes for a slide panel element on a DesignBais form.

AJX.DATAIN = 'DB.WORK4.WK';* name of the field that contains the content for the slide panel
JMAX = DCOUNT(DBWORK<DB.TEMP.PANEL.WK>,VM)
FOR J = 1 TO JMAX
    IF DBWORK<DB.TEMP.PANEL.WK,J> # '' AND DBWORK<DB.TEMP.PANEL.VALUE.WK,J> # '' THEN
        AJX.DATAIN<3,1,-1> = DBWORK<DB.TEMP.PANEL.WK,J>
        AJX.DATAIN<4,1,-1> = DBWORK<DB.TEMP.PANEL.VALUE.WK,J>
    END
NEXT J
CALL DBI.G.AJXCMD('SP',AJX.DATAIN)
```

dbCarousel

Refer to the DesignBais Demo Form DBDEMO_CAROUSEL in the DBINET.DEMO Account.

Refer to the section in this manual titled *Implementing a Carousel*.

addEvent

Add an event.

```
AJX.FUNC = "addEvent"
AJX.ARG    = "DBIPM.U.ID.WK"
AJX.ARG<2> = "onmousedown"
AJX.ARG<3> = 'this.size=this.length;'
*
AJX.ARG<1,-1> = "DBIPM.U.ID.WK"
AJX.ARG<2,-1> = "onblur"
AJX.ARG<3,-1> = 'this.size=0;'
CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

customAttribute

Add custom attribute to a form element at run time.

removeCustomAttribute

Remove a custom attribute from a form element at run time.

dbDayPicker

day (date) picker (RD forms only)

dbDayPickerSetHTML

day (date) picker (RD forms only)

showSpinner

Show spinner text (can be used in non-RD forms)
Refer to the DesignBais Demo Form DBDEMO_SPINNER.

hideSpinner

Hide spinner text (can be used in non-RD forms)

authenticated
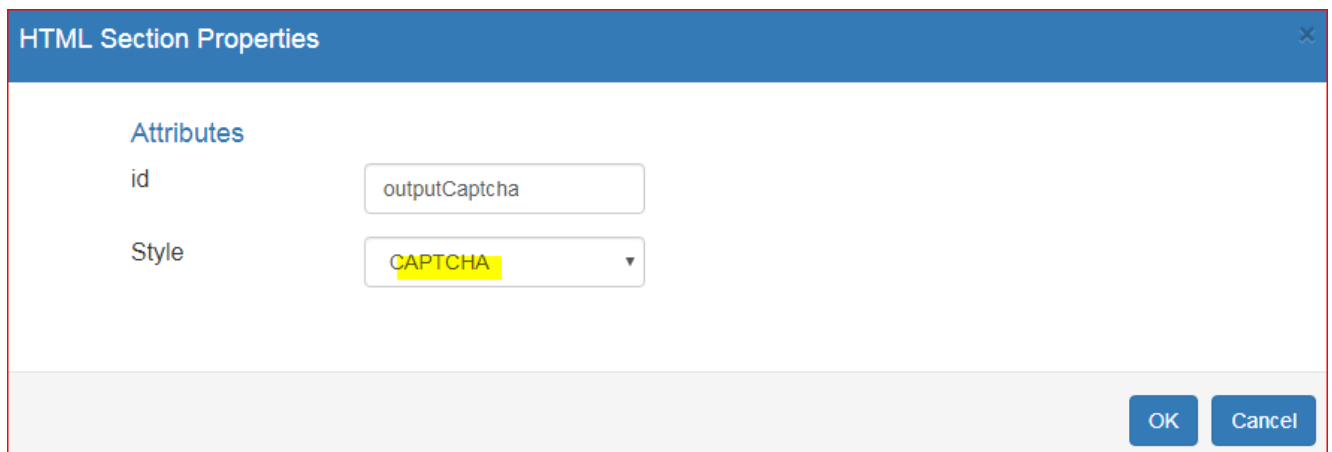
Set a user as authenticated (allow access to uploads)
Refer to the DesignBais Reference Manual.

CAPTCHA

This is a style attribute of an HTML form element. Use CTRL+K to insert the HTML Section. Select the *CAPTCHA* option in the *Style* dropdown.
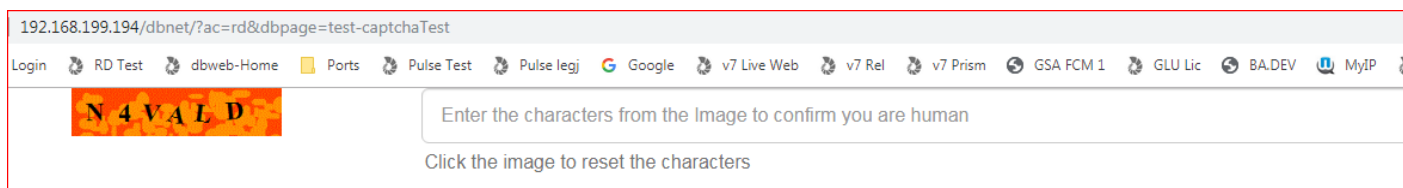
This will add a captcha image to the form and send the *captchaKey* to the data component with every event. The developer must add an *INPUT* field to the form. By comparing the value of the *INPUT* field to the *captchaKey* the developer can determine if the user has entered a matching string.

1) Add HTML field (Ctrl-K) and select CAPTCHA in the Style dropdown.

2) The *INPUT* field must have *Event* checked. The *captchaKey* value of the displayed captcha image is returned to the database via the xml string from the browser. DesignBais retains this value. In the Validation event of the input field the value of the captcha image entered by the user is sent to the database in DBVALUE as for any normal input field. In your basic subroutine set DBCAPTCHA = DBVALUE to allow the DesignBais engine to check the validity of the entered captcha field.

3) DesignBais then triggers another call to the field validation event in which the common variable PROCESS.PARAMETER = "GETCAPTCHA" and DBCAPTCHA.STATUS<1,4> has been set (1=failed, 0=matched).

This is demonstrated in the sample RD form test-captchaTest which is linked to DBCLIENT*RDCAPTCHA.

The validation event

```
119   VALIDATION:
120        *
121       BEGIN CASE
122          CASE SCREEN.NO = "RDCAPTCHA" AND EVENTSOURCE = "DBC.WORK1.WK"
123             IF PROCESS.PARAMETER = 'GETCAPTCHA' THEN
124                IF DBCAPTCHA.STATUS<1,4> = 1 THEN
125                   DBVALUE = ''
126                   DBDS<-1>='Your entry does not match'
127                END ELSE
128                   DBDS<-1>='All good'
129                END
130                RETURN
131             END
132             DBCAPTCHA = DBVALUE
```

Loading Scripts for DesignBais Responsive Design

Scripts can be loaded in several places.

On the Web Server for each web site
1. In the custom.js folder
2. In admin\htmlMetaTags.txt (javascript is normally in custom.js but can be loaded here as well)

On the Database Server – there should be maintenance forms for both:
1. Global Parameters - DBIGLOBAL SCRIPTS – for all accounts using the same login account. As you know, Bensons have 2 environments with separate login accounts.
2. System Parameters - DBIPARMS SCRIPTS – for individual accounts.

You don't need to load the same script more than once to make it available in a page.

Originally there was just the custom.js file on the web server. To allow the developer to control everything from the database component, as often access to the web server is restricted, the . It does mean that database security is important as well ... but that's a given.

You can load raw javascript into the SCRIPT items but the PEN test people prefer file links. Consequently, we have recently moved all of our standard javascript into our ajxw3c.js file and use arguments in function calls to tailor script behaviour to suit the database configuration.

When you start a session DesignBais loads the scripts = if you change a script then any current sessions need to be refreshed to load the updated script before it will be utilised. You will be used to this with stylesheets.

If you inspect a DesignBais page in your browser you will see in the <head> section the script, meta and stylesheets that get loaded.

Responsive Design pages work a bit differently.

The HTML is published to the database rather than constructed from DBIFORMS. The DBIFORMHTML file contains links to the header, footer, css and meta data files to be included when the page is run. These files live on the web server in res/bensons folder.

Consequently, the DBIPARMS SCRIPTS & DBIGLOBAL SCRIPTS records do not come into play for RD pages.

To make these visible to the database developer I created the Site Meta Data maintenance:

This form will update the site folder on the web server when submitted (similar to updating the css when a style is changed) and store the changed record in the database.

You have two options:

1. Check that the record shown in the above maintenance matches the res/bensons folder items before changing and submitting – I'm not sure if you touched the web items during your recent development project.

2. Delete the RDMETA*bensons record and always update the res/bensons record directly on the web server.

Jon

From: David Humphreys <dave@bealdavis.co.uk>
Sent: Tuesday, April 16, 2024 6:03 AM
To: Jon Legg <Jon@bais.com.au>

Cc: Bulent Can <Bulent@bais.com.au>
Subject: RE: DYNATRACE SCRIPTS

Hi Jon

Thanks for the below. We have added the new script to our Affinity software .

For self service I can see that the current Dynatrace script is in DBIPARMS RDMETA*bensons.

Can the new script be placed in the DBIPARMS SCRIPTS item (same as Affinity) and the RDMETA*bensons item deleted . If not how do I get the script into RDMETA*bensons.

Regards

Dave

From: Jon Legg <Jon@bais.com.au>
Sent: Tuesday, April 2, 2024 12:36 AM
To: David Humphreys <dave@bealdavis.co.uk>
Cc: Bulent Can <Bulent@bais.com.au>
Subject: RE: DYNATRACE SCRIPTS

Dave,

The DBAdministrator User Group probably needs to be set up and/or the users added for the option to show in the menus. In the latest version we have removed this restriction. See below.

On http://10.167.212.26/?ac=bensons

Global Parameters -> Meta Data – nothing entered in "Script References"

System Parameters -> Meta Data - nothing entered in "Script References"

On http://10.167.212.55/dbnet/

I created the DBAdministrator User Group and added it to the users designbais & dbnetuser .

This made the Global option appear after a fresh login.

Global Parameters -> Meta Data – nothing entered in "Script References"

System Parameters -> Meta Data - nothing entered in "Script References"

Both test systems – you may need to create the DBAdministrator User Group on live and add the users into the group to get the maintenance forms to show on the menu.

The maintenance items populate DBIGLOBAL SCRIPTS and DBIPARMS SCRIPTS = yes you could edit directly.
You will need to refresh your session to load the script.

I normally right-click and select Inspect – F12 messes with our remote connection tool.

We only have one page and any scripts loaded will be available for every DesignBais form. Forms are loaded into separate <div> elements within the page as needed.

Scroll to the top and expand the <head> section.

```
<html>
  ▶ <head> ⋯ </head>
  ▼ <body style="margin:0" x-ms-format-detection="none">
```

Scrolling through you will see the files loaded:

```
<link href="jq/jqalerts/jquery.alerts.css?v=437" rel="stylesheet" type="text/css">
<link href="designBAISStyle.css?v=437" rel="stylesheet" type="text/css">
<link href="designBAIS.css?v=437" rel="stylesheet" type="text/css">
<link href="css/compatible_w3c.css?v=437" rel="stylesheet" type="text/css">
<link href="css/awesome.css?v=437" rel="stylesheet" type="text/css">
<link rel="stylesheet" href="jq/themes/redmond/jquery-ui.min.css?v=437" type="text/css">
<link rel="shortcut icon" href="favicon/favicon.ico?v=437" type="text/css">
<link href="js/css/dbwidgets.css" rel="stylesheet">
<script src="w3cmenuScript.js?v=437" type="text/javascript"></script>
<script src="jq/jquery.js?v=437" type="text/javascript"></script>
<script src="jq/ui/jquery-ui.min.js?v=437" type="text/javascript"></script>
<script src="charts/highcharts.js?v=437" type="text/javascript"></script>
<script src="charts/highcharts-3d.js?v=437" type="text/javascript"></script>
<script src="charts/highcharts-more.js?v=437" type="text/javascript"></script>
<script src="charts/modules/annotations.js?v=437" type="text/javascript"></script>
<script src="charts/modules/no-data-to-display.js?v=437" type="text/javascript"></script>
<script src="charts/modules/exporting.js?v=437" type="text/javascript"></script>
<script src="charts/themes/sunset.js?v=437" type="text/javascript"></script>
<script src="ajxw3c.js?v=437" type="text/javascript"></script>
<script src="jq/jqalerts/jquery.alerts.js?v=437" type="text/javascript"></script>
<script src="js/dbwidgets.js"></script>
<script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?key=AIzaSyD7c!
ces" async defer></script>
<script src="https://protect.sandbox.paytrace.com/js/protect.min.js"></script>
<script type="text/javascript" src="js/dbmisc.js"></script>
<script type="text/javascript" src="js/rick/rick.js"></script>
<script src="custom.js?v=437" type="text/javascript"></script>
```

You can then right-click say the custom.js file and open in a new tab to see what's inside. In my case our release version:

```
//JQUERY is available

function beforeHitCustom(dbEventSource, dbEventType, dbClass, dbFormName) {

    //Your code
    //return true to continue with the server hit
    //return false to cancel the server hit

    return true;
}




function afterHitCustom(dbEventSource, dbEventType, dbClass, dbFormName) {
    //Your code
    //Always return true
    //
    return true;
}
```

Calling RD from DesignBais and Returning to DesignBais

Calling a Responsive Design form from a DesignBais form.

```
BUTTON:
   BEGIN CASE

      CASE EVENTSOURCE = "B.RDTEST" AND SCREEN.NO = "JL10"
         DBCALLURL = "http://192.168.199.194/dbnet/?dbpage=test-clientMnt;newWindow"

      CASE EVENTSOURCE = "B.TEST"
         DBDS<-1> = 'HERE'

   END CASE
```

Data is carried forward to the new page using the old session data via *IdWindowOld* passed in by the web component.

The old session records are then deleted so that DBISESSIONS does not grow unnecessarily large.

SESSION.REC attribute <72> holds DBRDMODE which determines that this is a Responsive Design session.

- DBCALLURL = http://192.168.288.288/dbnet/?dbpage=test-clientMnt;newWindow opens a new tab.

- DBCALLURL = "dbpage*DBCLIENT_MAINT" replaces the current tab.

For the technically-minded the process is as follows:
- Hit 1 establishes the connection.
- Hit 2 the data component sends the basic form and sets RDMODE if the form is RD.
- Hit 3 the web component sends IDWindowOld if it can identify an earlier session. The data component then either does an Auto Logon or for an earlier session carries the session variables forward from the previous RD page for the new RD.

Once you are in RD then use PROCESS.STACK in the normal way to go to another RD form:

- PROCESS.STACK="DBCLIENT_MAINT"

In order to move back to DesignBais from a Responsive Design form it is necessary to use a full URL in order to invoke a fresh session start up.

We should always go to the dbpage. The developer should decide if a user can be on that page at the beginning of a session.   But in most cases the developer will need to redirect to the start page.
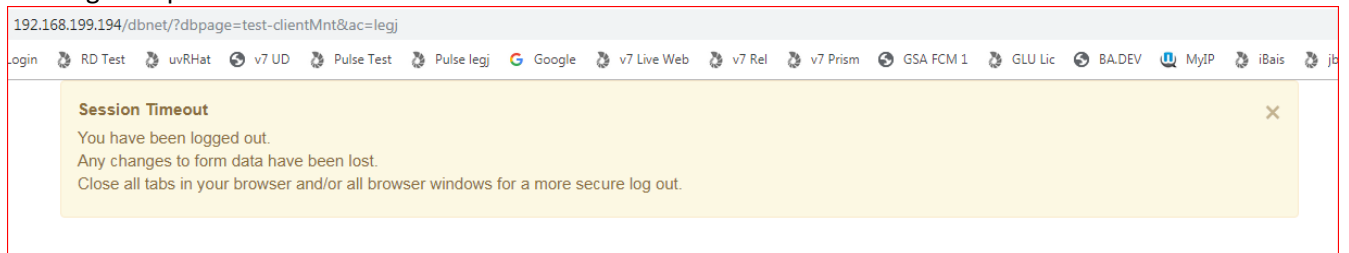
Developers should check their session data and decide if they allow access to that page.
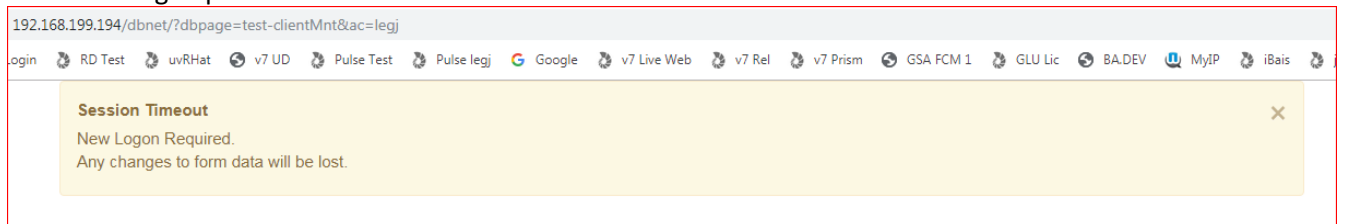
Session Timeout
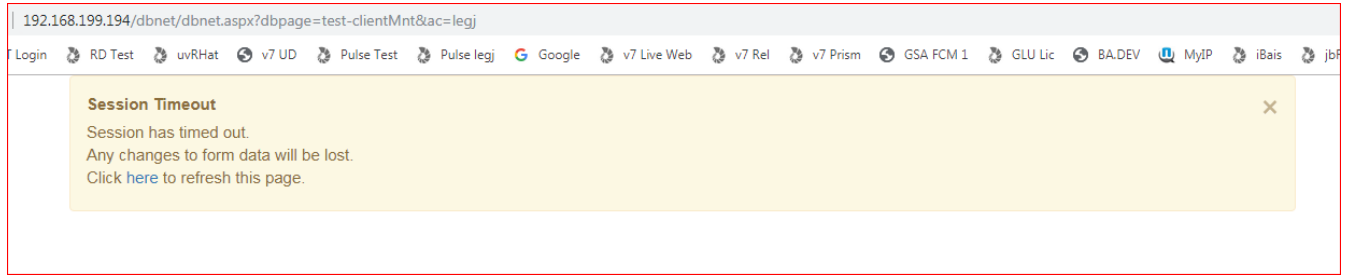
At this point the user has been logged out in order to kill the session data.

The Logout Option:

The Error Page option:

When there is a time out, we should clear the session data and then load that page with sesssion=null. At that point the developer decides what to do.
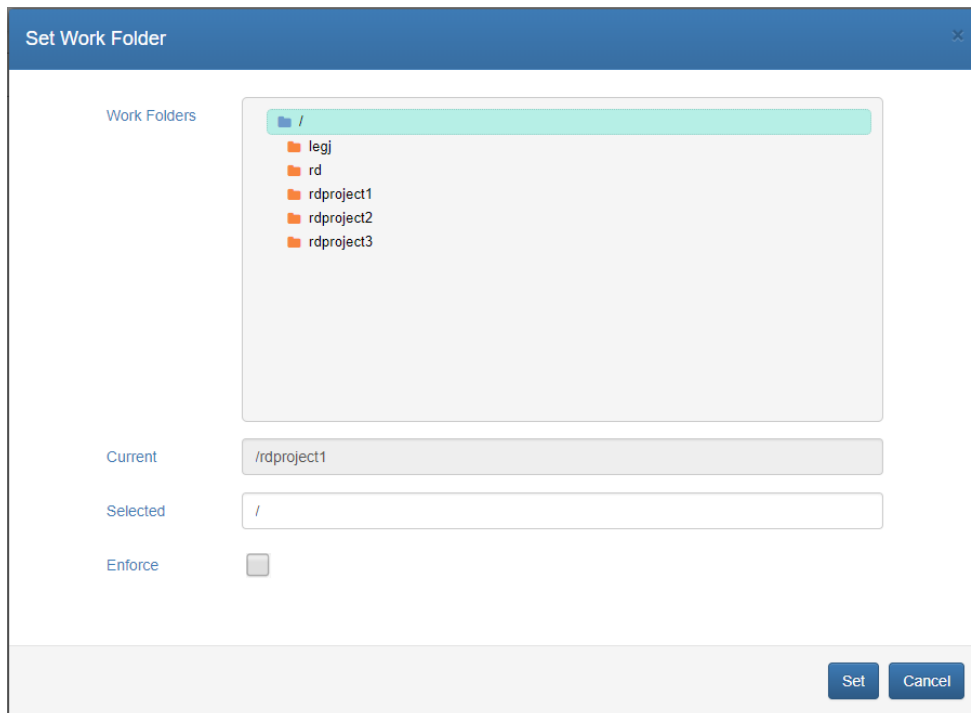
Responsive Design Top Menu Options

The top menu options are *File*, *After* and *GridOff*.

File Options: Work Folder [Alt W]

Let's look at the *Work Folder* option first as we need to exercise this option before we can produce a web page.

Use the *Set Work Folder* option to create a folder or to select an existing folder as your current work folder.

Type the name of the folder you wish to create in the *Selected* field. Click *Set*. This will create a new folder and also set it as your current work folder.

To change your current work folder simply click the orange folder icon adjacent to the required folder, then click *Set*.

Folder names for *Site Paths* must not contain space or special characters.
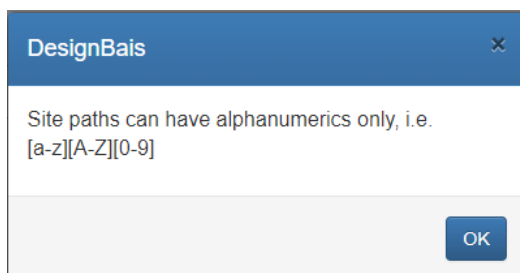
In order to move a page to a different work folder, open the page, then click Alt-W and select the new work folder, then click Alt-A and enter the required page name in the File Name field.

The following message is displayed if your current work folder has been deleted, or if you attempt to open a page in a work folder that no longer exists.

Path error ✕

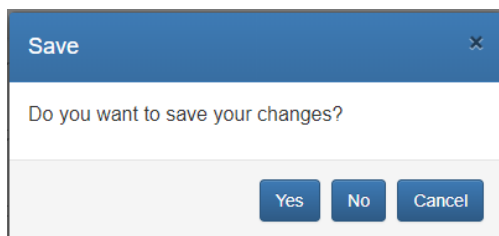Directory not found
Your work folder is now set to '/'
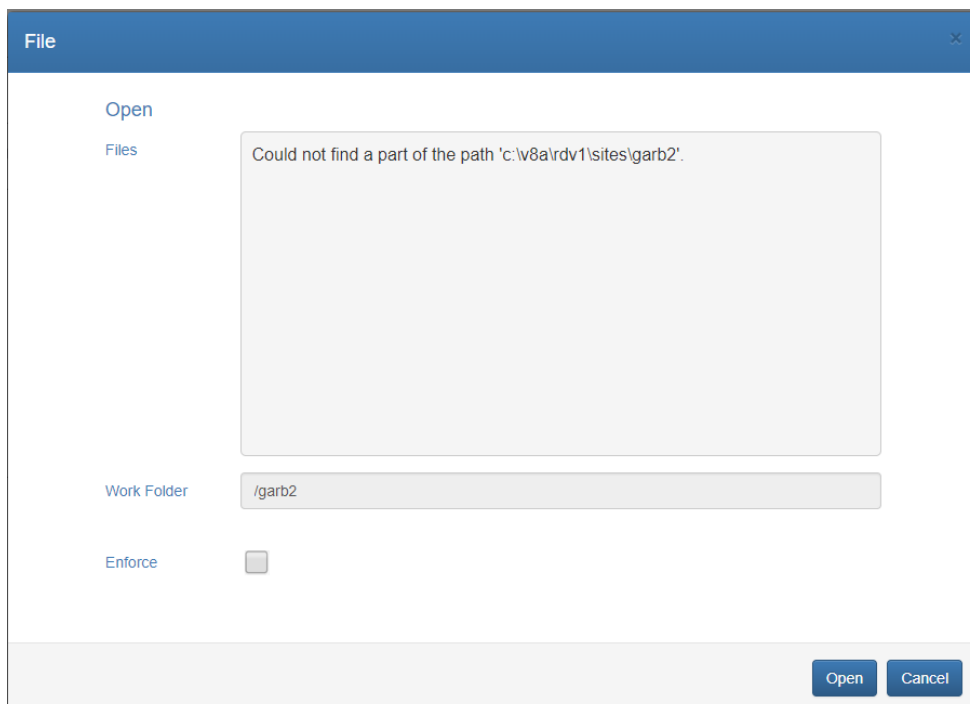
OK
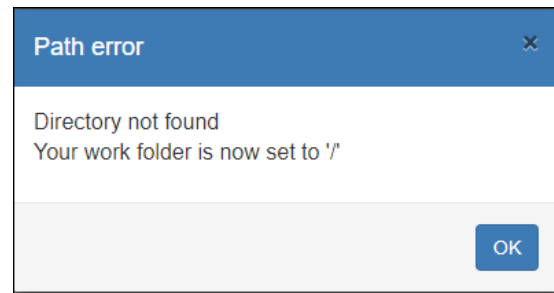
File Options: Open [Alt O]

Opens an existing page for review or change.

Select the required page from the selection window.

If a page is already open and has been modified then you will be prompted:

Save ✕

Do you want to save your changes?

Yes No Cancel

If a folder name is modified while a page from that folder is open then the following message displays:

File ✕

Open

Files

Could not find a part of the path 'c:\v8a\rdv1\sites\garb2'.

Work Folder   /garb2
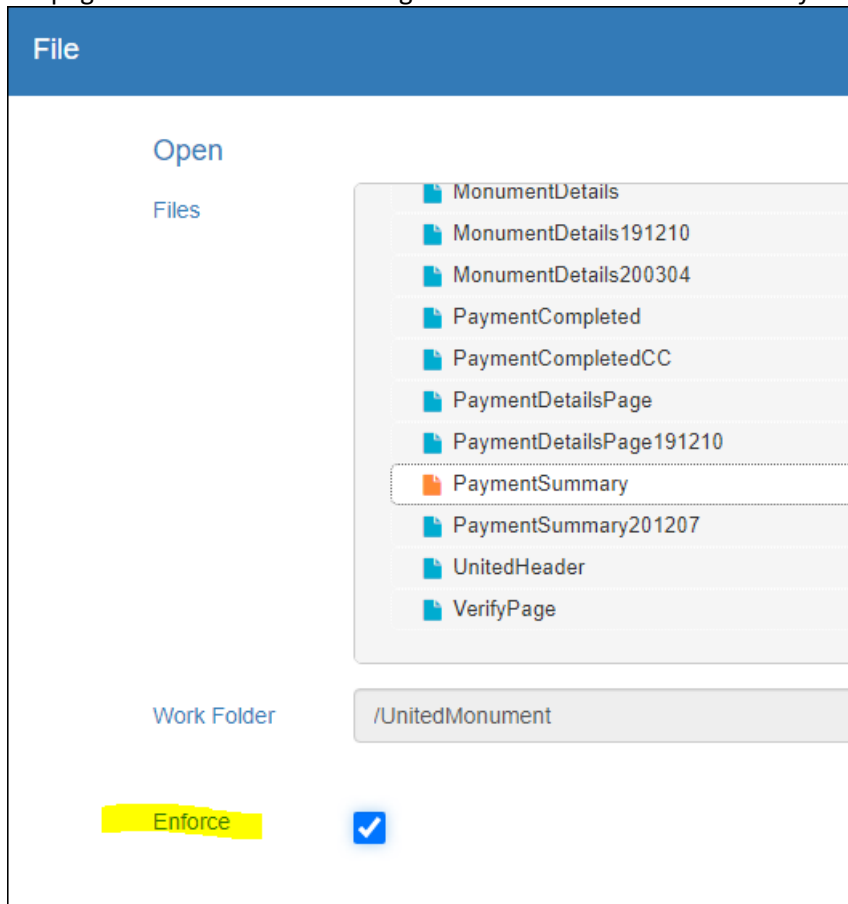
Enforce   ☐

Open Cancel

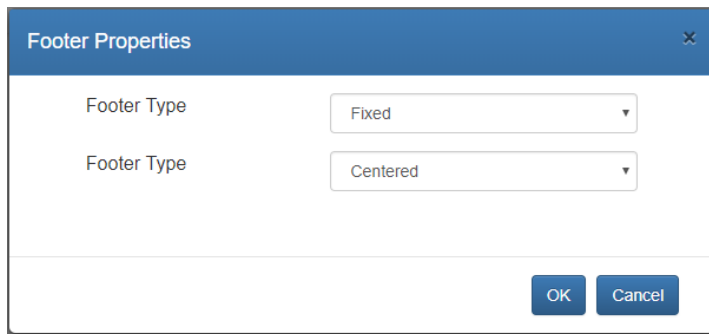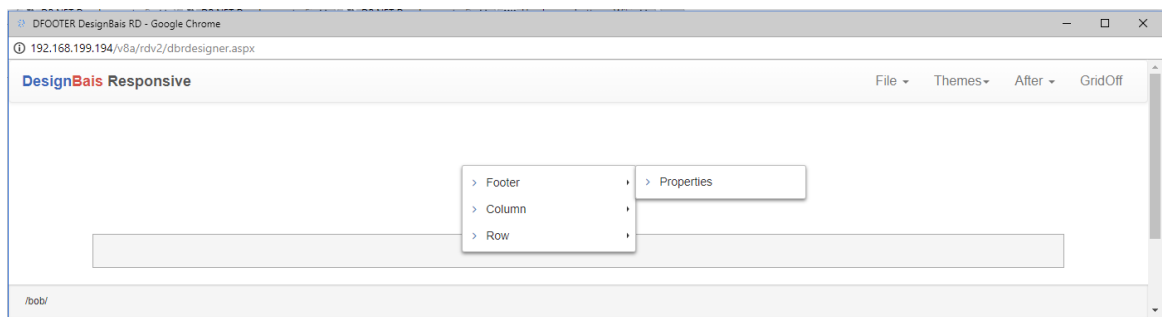Use the *Save As* option to save the current page to a new folder.

If the page is locked then the following message displays:
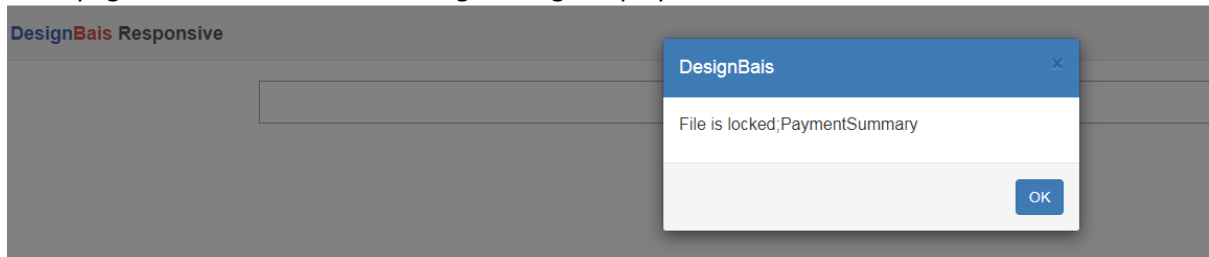
If a page remains locked following a browser crash then use the *Enforce* checkbox to open the page:

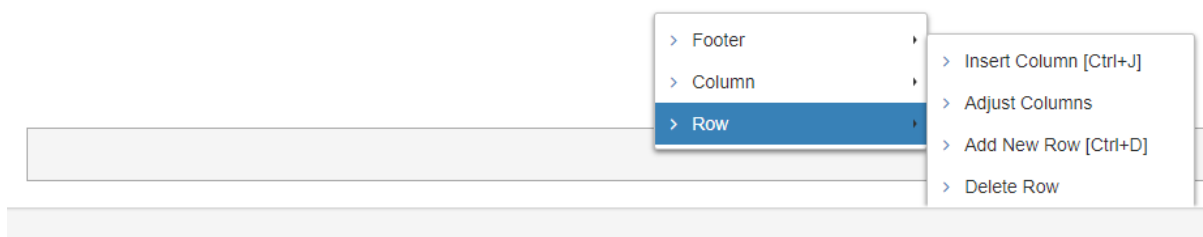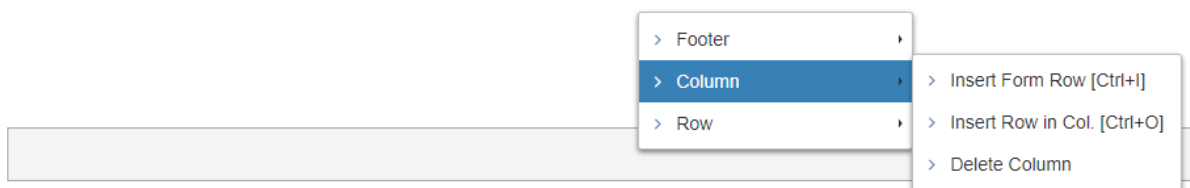## File Options: New Footer [Alt F]

Footer Type   Options are:

> *Scrolling*
> *Fixed*
> *Sticky*

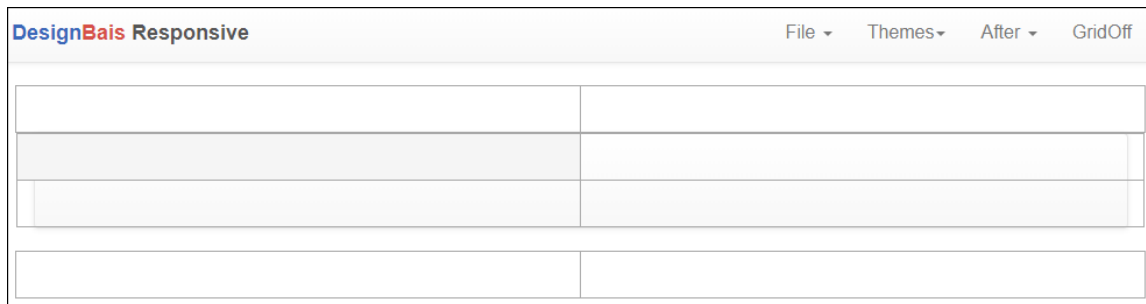*Footer Type*   Options are:

> *Full*
> *Centered*

The Footer Column and Row options shown below are documented in the *Responsive Design Form Creation* section of this manual.

File Options: New Header [Alt H]

Create a new form header.

The header is sandwiched between two grid rows. The default header rows are distinguished by the presence of the *NavBar* option when the row is right-clicked. The second header row is present to allow for a header design that requires no space, or buffer, between header elements, such as menu items, and some other form element such as a text input field. If it is not required it can be deleted.

Right click the selected column to display the header page options.
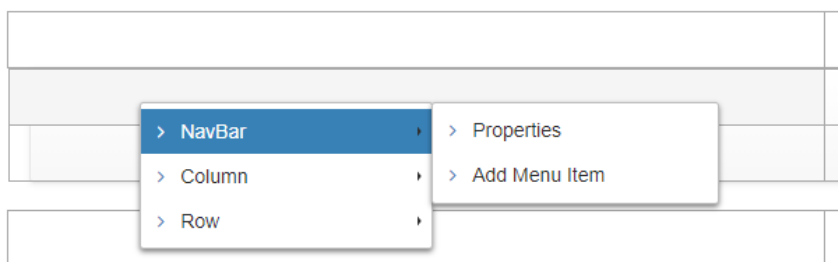
NavBar > Properties
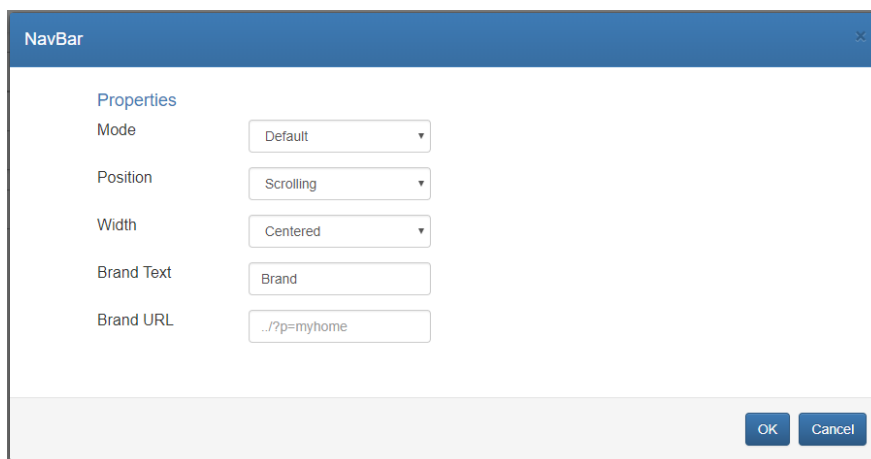
Mode   Select the Inverse option for a photo-negative effect otherwise leave as the Default option.
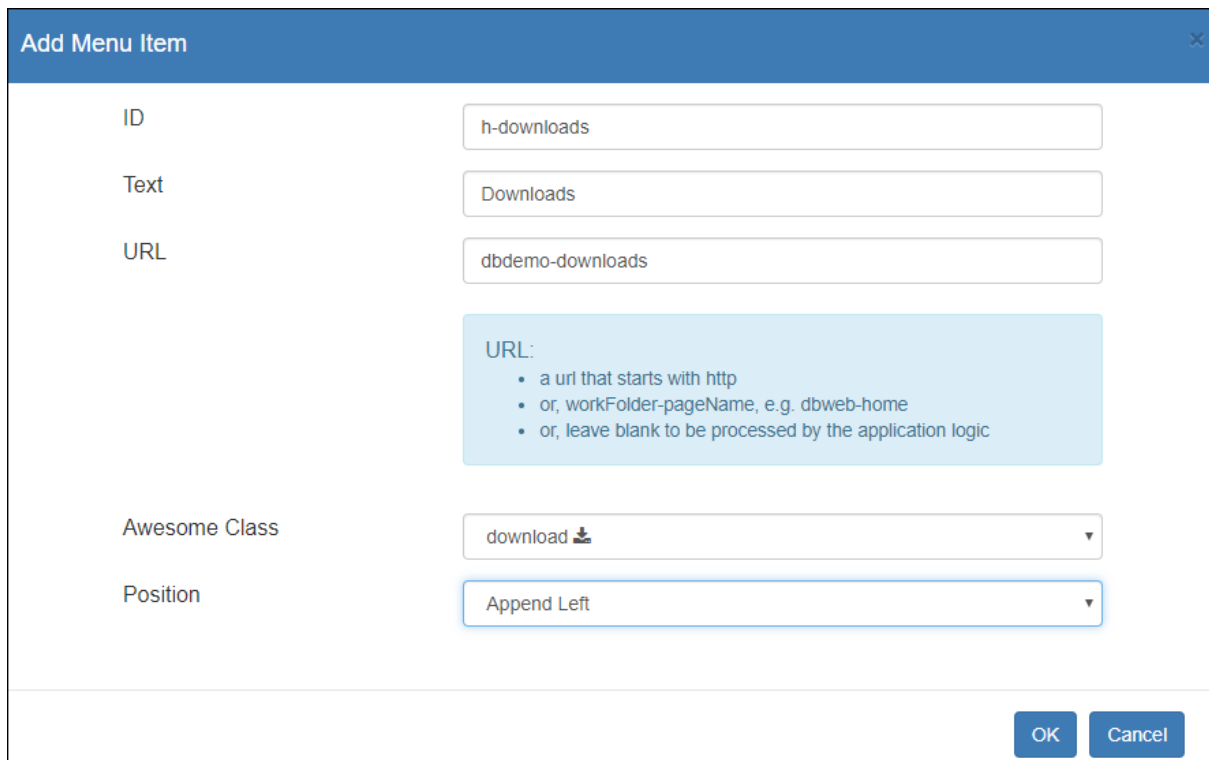
Position   There are two options Scrolling and Fixed. The Scrolling option allows the header to scroll off the top of the browser display. The Fixed option causes the remainder of the page to scroll behind the header, leaving the header always in view.

Width	Full expands the header to fill the width of the form while Centered centers the header on the form.
Brand Text	Text to appear in the header row as a link.
Brand URL	The URL to run when the *Brand Text* link is clicked.

NavBar > Add Menu Item

ID	The id to be assigned to the menu item. This is required to allow it to be referenced.
Text	The text to appear in the header bar to identify the function of the menu item.
URL	As shown this can be either a url starting with http, or a workFolder-pageName such as dbdemo-downloads. Leave blank if the click event will be handled by the application logic. Note that external URLs are opened in in the same window. If the external URL needs to be opened in a new window then leave the URL blank and use window.open(url) in the application logic in the data component basic subroutine. Example: add ";newTab" to the end of the URL.

```
IF DBW3CQSTRING # '' THEN QS = "&":DBW3CQSTRING ELSE QS = ""
DBCALLURL = WEBSERVER:"?dbpage=":DBWORK<DEM.RD.PAGE.WK>:QS:";
newTab;dbAccount=":DBIACCOUNT
```

This results in the following:

```
window.open('http://192.168.199.194/dbnet/dbnet.aspx?dbpage=dbdemo-
slider&ac=legj');
```

Note: the dbAccount option is only needed if the URL + ac code does not start in the correct account.
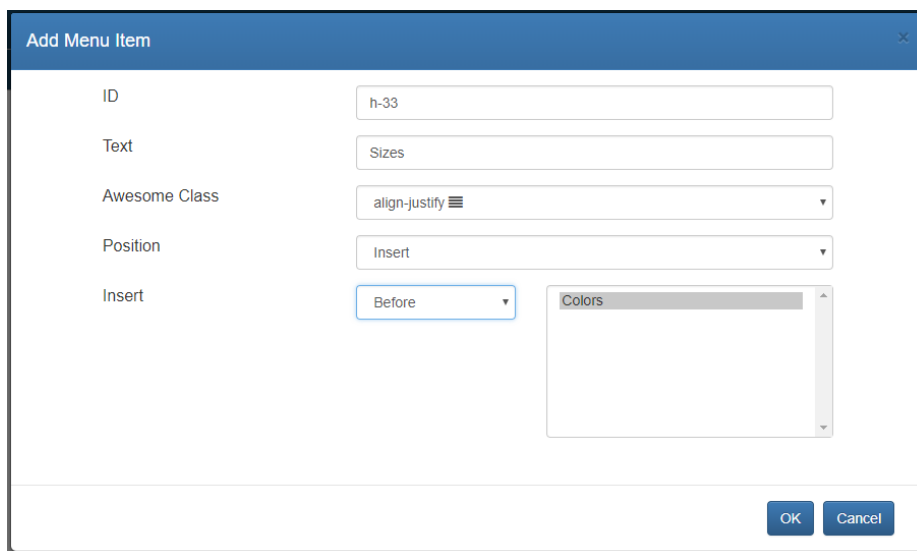
The old option ";newWindow" opens a new browser instance.

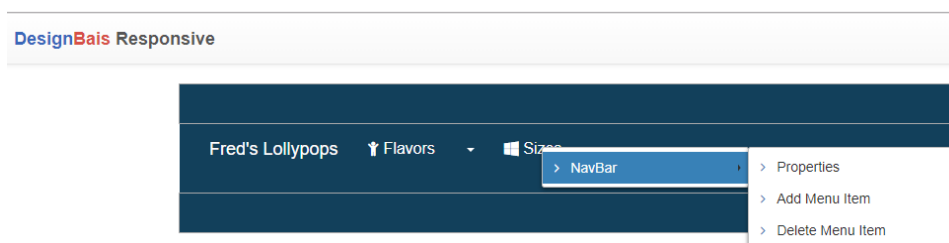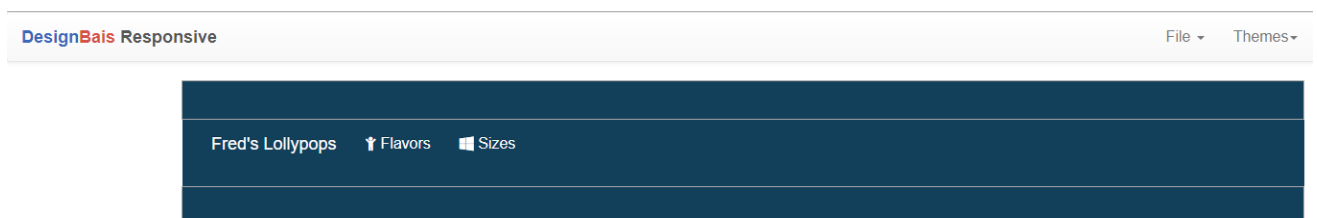*Awesome Class*　　　　Select an icon from the dropdown list.

Position　　　　　　The options are Append Left, Append Right or Insert. The left and right options refer to the position of the menu text within the column housing the menu item. Use the Insert option to place a menu item between existing menu items.

Example of calling a form from a menu by using your application code:

Your basic code can set DBCALLURL or use PROCESS.STACK = "Filename_Formname".
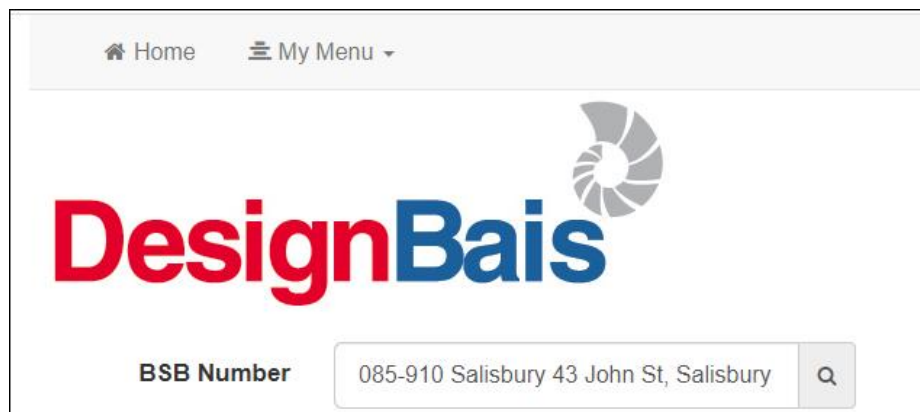
NavBar > Delete Menu Item

Deletes the focused menu item. There is no request for confirmation.

The header form navigation bar may display above the header form image even though in forms designer it appears to be below the image.

Designer view
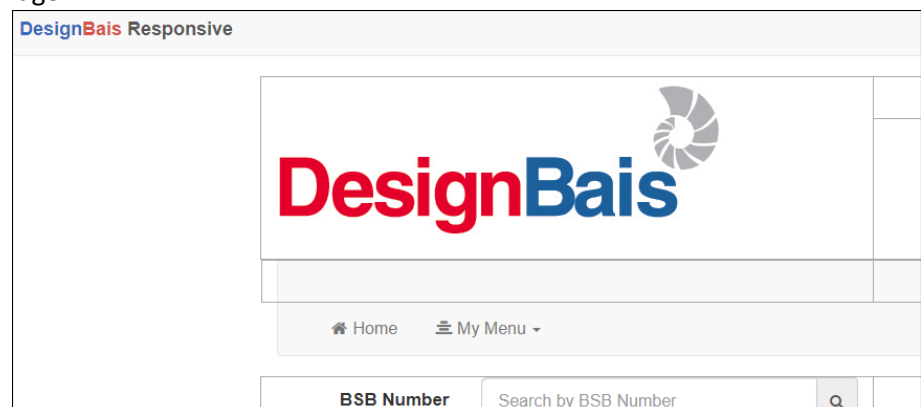
Run time view

To overcome this issue proceed as follows:
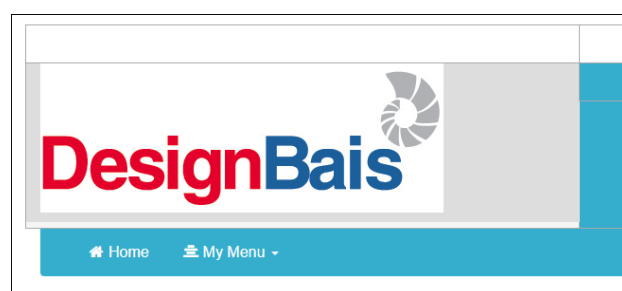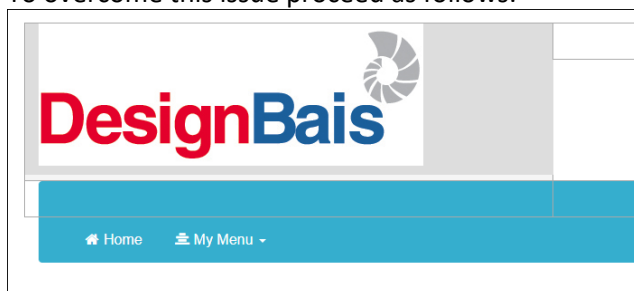
Press CTRL-X to cut the image from the top row of the header and paste it (CTRL+V) into the row beneath (the top row of the two green rows in the snip above).

You can also right click row 2 and insert more rows if you need to. The NavBar can have more than two rows in it. You may, depending on your theme, have a background color problem, e.g. you may want white background colour where you place the logo image, because by default, this row takes the background colour of the NavBar. In that case right click on the row and select:
> Row > Edit Code.

```
 1  <div class="row dbrow">
 2      <div class="col-md-6 dbcol dbcolhilite" style="">
 3          <div class="dbform form-horizontal dbrowhilite">
 4              <div class="dbformgroup form-group dbrowcolhilite">
 5                  <div dbtype="segment" class="html-text" id="h-logo" existingid="h-logo">
 6                      <p>
 7                          <img alt="" src="../images/dblogomaster.jpg" class="img-responsive">
 8                      </p>
 9                  </div>
10              </div>
11          </div>
12      </div>
13      <div class="col-md-6 dbcol" style=""></div>
14  </div>
15
```

In the code, for example, set style="background-colour:white" on the first <div> of the code.

Edit Code

```
i 1 ▾ <div class="dbrow row" style="background-color:white">
  2       <div class="dbcol col-md-6 dbcolhilite"></div>
  3       <div class="dbcol col-md-6"></div>
  4 </div>
  5  |
```

With a fixed NavBar the top row of the header is not needed. In fact you can right click and go Row -> Delete. We don't encourage editing the code but we don't have a better way in this case.

You need at least one row in addition to the nav bar. On a header page you have these by default:
    row
    navbar (can have multiple rows)
    row

You cannot delete both rows that exist outside the navbar. You must have at least one row on the page but a row won't display anything if left blank.

The detailed explanation for those interested:

The rows highlighted with yellow color is the NavBar.

Now, with that definition:

When the navbar is Fixed, it (those three rows) ignores everything and places itself right on top of the page. The first row sadly disappears behind it. Not in the designer though. This is one of the rare exceptions of the Designer (i.e. designer and test look different). It had to be like that because we don't want a row (ie. [1]) to disappear behind another row in design mode.
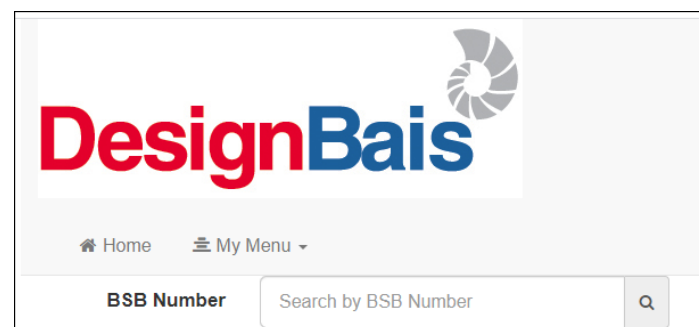
In this example the slide panel stays in front of the header when scrolling down the page. This can be solved by changing the z-index:

```
AJXIN = "dnavbar"
AJXIN<2> = "element"
AJXIN<3> = "z-index"
AJXIN<4> = "200"
CALL DBI.G.AJXCMD("SS",AJXIN)
```

File Options: New Page [Alt P]

Opens a new page.

If there is a page already open and there are changes the following prompt appears:
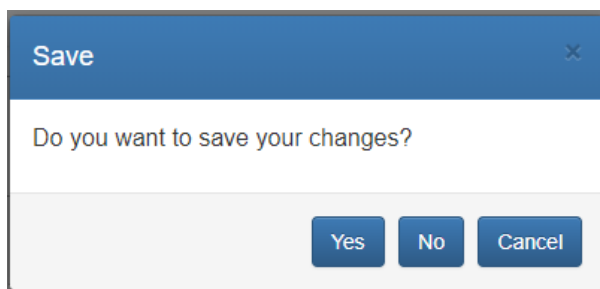
File Options: Save [Alt S]

Saves the current form. A message *Saved!* appears briefly to confirm that the form has been saved.

If a folder name is modified while a page from that folder is open then a message similar to the following displays:
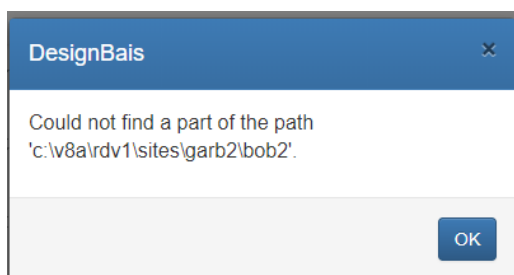
If a folder name is prefixed to the page name in the *File Name* field then the following error message will display:
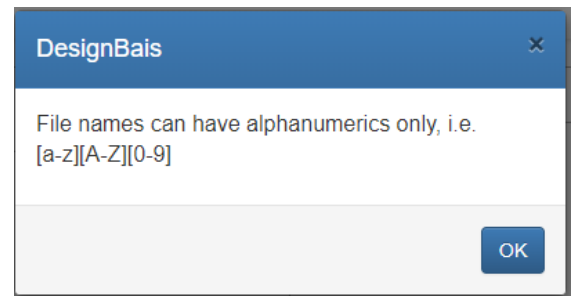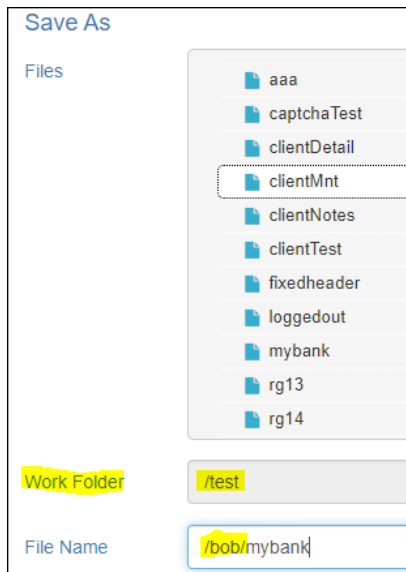
Use the *Save As* option to save the page to a different folder. See *Save As* below.

File Options: Save As [Alt A]

Allows a page to be saved under a new name.

Use this option to save an existing page to a different folder. With the page open in the designer select the *Work Folder* option from the *File* menu (or press *Alt-W*).

Select the new folder from the list of existing folders, or enter a new folder name in the *Selected* field. Click *Set* to set this as the current folder name.

In File -> Properties set theme, header and footer and publish.   It is recommended that the header and footer pages are exported first. Review all hyperlinks (including image links), delete and recreate links if necessary.

Then click either *Save* or *Save As* to save the page to the selected folder under either the current or a new page name.

File Options: Close [Alt C]

Close a page without saving changes. Note that there is no warning that changes will be lost.

File Options: Properties [Alt R]

Name	The name of the page.
Title	The title of the page.
Footer	The path to the footer, if any, to be included with this web page.*
Header	The path to the header, if any, to be included with this web page.*

* Always start with "/work folder/". It points to the "your_website/rdv2/sites" folder which is the root folder for all workspaces. If the header form "headform" is in work folder "bob" then enter "/bob/headform"

Theme	Set the theme for the web page.
Hidden Fields	Enter the ID for any hidden fields. The ID cannot commence with *f-* or *h-*. Hidden fields will be displayed in the Form Data Link option and can be linked to fields in the database.

Responsive Design Form Data Link   Recent   Copy Form

Page Name   test-rg13   test-rg13 Untitled
Page Description   Untitled

Filename   _____   *   --No File Selected--
Form Name   _____   *
                                                           CSS Filer
Full Description   _____   Header
                                                           Footer

Preserve Common   ☐   Sub Form   ☐
Button to Action when Enter is pressed   -No Items- ⌄

Process Before Display   _____   Par
Process after Display   _____   Par
Modal Close via X Process   _____

Default Key Value   _____
Form Read Group   ____   Form Read Variable -- Not Used -- ⌄
Form File Name   --No File Selected--   ⌄
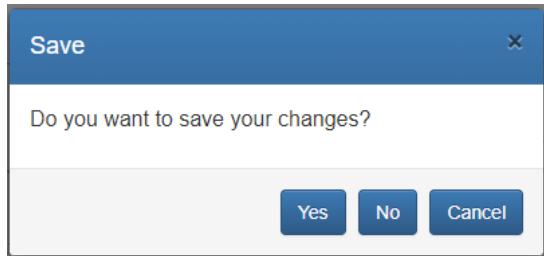Form Read Type   No Lock   ⌄

Search for ID   _____

ID	Type	Unlink	File	Field	Varia
d137098	text	✖			
t-rg	hidden	✖			
fred	hidden	✖			

File Options: Refresh [Alt E]

Refresh will refresh the page by reading the form from the work folder. If the current form has not been saved then any changes made since the form was read will be lost. The following message will display if changes are detected:

The following warning will display regardless of whether or not changes are saved.
It will also display if there are no changes.
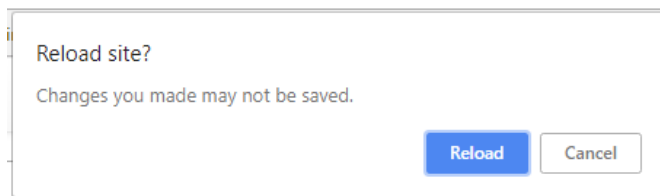
File Options: Delete Files [Alt D]

Use this option to delete pages from a work folder. The list of pages in the current folder is displayed. Select those that are to be deleted. Click the *Delete* button.

Warning! There is no confirmation step after clicking the *Delete* button. Be sure that the pages selected and highlighted in orange color are the pages that you want to delete.

If the *Delete* button is clicked without selecting a page name the following message will be displayed:
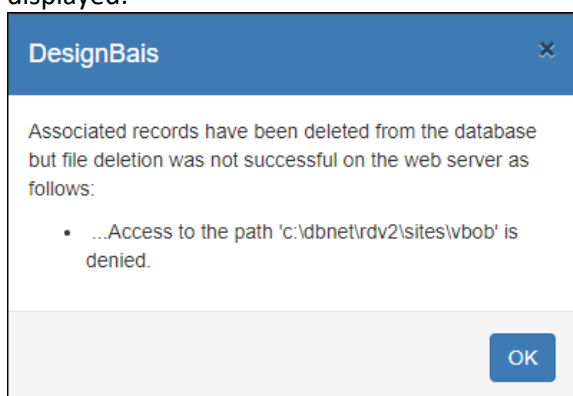
File Options: Publish [Alt L]

A form must be saved before it can be published.

Publish calls a web service to pass the XML record to the DesignBais Database Component where the developer can link the elements on the form to table fields and events on the database.

The Responsive Design *Page Name* holds the id of the published form within the database component. It is stored in the file DBIFORMHTML. The *Page Name* is formed from the path to the form within the *sites* folder.

With reference to the snip below the form called *clientMnt* when published will be assigned a *Page Name* of *test-clientMnt* reflecting the path within the *sites* folder.

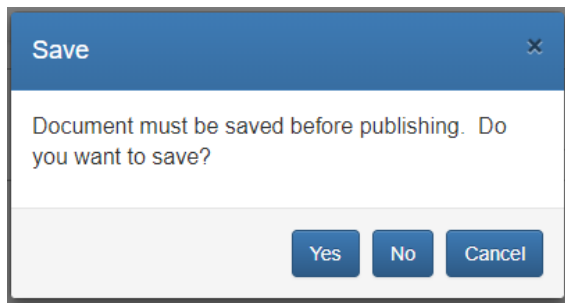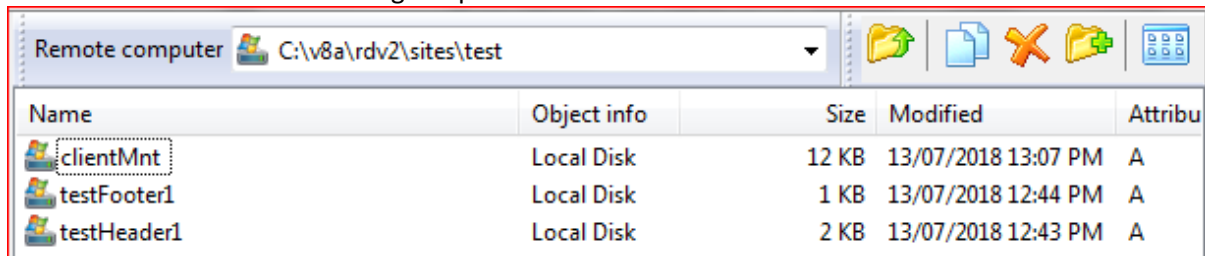After a form is published use the Database Component tool to link database fields and basic routines to the various elements on the form. Refer to *Form Data Link* option on the DesignBais tools menu displayed from the DBIFORMS_DEVELOP form.
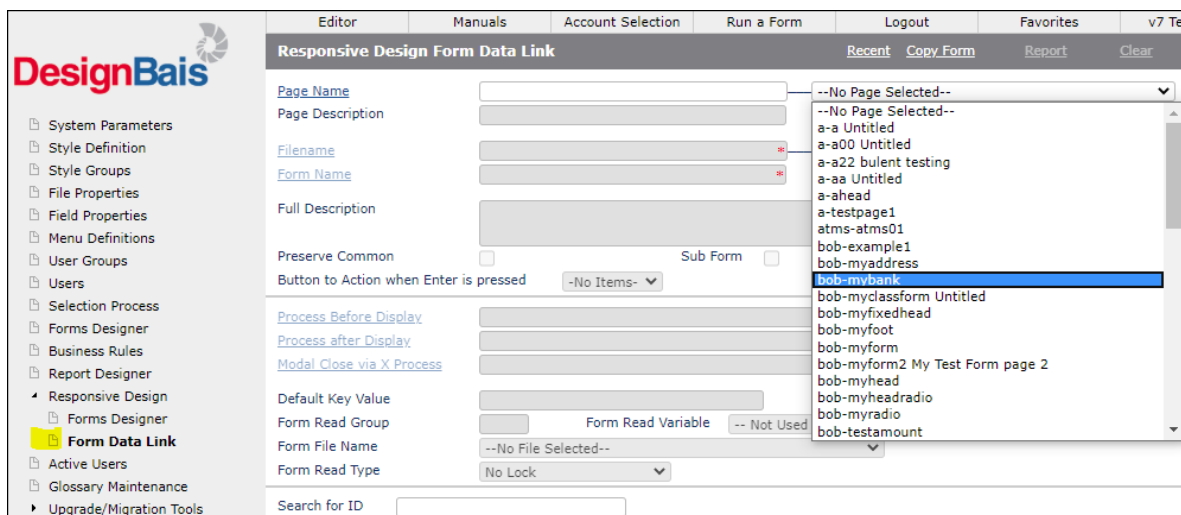
When a page is published the associated header, footer (if they are present) and the template files are all created, or updated, into the DBIFORMHTML file along with the published page record.

Summary of Alt Short Cut keys in alphabetical order for reference:

Alt A   save **A**s
Alt C   **C**lose
Alt D   **D**elete files
Alt E   refr**E**sh
Alt F   new **F**ooter
Alt H   new **H**eader
Alt L   pub**L**ish
Alt O   **O**pen
Alt P   new **P**age
Alt R   p**R**operties
Alt S   **S**ave
Alt W   **W**ork folder

After Options

Control the positioning of new form elements.

After Options: Append
New form elements are appended to the existing form elements.
A new column is placed to the right of existing columns.
A new row is added below existing rows.

After Options: Prepend
New form elements are prepended to the existing form elements.
A new column is placed to the left of existing columns.
A new row is inserted above existing rows.

After Options: Before
New form elements are placed before existing form elements.
A new column is placed to the left of existing columns.
A new row is inserted above existing rows.

After Options: After
New form elements are placed after existing form elements.
A new column is placed to the right of existing columns.
A new row is added below existing rows.

Responsive Design Form Creation

Opening a new page displays one default column stretching across the width of the canvas. This canvas is ready for the developer to create the columns and rows that will house the required form elements.

Fast Options

Most of the available menu options can also be triggered by a control key sequence.

You can use *Ctrl+Z* to undo the most recent action.

Form elements can be inserted using the *Ctrl+key* option as detailed below. Starting from a blank page the following keyboard sequences can be used to rapidly create a responsive design form:

Ctrl+I	Insert a form row in a column
Ctrl+O	Insert a row in a column
Ctrl+J	Insert a column in a row
Ctrl+D	Add a new row
Ctrl+P	Display the properties of a form element
Ctrl+C	Copy a form element
Ctrl+X	Copy and cut a form element
Ctrl+V	Paste a form element
Ctrl+K	Insert HTML element
Ctrl+M	Insert Test Input element
Ctrl+L	Insert Lookup element
Ctrl+A	Insert Address element
Ctrl+B	Insert Button element
Ctrl+S	Insert Select element
Ctrl+R	Insert Radio Button element
Ctrl+H	Insert Checkbox element
Ctrl+E	Insert Text Area element
Ctrl+F	Insert File Upload element
Ctrl+Q	Edit HTML code

Here is an example of the use of these keyboard shortcuts.

Starting with a new page:

Ctrl+I   Insert a form row in a column
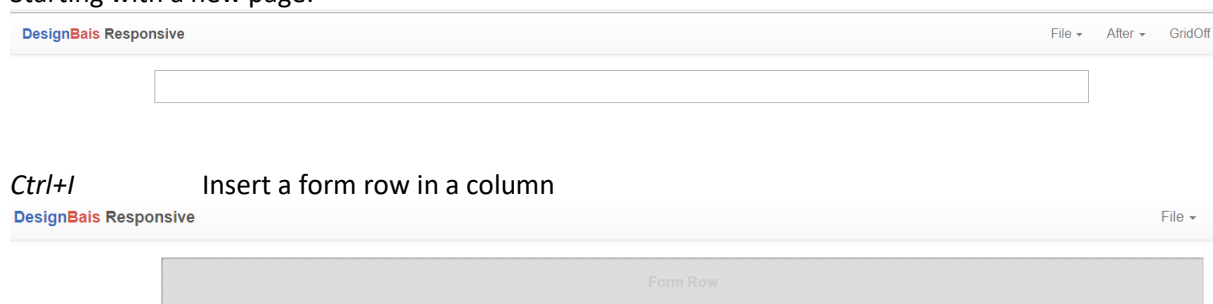
Ctrl+O   Insert a row in a column

DesignBais Responsive   File ▾

Form Row

Ctrl+J   Insert a column in a row

DesignBais Responsive   File ▾

Form Row

Ctrl+D   Add a new row

DesignBais Responsive   File ▾   After ▾   GridOff

Form Row

Ctrl+K   Insert HTML element

DesignBais Responsive   File ▾   After ▾   GridOff

Form Row
Lorem ipsum..

Ctrl+M   Insert Test Input element

DesignBais Responsive   File ▾   After ▾   GridOff

Form Row
Lorem ipsum..
Text   Input

Ctrl+L   Insert Lookup element

DesignBais Responsive   File ▾   After ▾   GridOff

Form Row
Lorem ipsum..
Text   Input
Look Up   Input   🔍

Ctrl+A   Insert Address element

Ctrl+B   Insert Button element

Ctrl+S   Insert Select element

Ctrl+R   Insert Radio Button element

Ctrl+H   Insert Checkbox element

Ctrl+E   Insert Text Area element

DesignBais Responsive   File ▾   After ▾   GridOff

Form Row

Lorem ipsum..

Text	Input	
Look Up	Input 🔍	
Address	Input 📍	
	Button	
Select	Please Select ▾	
Radio	◯ item ⦿ item	
Checkbox	☐ item ☑ item	
TextArea	Input	

In Responsive Design dates are transmitted to the database as YYYY-MM-DD and then converted to the normal format DD-MM-YYYY.

Ctrl+F   Insert File Upload element

The properties of any of the above form elements can be viewed. For example the properties of the *File Upload* element:

Ctrl+P   Display the properties of a form element

Note that the File Upload can be tested in *Preview* mode provided that the form has a button with *Event* and *Page Validation* checked.
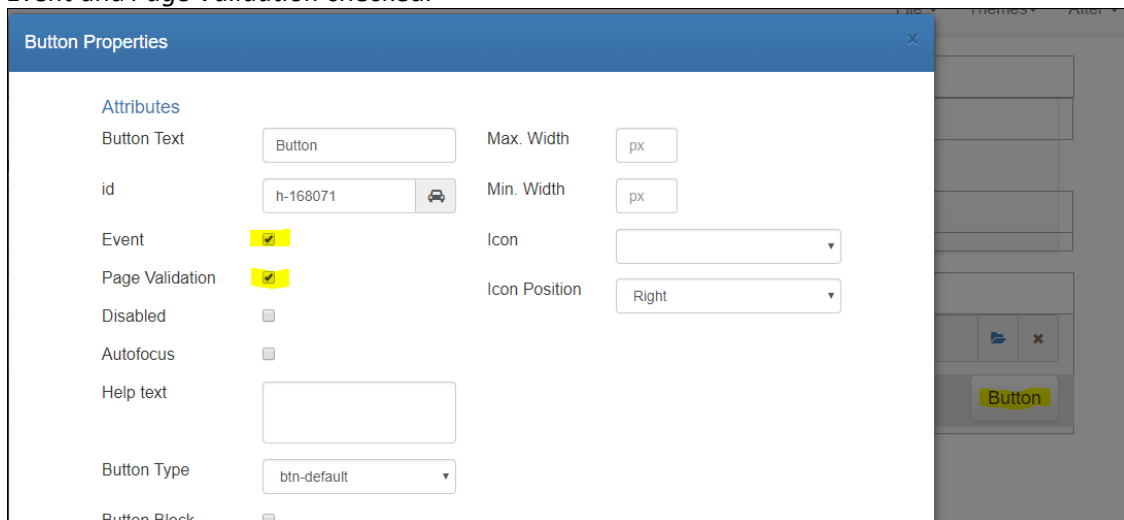
The following example shows a form with a File Upload element and a Button with *Event* and *Page Validation* checked. Using *Preview* (Alt V) opens a new tab with the form active. Select a file by clicking the folder icon.
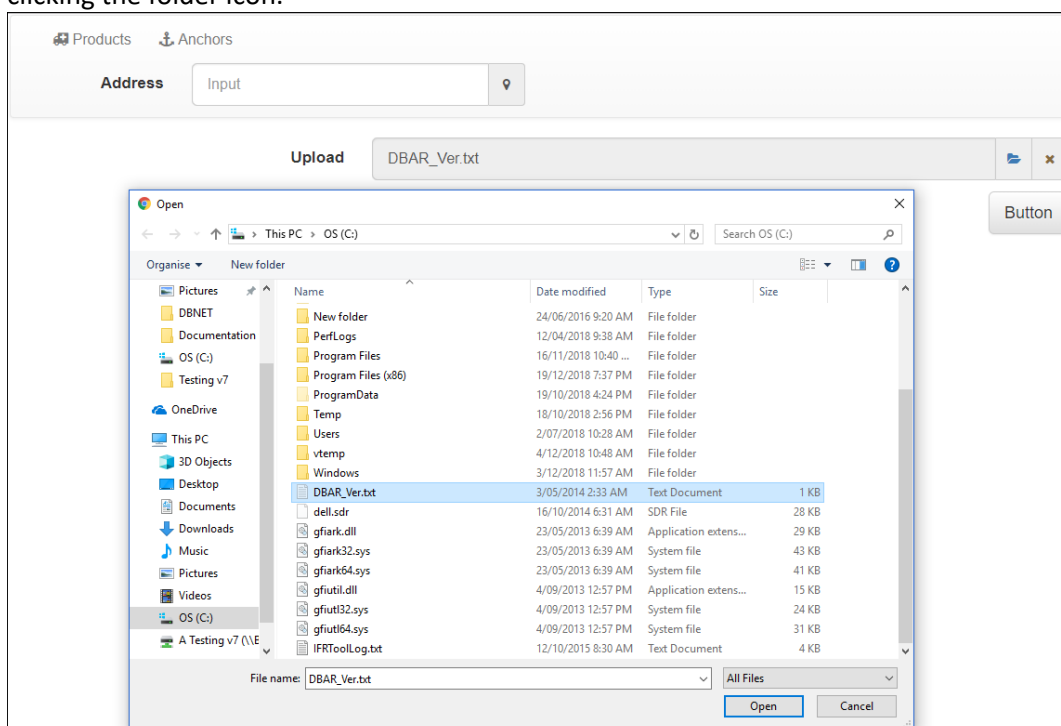
Then click the button. In this case the error message displays to inform that *.txt* files are not permitted to be uploaded.
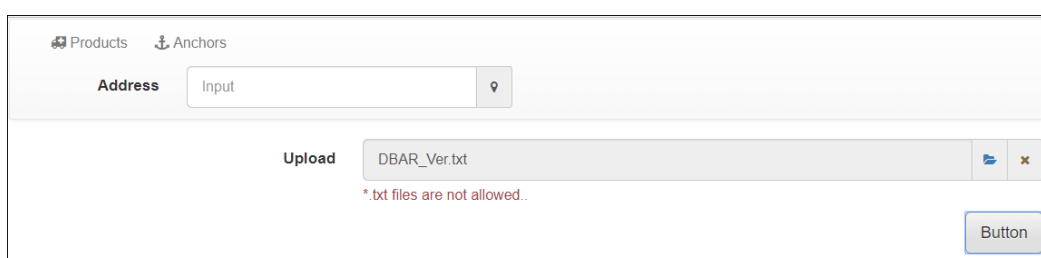
Following are the above options in alphabetical order for reference:

Ctrl+A	Insert Address element
Ctrl+B	Insert Button element
Ctrl+C	Copy a form element
Ctrl+D	Add a new row
Ctrl+E	Insert Text Area element
Ctrl+F	Insert File Upload element
Ctrl+G	Displays the Find String Bar
Ctrl+H	Insert Checkbox element
Ctrl+I	Insert a form row in a column
Ctrl+J	Insert a column in a row
Ctrl+K	Insert HTML element
Ctrl+L	Insert Lookup element
Ctrl+M	Insert Text Input element
Ctrl+N	Opens a new browser instance
Ctrl+O	Insert a row in a column
Ctrl+P	Display the properties of a form element
Ctrl+Q	Edit HTML code
Ctrl+R	Insert Radio Button element
Ctrl+S	Insert Select element
Ctrl+T	Opens a new browser tab
Ctrl+U	Opens DesignBais HTML Editor
Ctrl+V	Paste a form element
Ctrl+W	Prompts to leave site
Ctrl+X	Copy and cut a form element
Ctrl+Y	
Ctrl+Z	Undo the most recent action
Delete	Delete the element that has focus
Esc	To return to Designer after GridOff

New Page with no defined rows

Right click the default row to display *Column* and *Row* menu options. This menu appears when there is no existing row or column.

DesignBais Responsive
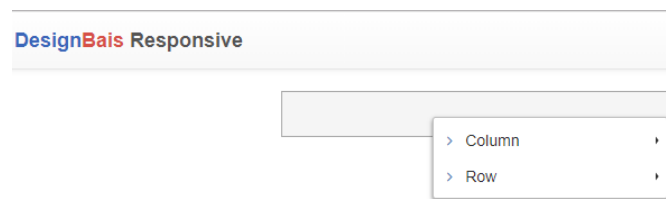
> Column
> Row

Menu Option > Column

Highlight *Column* to display the options.

> Column   > Insert Form Row [Ctrl+I]
> Row   > Insert Row in Col. [Ctrl+O]
                  > Delete Column
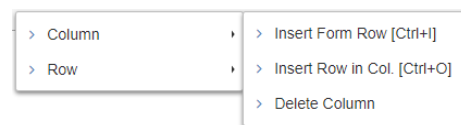
Insert Form Row [Ctrl+I]

Clicking Insert Form Row or Ctrl+I will create a form row. This is a row occupying the width of the column. A form row must exist within a column hence Insert Form Row will always require a column to be created first (the default page is created with 1 default column) and a form row will only extend for the width of the column in which it is created.

Insert Row in Col. [Ctrl+O]

This will create a form row within a column. It implies the creation of a new column and a row within that column.

Note that if there is 1 existing column then this will create a 2nd column of equal width and place a row in the leftmost column. If there are 2 existing columns then this will create a new column within the column that is clicked and add a form row to it.

Delete Column

Deletes the column and any contained elements from the form.

Menu Option > Row

Highlight Row to display the options.

> Column	▸
> Row	▸

> Insert Column [Ctrl+J]
> Adjust Columns
> Add New Row [Ctrl+D]
> Delete Row
> Hide Row
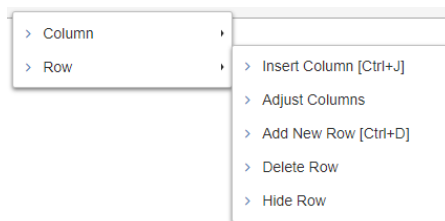
Insert Column [Ctrl+J]   Insert a column in the currently focused row.

Adjust Columns   Opens a form to allow adjustments to column size.

Adjust Columns

-xs	-sm	-md	-lg
3	6	12	12

OK   Cancel

The columns in the above image correspond to:
- Extremely Small (-xs)
- Small (-sm)
- Medium (-md)
- Large (-lg)

These correspond to @media query breakpoints. The screen size of the device running the form determines which of the above columns is used to control the display of the form elements.

Use these settings to control the point at which you want form elements to change from displaying one below another to displaying side by side.

In order to keep the buttons (or any other element) always side by side (in the same row) use the setting shown here:

Note that when -*xs* is used the developer should test to make sure that elements don't exceed the small device screen dimensions.

If the button element is too wide (due to, say, a long caption) then -*md* may be the better setting.

Adjust Columns

-xs	-sm	-md	-lg
6			
6			

OK   Cancel

Note that each row in the *Adjust Columns* display corresponds to a column. So if the row contains three columns, as shown below, there will be three rows displayed. By adjusting the values below from 4, 4, 4 to say 8,2,2 then the buttons can be spaced out along the row as required.
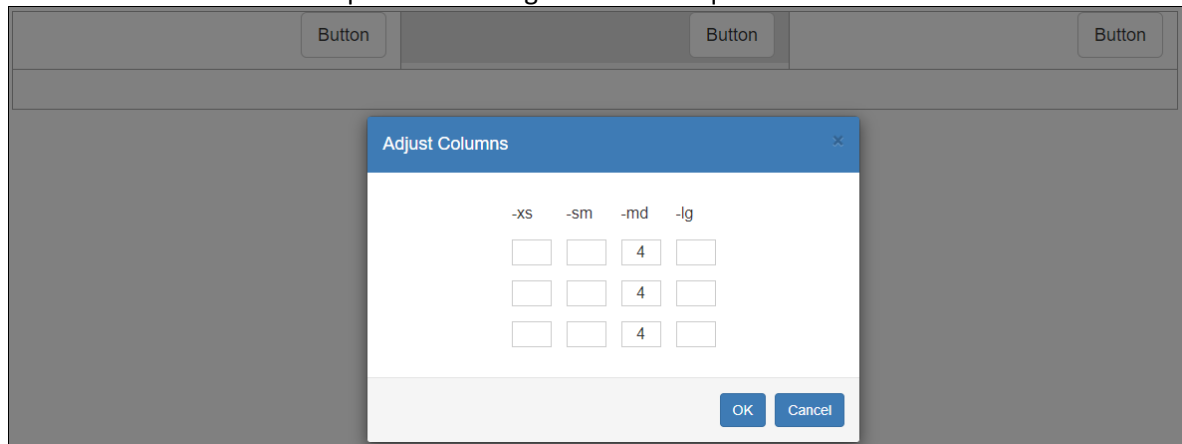
Note that different values can be specified for devices depending on screen size.
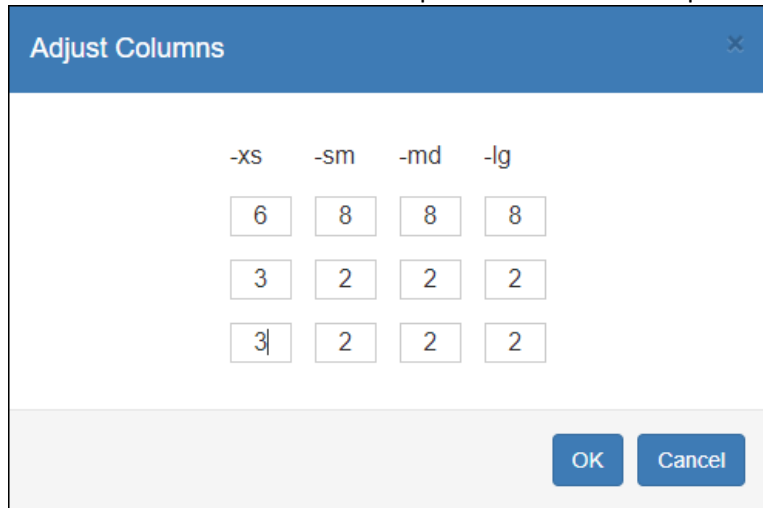
The result of the 8,2,2 values is shown below where the left hand button is centered on a wider column.
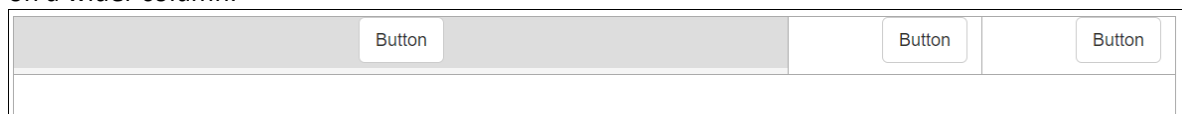
Refer to the section *Button Groups* if buttons need to be placed close to one another on the same row.

Add New Row [Ctrl+D]	Adds a new row. The position of the row will be determined by the setting of top menu *After* option [Append, Prepend, Before, After]. Adding a row
Delete Row	Deletes the currently focused row.
Hide Row	Will hide the currently focused row. Once a row is hidden the option changes to Show all Rows.

Form Row [New Page with defined rows]

When a row is already defined then right clicking the column displays an additional menu option *Form Row*. Hover on *Form Row* displays the options *Insert Element* and *Delete [Del]*.

Insert Element

Use the *Insert Element* option to place fields on the form.
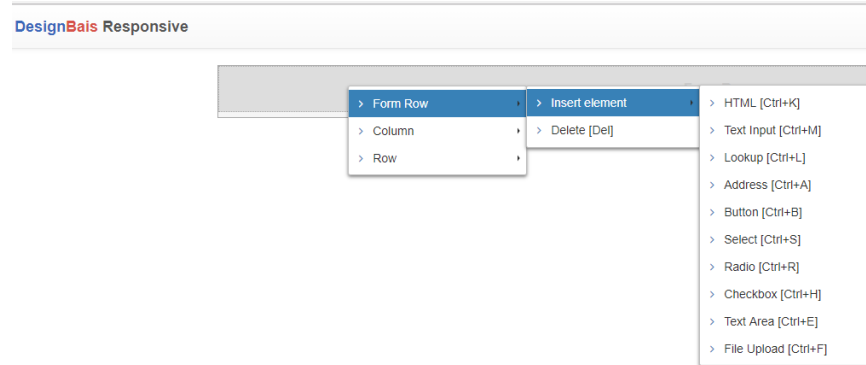
See below for a description of how to create each element.

Delete: [Del]

Selecting the *Delete* option deletes the focused form element. There is no prompt to request confirmation.

Column

Hover on *Column* displays the following options.
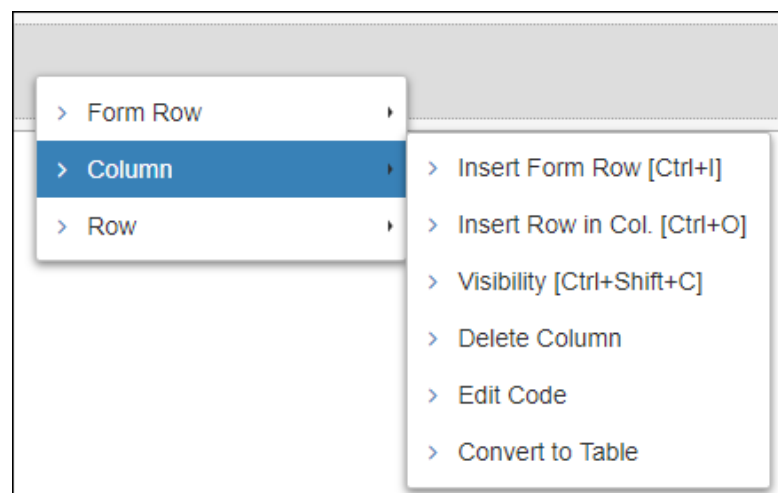
Insert Form Row [Ctrl+I]

Inserts a form row.

Insert Row in Col [Ctrl+O]

Inserts a form row in the focussed column.

Visibility [Ctrl+Shift+C]

Allows the developer to determine whether a form element is to be visible on various screen sizes.
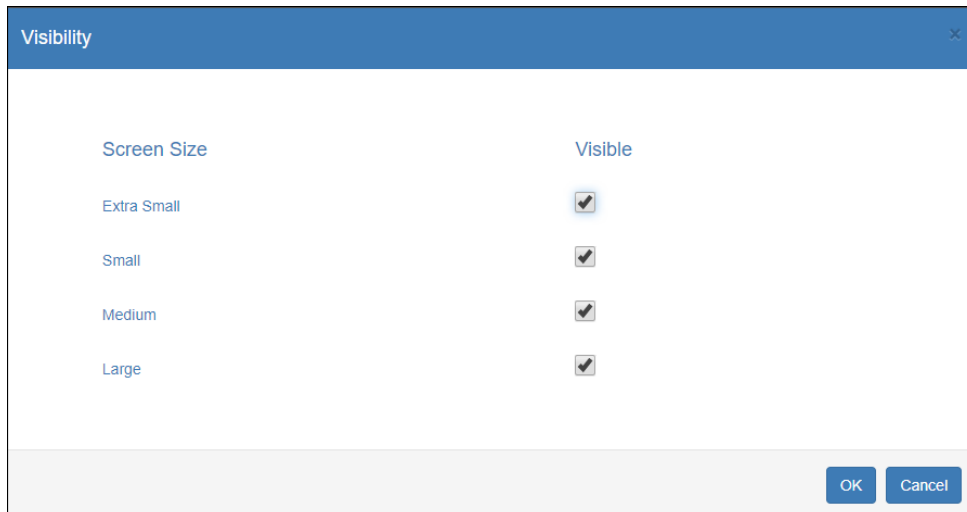
Delete Column

Deletes a column.

Edit Code

Display and maintain the HTML code for the form.

Convert to Table

To create a table commence with any form element such as a Text Input. This can be created using Ctrl-M from within a form row. Then press Ctrl-I to create a form row within the same column. Press Ctrl-M to create a Text Input form element, or create any other required form element type.

When you have the number of form elements corresponding to the number of table columns required right-click on any form element and select *Convert to Table*.
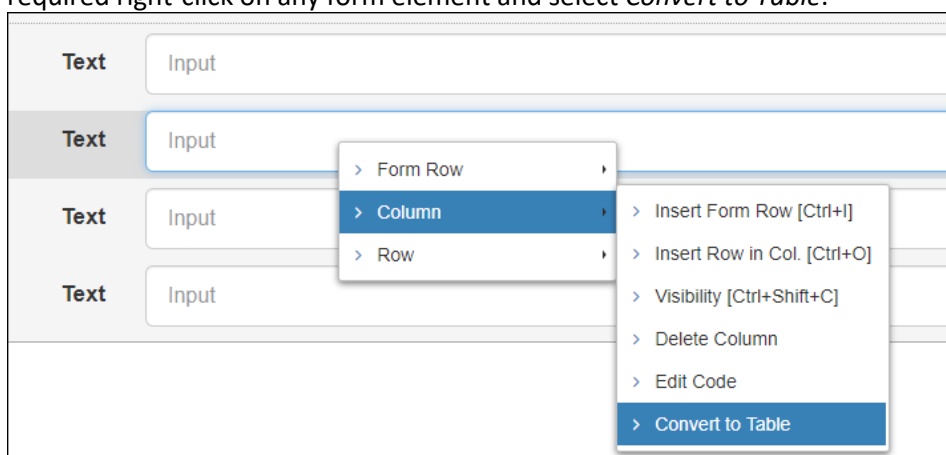
This will create the table.

Right-click any form element:

And select *Properties* or press Ctrl-P.

Replace the default *ID* with a meaningful id of your choosing. This makes the form data linking process more straightforward.

Responsive Break Point   The available options are:
- Not responsive
- Extra Large
- Medium
- Small
- Extra Small

Column No.   Select the required column for which the properties are to be set.

Alignment   The available options are:
- text-right
- text-center
- text-left

Text	Enter the text required for the column heading.
Set	Click the *Set* button to save the details for each column.

If you wish to revert to form rows then right click on a Table to display the available options.

Refer to the **Table Row Control** section below which describes how to insert buttons for row insert and delete.
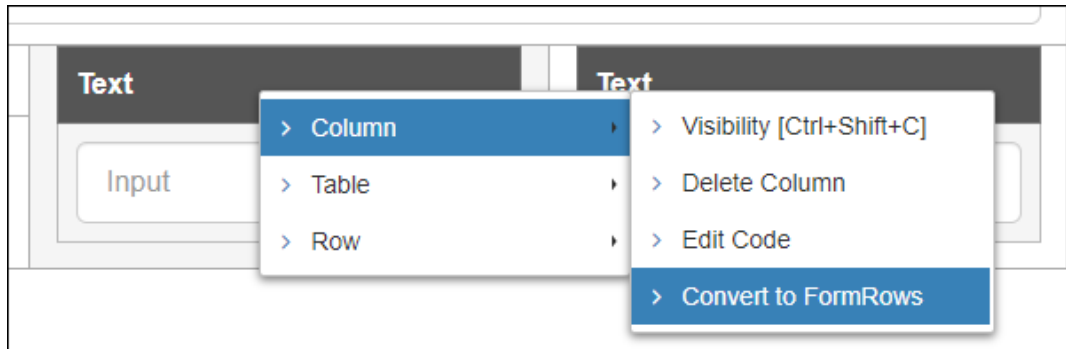
The Visibility, Delete Column and Edit Code are described elsewhere. An example of the use of a Table Row can be found here #Example of Table Row with Checkbox

Convert to FormRows

This option will return the table form elements to form row elements.

Table Row Control

As part of setting up a Table it is usually necessary to set up a method to insert, delete and add rows to the grid.

This is achieved by adding button elements to the table from within the RD designer. In the example below three buttons have been added. These correspond to the grid control buttons in the top left corner of a non-RD DesignBais form multi-value grid.

In the Form Data Link option, when defining these buttons, select the required button type from the dropdown list.

Do not alter the assigned Button Name. The button name is set so as to link in with the standard DesignBais multi-value grid row control function.

Row

Hover on *Row* displays the following options.
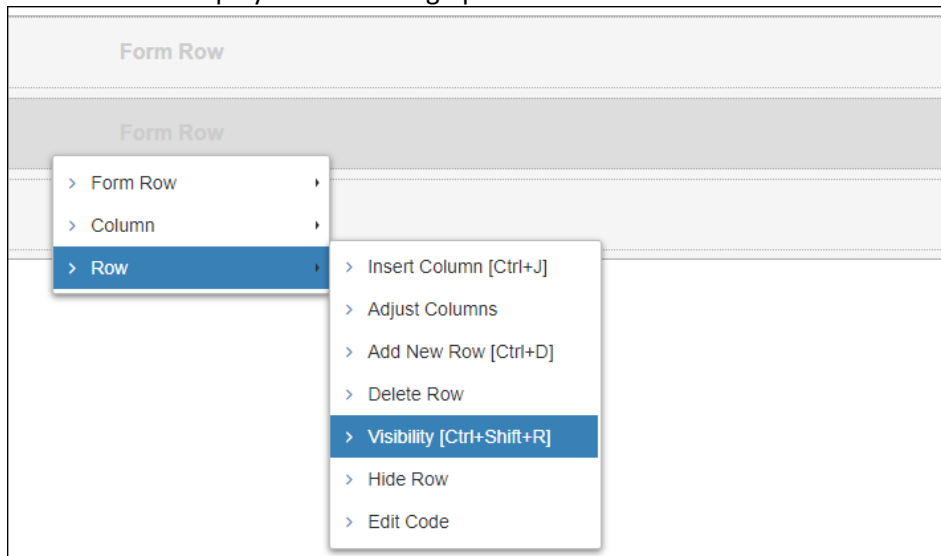
Insert Column [Ctrl+J]

Inserts a new column.

Adjust Columns

See *Menu Option > Row* for details.
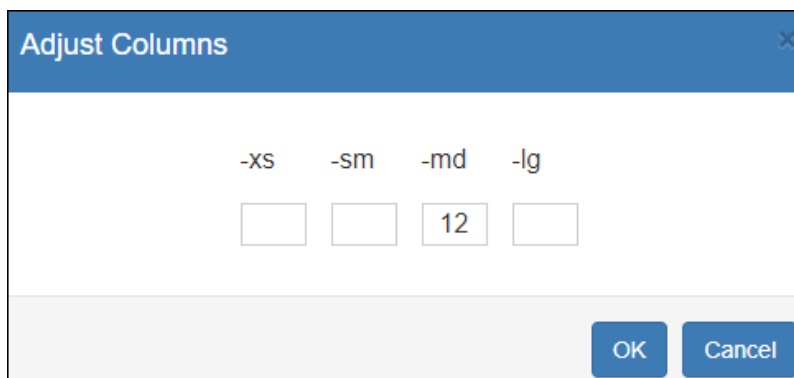
Add New Row [Ctrl+D]
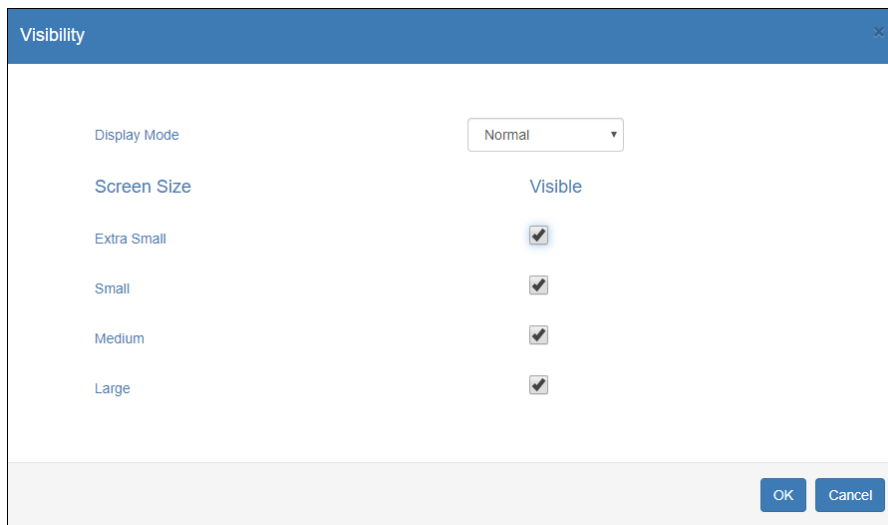
Adds a new row. Use the top menu option *Append, Prepend* to control where the row is placed.

Delete Row

Deletes the form row.

Visibility [Ctrl+Shift+R]

Allows the developer to determine whether a form element is to be visible on various screen sizes.

In addition, for row visibility, you can set the *Display Mode* as either *Normal, Initially Hidden* or *Modal*.

Initially Hidden	The form element is hidden until an event or condition, triggered by the form processing subroutine, causes the element to be displayed.
Modal.	

Hide Row

The row will be hidden. In order to see the row again right-click and select *Row* then *Show All Rows*.

Edit Code

Setting Table Column Width

Table column widths are controlled by the maximum width of components

For output fields (i.e. HTML), you need to edit code and add the max-width style attribute of the HTML segment as follows:

Edit Code

```
i 1▾ <div class="dbformgroup form-group dbrowcolhilite" custom="true">
  2▾     <div dbtype="segment" class="html-text" id="d277347" style="max-width:100px">
  3           <p>Lorem ipsum..</p>
  4       </div>
  5 </div>
  6
```

Insert Element

Right click a column to display the available options.
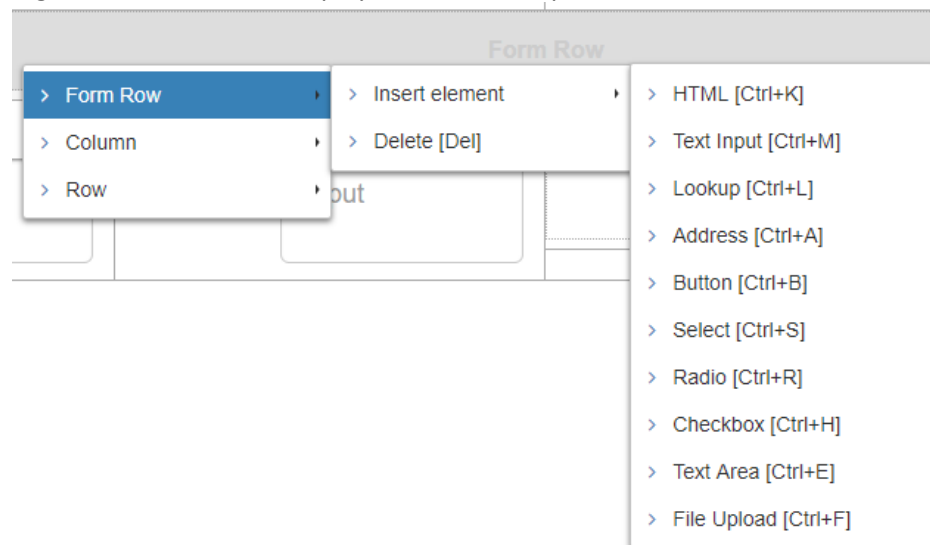
There are ten form elements. Each element is described below. An element can be inserted by clicking the required menu option or by pressing concurrently the *Ctrl* key and the letter designated for each form element. Using the *Ctrl* key method does not require the menu to be displayed.

Assigning Element Ids

When assigning the element id for a form element the following should be noted.

Avoid using any of the following strings anywhere in the name of the element (the id). The DesignBais engine looks for these strings to determine particular processing paths:

- *button*
- *label*
- *img*

Care should be taken when using the lower case letters which are used by DesignBais as separators within XML label ids. It is best to avoid these lower case letters in your element ids:

- *v*
- *x*
- *z*

Insert HTML: [Ctrl+K]

Use this element to create an output field on a form. Note that a click on an output field in RD will trigger a button event in the database component. If a database field is linked to an HTML form element then the content of the linked field will control the display of this element on the RD form.

Attributes

id　　　　　The id for this form element. Must be unique within this form.

Style　　　　Select the required style.

Size　　　　Select the required size.

Alignment	Select the required alignment.

Event	Set to on if an event is to be associated with this field.
HTML Text	Enter text as required.
Tip Text	If text is entered here it causes a question mark character to be displayed after the *Label*. Clicking this question mark character at run time displays the *Tip text* in a help bubble overlay.

Example

Maintaining Existing HTML Text Element

Right clicking an existing HTML Text element displays a menu to review Properties, Edit Code, Edit HTML text, Copy, Cut or Delete.
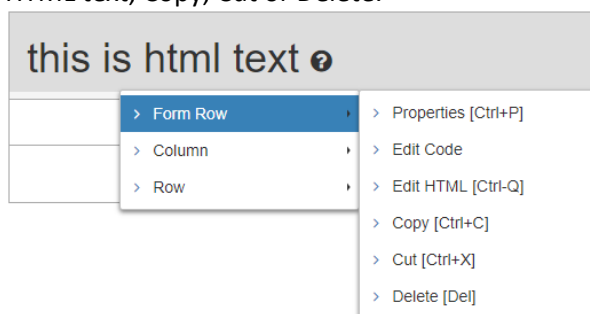
Edit Code

This option id designed for specialists who know how to create and amend HTML code. Note that if you edit the code you will not be able to access the Properties of this element. All future changes will also have to be implemented by editing the code.

Edit HTML

This method allows non-specialists to amend the HTML code with the assistance of an intelligent editor.

Use this method to set things like the justification of the element.

HTML Editor

this is html text

Cancel   Save
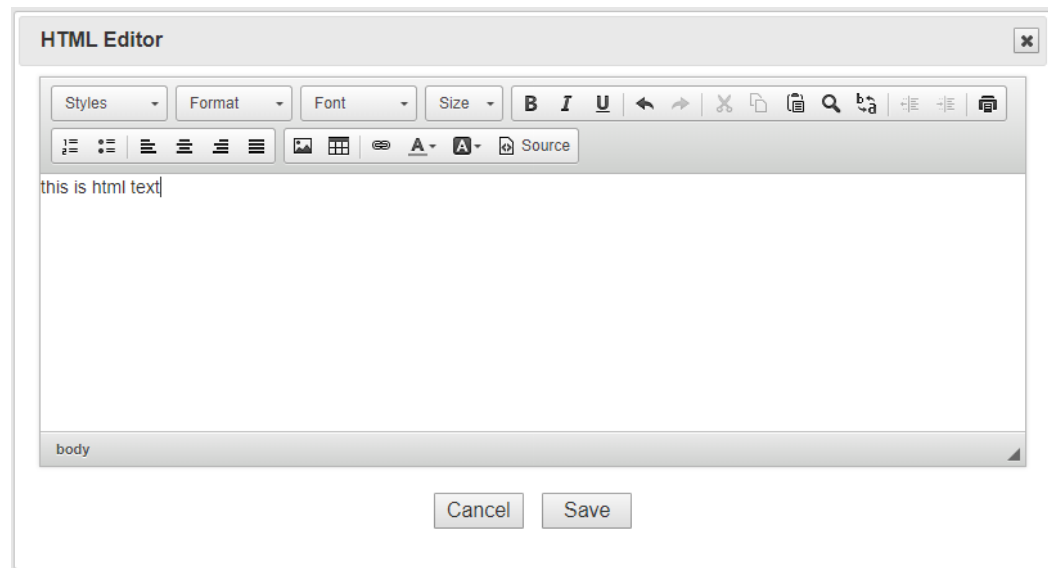
Copy [Ctrl+C]

Use this option to copy a form element which can then be pasted elsewhere on the page.
Note that the element id is re-assigned when an element is cut and pasted.

Cut [Ctrl+X]

Use this option to cut and copy a form element which can then be pasted elsewhere on the page.

Delete [Del]

Use this option to delete a form element from the page.

Output Only Form Element

It is possible to link an event to an output only field.

In Form Data Link the "field name" is set to the XML label so that it can be differentiated from an output field linked to a database field.

HTML Section Properties

Attributes

id: `outputEventOnly`

Style: `Normal ▾`

Event: ✔

HTML Text: `<button type="button" "="">Click Me</button>`

Tip Text:

OK   Cancel

192.168.199.194/dbnet/?ac=rd

Login   RD Test   uvRHat   v7 UD   Pulse Test   Pulse legj   G Google

Client Code ❓ *   Input

Name *   Input

Address   Input

Notes   Input

DesignBais

Click Me

Test Field   Test Field

The BUTTON event in the code does a DBDS:

When building tables in RD the developer can use OFR style IDs in order for DesignBais to recognise these IDs. DesignBais will populate the DBREPORT.CELL common variable with the row and column position of the form element that is clicked. The developer can then use on form report type element ids such as id="tdxoutputEventOnlyxrowzcol" and **not** id="fred" or id="tdxfredxrowzcol".

An id such as "tdxoutputEventOnlyx2z1" will signify that the first column of the second row of the table has been clicked.

Chrome replaces the current tab.

Inserting an Image

Use the HTML element to insert an image. Click the *Image* icon.
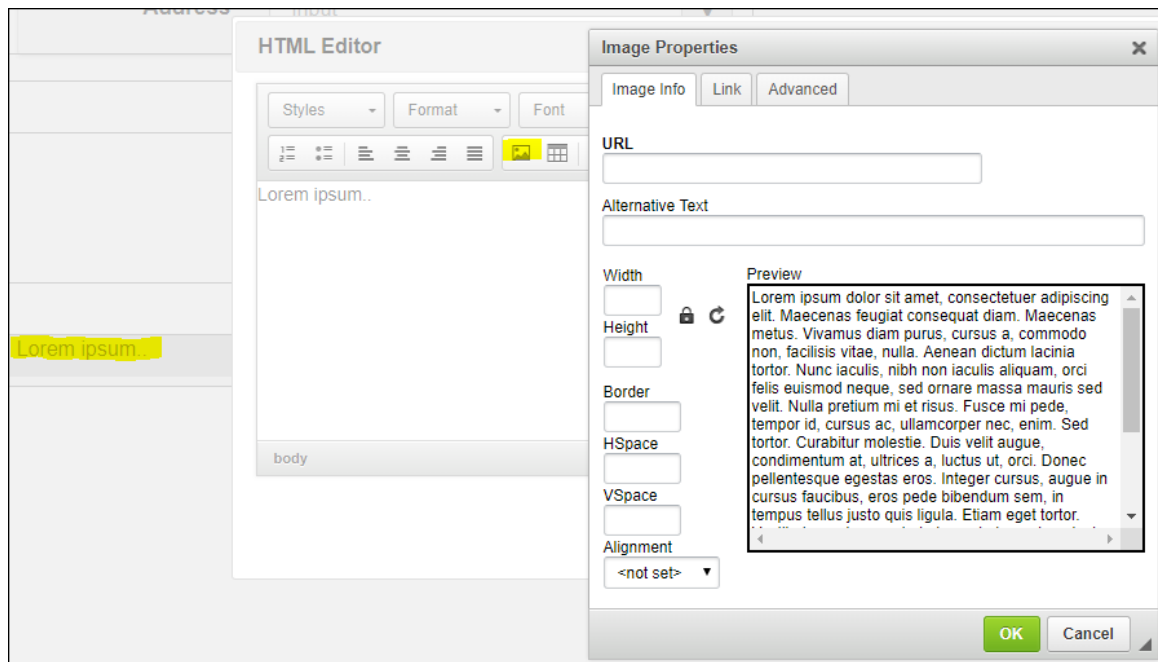
URL   In inserting an image, always start the image URL with "../images".

                  The "../images" points to the c:\your_website\images folder.

                  For demonstration, consider the folder:
                          c:\DBNET\images\rddemo
                  has some images stored in it. For example hol1.jpg.

                  The URL becomes: *../images/rddemo/hol1.jpg*
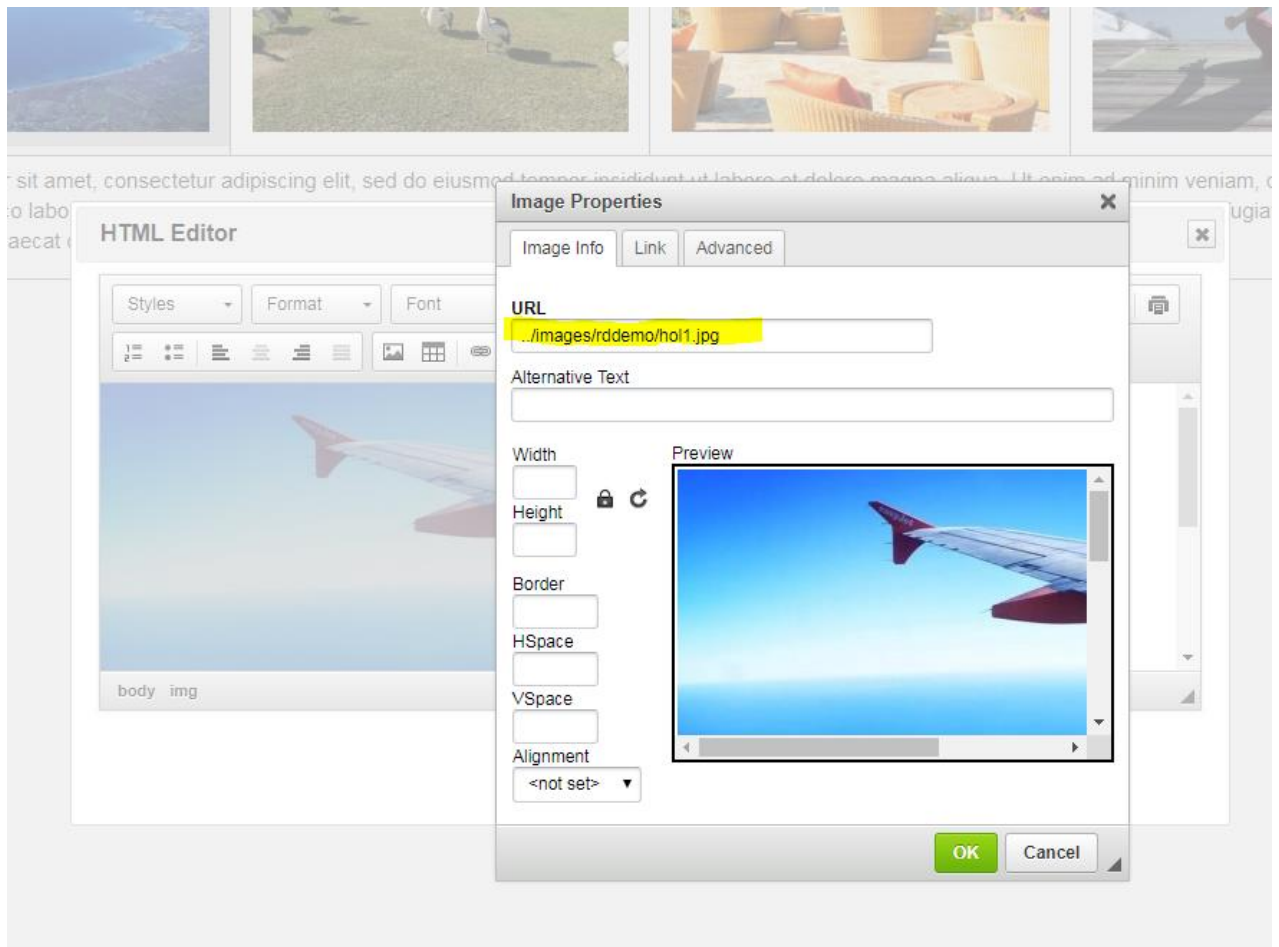
Image Dimensions

Width
Height

Delete the width and height from the properties of the image dialog shown above. On detecting that the Responsive Design tool makes the image "responsive".

If the developer does not want the image to be "responsive" then leave the original dimensions intact.

Insert Text Input: [Ctrl+M]

Properties

Label   Enter the label for this input field. For example *Last Name*.

Id   The id for this form element. Must be unique within this form. Click the *Auto* icon to create the id from the Label Text.

Placeholder   The text that appears in the field prior to entry of data by the user. Use this to indicate the nature of the required input.

Event   Set to on if an event is to be associated with this field.

Disabled   Set on to disable this field.

Readonly   Set on to make this field *Read Only*.

Autocomplete   Set on to enable *Autocomplete* behaviour for this field.

|	
Autofocus	Set on to place focus to this field on entry to the form.
Help text	Enter text, that will appear under this field, to assist a user to complete the entry of this field.
Max. Width	The maximum width of thie field in pixels.
Input Size	The font size, either *Large, Medium*, or *Small*.
Tip text	If text is entered here it causes a question mark character to be displayed after the *Label*. Clicking this question mark character at run time displays the *Tip text* in a help bubble overlay.
Icon	Select an Awesome Class from the list or leave empty. Awesome font icons are available on the web.
Text Alignment *Box Alignment*	The justification for this field, either *text-right* or *text-left* within the column.
Label span *Input span*	The proportion of this column, from a total of 12 units, that is to be assigned to the label with the balance being assigned to the actual input field. If the *Label span* is assigned a value of *X* then the *Input span* is set to *12 − X*. The above defines the default scenario. It is possible to assign values to the Label and Input spans that sum to a number greater then 12. In this case the Label will be placed above the input field. Similarly the values assigned can sum to a number less than 12 which leads to the label and input occupying a smaller proportion of the available column width.

Test Area

After changing the above properties click the *Submit (simulation)* button. This Text Area input field allows the developer to test input in this form input field.

Validation

Mandatory   Check the box to make this field mandatory. You will probably want to set the *Page Validation* flag on your *Submit/Next Page* button as well.

Input   The type of data.

In Responsive Design dates are transmitted to the database as YYYY-MM-DD and then converted to the normal format DD-MM-YYYY.

Validation   The type of validation to be applied to the data entered in this field.

The URL validation check is done by JavaScript's internal function. It only checks that the value matches *string1:string2*, such as **protocol**"**:**"**domain.** Note that for the *Url* validation to work the *Input* type must also be set to *Url*.

Input Mask

Allow All	This field allows all characters to be input.
Allow [a-z]	Only lower case alphabetic characters can be input. Use this option for field properties that have an Input conversion setting of *Lowercase*.
Allow [A-Z]	Only upper case alphabetic characters can be input. Use this option for field properties that have an Input conversion setting of *Uppercase*.
Allow Numbers	Only number characters can be input.
Allow Chars.	Specify those characters that are permitted to be entered.
Input Mask	If required enter the mask to be used to format the data entered into this field. For example *aa-nnn* will require the entry to be formatted as 2 alpha characters, a hyphen, then 3 numeric characters.
Min. Chars	Specify the minumum number of characters that must be entered.
Max. Chars	Specify the maximum number of characters that can be entered.

Test Area

After changing the above properties click the Submit (simulation) button. This Text Area input field allows the developer to test input in this form input field.

Buttons

Submit (simulation)

Apply changes to the Test Area only. Changes are not updated to the form.

OK	Exit the properties form and update all changes.
Cancel	Exit the properties form without updating any changes.

Datalist

Not a datalist   The default option where the input field is not a datalist.

Database Look up   Select this option where the input is selected from a dropdown list.
The following options apply.

Allow selection from the list items only

Limit field input to items from the displayed list. If not set then the user can enter data that is not in the list. Validation of this input may be required using bespoke code. Tick the *Event* option on Page 1 and attach a process such as a basic subroutine if required. This is done via the Database Component option *Form Data Link*.

Min. trigger length

The number of characters that are entered before predictive / autocomplete text is displayed.

Max. number of results

This value limits the number of result lines of predictive / autocomplete text that are displayed.

Geolocation   Signals that this input field uses Geolocation. If the website is configured to allow the site to determine the location of the user then this location will be utilised by this field.

Test Area

After changing the above properties click the Submit (simulation) button. This Text Area input field allows the developer to test input in this form input field.

Buttons

Button ID	Assign the required element ID. This will be the Id that displays in the *Form Data Link* form.
Button Text	The text to appear on the button.
Add (button)	Click to add the button to the input element. The button Id will appear in the Added Buttons display area.
Double click to Delete	Highlight the button element Id in the display area and double click to delete the button.
OK	Click OK to save the entered or changed details.
Cancel	Click Cancel to exit without changing the details.

Why use Buttons

Developers may be used to the input field validation or read event being triggered by the user tabbing out of the field. On small devices like phones there is no *Tab* key and therefore validation or read events must be triggered by a *click* event. These buttons provide the click event. When using buttons like this it is not necessary to click the *Event* checkbox on *Page 1* of the input element properties. If this checkbox is checked then there will be a validation event as well as a button click event.

Use the *Form Data Link* function to link the button element to a subroutine process.

In the subroutine you can process the required action in the *BUTTON* event where the *PROCESS.EVENTSOURCE* will be set to the name of the button, *Button Name* in the above image, not the *Field Xml Label*.

```
BUTTON:
  BEGIN CASE
    CASE EVENTSOURCE = 'B.BTNREAD'
      * Need the Email Validation Logic
      EVENTSOURCE = "DBC.CLIENT.CODE.WK"
      DBVALUE = DBWORK< DBC.CLIENT.CODE.WK >
  END CASE
  RETURN
```

Note that if the button is to trigger a read then the DesignBais form field that the read is linked to MUST use DBWORK as the read *variable to use*. This is so that the key value, entered into the field on the form by the user, will be in a DBWORK field when the button is clicked. This allows the code shown above to work by setting DBVALUE = DBWORK<DBC.CLIENT.CODE.WK>.

Insert Lookup: [Ctrl+L]

Properties

Label	Enter the label for this input field. For example *Last Name*.
Id	The id for this form element. Must be unique within this form.
Placeholder	The text that appears in the field prior to entry of data by the user. Use this to indicate the nature of the required input.
Event	Click to set on if an event is to be associated with this field.
Disabled	Set on to disable this field.
Readonly	Set on to make this field *Read Only*.
Autocomplete	Set on to enable *Autocomplete* behaviour for this field.

Autofocus	Set on to place focus to this field on entry to the form.
Help text	Enter text, that will appear under this field, to assist a user to complete the entry of this field.
Max. Width	The maximum width of thie field in pixels.
Input Size	The font size, either *Large, Medium*, or *Small*.
Tip text	Text text causes a question mark character to be displayed after the *Label*. Clicking this question mark character at run time displays the *Tip text* in a help bubble overlay.
Icon	Select an Awesome Class from the list or leave empty.
Text Alignment *Box Alignment*	The justification for this field, either *text-right* or *text-left* within the column.
Label span *Input span*	The proportion of this column, from a total of 12 units, that is to be assigned to the label with the balance being assigned to the actual input field. If the *Label span* is assigned a value of *X* then the *Input span* is set to *12 − X*. The above defines the default scenario. It is possible to assign values to the Label and Input spans that sum to a number greater then 12. In this case the Label will be placed above the input field. Similarly the values assigned can sum to a number less than 12 which leads to the label and input occupying a smaller proportion of the available column width.

Test Area

After changing the above properties click the Submit (simulation) button. This TextArea input field allows the developer to test input in this form input field.

Type   Select the required option:

Not a datalist:	No call to the database is required.
Database Look up:	A change event will be raised when the field's value changes.
Geolocation:	Use where this is an address or location field in order to invoke Google location services.

Implementing Lookup using a subroutine

RD lookup fields are code value pairs much the same as a file lookup or value list used for dropdown lists.

In the Form Data Link process enter the name of a database file in the *Lookup File* field for the *Lookup* form field element.

DesignBais builds the normal dropdown combo box record on DBISESSIONS and uses this to locate the current code value in the list, and can then return the associated description. The DBISESSIONS record id is *SESSION.ID*FIELDNAME*COMBO.* In the above case the id is SESSION.ID*DBC.CLASS*COMBO.

If there is no Lookup File defined then the developer must provide code in the *LOOKUP* event processing paragraph in the basic subroutine. The string entered by the user will be in DBVALUE. The developer can decide what to return to the form based on this string.

```
PROCESS.EVENT = "LOOKUP"
PROCESS.EVENTSOURCE = Field name
PROCESS.PARAMETER = FIELD.AFTER.PARAMETER
```

After text is typed in the lookup field in the RD Form it is passed to the subroutine in DBVALUE so that the developer can build a list of codes and labels in DBDROPLISTADD.

```
DBDROPLISTADD <2> = Multivalued codes
DBDROPLISTADD <3> = Multivalued associated descriptive labels
```

The correct rdLookupList command is constructed from the returned values. There is no need for the developer to include this in their code.

Once an option is selected the normal change event is fired and the VALIDATE event will be passed to the subroutine.

Example Code:

Set up the new event in the event processing case statements

```
        CASE THIS.EVENT = "LOOKUP"
           GOSUB LOOKUP
```

Example code for the LOOKUP event processing:

```
LOOKUP:
     BEGIN CASE
        CASE SCREEN.NO = "MAINT" AND EVENTSOURCE="DBC.CLASS"
           SRCH.VALUE = DBVALUE
           * based on the SRCH.VALUE a list of options for the user to select from
           * is passed in DBVALUE
           * DBDROPLISTADD<1> (normally the multivalued field list) is not required
           DBDROPLISTADD = ''
           DBDROPLISTADD <2> = 'A':VM:'B'
           DBDROPLISTADD <3> = 'Class A':VM:'Class B'
     END CASE
     RETURN
```

Important note: you cannot use DBDS to display variables when implementing a lookup element. DBDS stops the display of the lookup dropdown selection list.

If the user enters a string that is not found in the lookup table then the following message box is displayed:
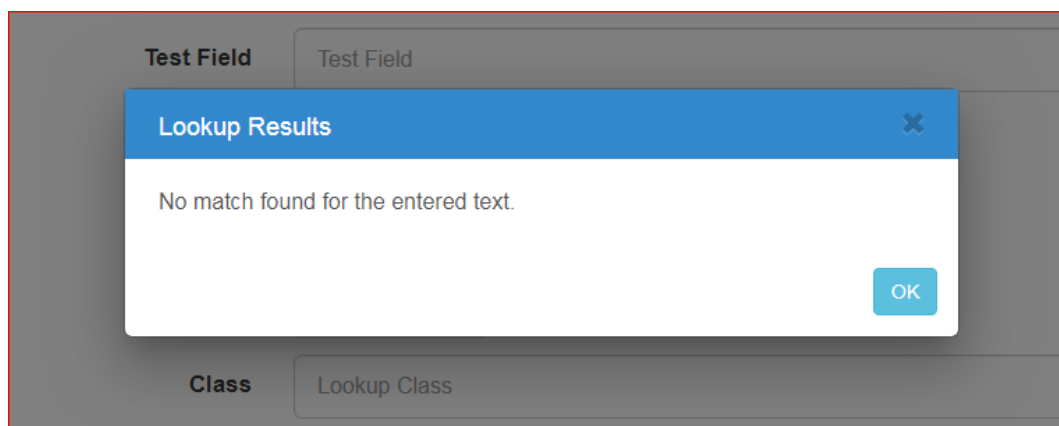
In the case where the developer needs to inhibit the DesignBais Data Component from auto filtering the results of the lookup then set DBVALUE = "DBNOFILTER". See the example below:

```
LOOKUP:
     BEGIN CASE
        CASE SCREEN.NO = "MAINT" AND EVENTSOURCE="DBC.CLASS"
           SRCH.VALUE = DBVALUE
           * based on the SRCH.VALUE a list of options for the user to select form
           * is passed  in DBVALUE
           * DBDROPLISTADD<1> (normally the multivalued field list) is not required
           DBDROPLISTADD = ''
           DBDROPLISTADD <2> = 'A':VM:'B'
           DBDROPLISTADD <3> = 'Class A':VM:'Class B'
           DBVALUE = "DBNOFILTER"
     END CASE
     RETURN
```

The default behavior of DesignBais Web Component is to filter out the entries retrieved by the lookup that don't match the search pattern. To override this the developer can use an ajax call setting the "nofilter" attribute:

```
        rdSetAttribute(elementID,"element","nofilter", "true")
```

Use a call to DBI.G.AJXCMD as shown below:

```
AJXIN = 'VesselLocation'
AJXIN<2> = 'element'
AJXIN<3> = 'nofilter'
AJXIN<4> = 'true'
CALL DBI.G.AJXCMD('SA',AJXIN)
```

This results in the lookup displaying dropdown entries that do not exactly match the search string but are considered relevant to the search.

The LOOKUP LABEL event is used to set the description for the code selected from a lookup if the process is being handled within a basic subroutine.

For example, in this LOOKUP LABEL event the code that the user selected is in DBVALUE. Read this record from the database and set DBVALUE as required. In this case the DBVALUE is changed from just the selected code value to a description made up of the music title and the artist:

```
LOOKUP.LABEL:
    BEGIN CASE
      CASE SCREEN.NO = 'CATALOG1' AND EVENTSOURCE = 'MUS.ID'
        OPEN 'DBMUSIC' TO F.DBMUSIC ELSE RETURN
        READ REC FROM F.DBMUSIC,DBVALUE ELSE
          REC = ''
          REC<MUS.TITLE> = DBVALUE:' not found'
        END
        DBVALUE = REC<MUS.TITLE>[1,50]:' by ':REC<MUS.ARTIST>
        DBRETURN.TO.FIELD = 'MUS.ARTIST'
    END CASE
    RETURN
```

If you want to allow the user to enter text, in the lookup field, that is not from the dropdown list then use the following when loading the page:

```
AJXIN = 'VesselLocation'
AJXIN<2> = 'element'
AJXIN<3> = ' listonly'
AJXIN<4> = 'true'
CALL DBI.G.AJXCMD('SA',AJXIN)
```

Insert Address: [Ctrl+A]

Input

Page 1 | Page 2 | Page 3

Properties

Label Text	Address	Max. Width	px
id	[🖨]	Input Size	Large ▾
Placeholder	Input	Tip text	
Event	☐		
Disabled	☐	Icon	map-marker 📍 ▾
Readonly	☐		
Autocomplete	☐	Alignment	text-right ▾
Autofocus	☐	Label span	4
Help text		Input span	8

Test Area

Address   Input   📍

Submit (simulation)   OK   Cancel

Input

Page 1 | Page 2 | Page 3

Validation

Mandatory	☐
Input	Text ▾
Validation	No Validation ▾

Input Mask

Allow All	☑
Allow [a-z]	☐
Allow [A-Z]	☐
Allow Numbers	☐
Allow Chars.	
Input Mask	e.g. nnn-nnn
	Use "n" for numerics and "a" for letters.
Min. Chars	
Max. Chars	250

Test Area

Address   Input   📍

Submit (simulation)   OK   Cancel

In order to implement the Google Address lookup when the Datalist type is set to:

Geolocation - invoke Google location services for address

the website db.config file must include the *Google Places API Key*.

Obtain the key from Google and plug it into db.config in the <setup> node:

```
<setup>
    ...
    ...
    ...
    <RDPlacesKey>xxxxxxxxc5SaYsIyyyyyyyyyyyyyyyQtt2bKyFDMA</RDPlacesKey>
</setup>
```

Google location bias

Location bias is automatic if using https (the user must accept the dialog)..
If there is no https, or you want to be on the safe side then send:
rdSetAttribute("element_id", "element", "countryFilter", "aus");

Note that "element id" in the parameter list must contain the id of the address element in the form.
The strings "element", "countryFilter" are fixed strings. "aus" refers to Australia. For multiple
countries use a pipe delimited list. Example "aus|nzl" for Australia and New Zealand.

Address handling on the database file when using Google Address lookup

The address returned by the Google address lookup will be formatted like this:

street_number|^|265|#|route|^|Darling Street|#|locality|^|Balmain|#|admin istrative_area_level_2|^|Inner West Council|#|administrative_area_level_1 |^|New South Wales|#|country|^|Australia|#|postal_code|^|2041|#|formatted _address|^|265 Darling St, Balmain NSW 2041, Australia

It is essential that your database address field store the address data in this format. To display the formatted address use basic text extraction commands something like this, assuming that the variable *RAW.ADDRESS* contains the entire Google address:

```
FLD.SEP = "formatted_address"
IDX = INDEX(RAW.ADDRESS,FLD.SEP,1)
IF IDX THEN
    FMT.ADDR = OCONV(RAW.ADDRESS [IDX,LEN(RAW.ADDRESS)-IDX+1],'G2|1')
END ELSE
    FMT.ADDR = RAW.ADDRESS
END
DBRECORD<BNK.ADDRESS.FMT> = FMT.ADDR
```

The *FMT.ADDR* is thus assigned the value:

265 Darling St, Balmain NSW 2041, Australia

Note that if address is to be used as a Dropdown Description field then using the raw Google address field will give an unacceptable result as shown on the right.

It is necessary to assign a field property name to an attribute containing just the formatted address, as shown in *FMT.ADDR* above.
This field property can then be assigned to the dropdown description.

BSB Number

| 082| | Q |

Armidale 082-407 street_number|^|191|#|route|^|Beardy Street|#|locality|^|Armidale|#|administrative_area_level_2|^|Armidale Dumaresq Council|#|administrative_area_level_1|^|New South Wales|#|country|^|Australia|#|postal_code|^|2350|#|formatted_address|^|191 Beardy St, Armidale NSW 2350, Australia

Balmain 082-125

Bateau Bay 082-528 street_number|^|12|#|route|^|Bay Village Road|#|locality|^|Bateau Bay|#|administrative_area_level_2|^|Wyong Shire Council|#|administrative_area_level_1|^|New South Wales|#|country|^|Australia|#|postal_code|^|2261|#|formatted_address|^|Shop 77-79 Bateau Bay Square, 12 Bay Village Rd, Bateau Bay NSW 2261, Australia

Dropdown Select Statements
Refresh Result

	Dropdown Select Statement
SSELECT ABANK	

Dropdown Selection Field BNK.BSB

Dropdown Description

	Dropdown Desc
BNK.NAME	
BNK.ADDRESS.FMT	

Obtaining Geo-coordinates

Following is a method to record the current geolocation information (Latitude & Longitude) of a user to the database. When you upgrade the web site be careful not to over write the files:

- /admin/htmlMetaTaga.txt
- custom.js

It is good practice to take a backup of the existing web site before unzipping the download file, then in the new download file delete all files that you want to keep, then copy the new download over the top of the existing file.

Create a hidden input field on your first logon page and add a custom attribute in DesignBais (e.g. coor='1'). Put this field in its own section and make the section hidden in forms designer.

Download the following two script files to your DBNET root (e.g. c:\dbnet\)

- gears_init.js
- geo.js

Edit your /admin/htmlMetaTags.txt file and append these two lines:

- `<script src="gears_init.js"></script>`
- `<script src="geo.js"></script>`

Add this to the end of your custom.js file as follows (custom.js is already in your c:\dbnet folder):

```
$(document).ready(function(){
   if(geo_position_js.init()){
      geo_position_js.getCurrentPosition(success_callback,error_callback,{enableHighAccuracy:true});
   }else{
      //do nothing
   }

   function success_callback(p)  {
      var coordinates = p.coords.latitude +  "|" + p.coords.longitude;
     $("input[coor='1']").val(coordinates);
   }
});
```

When the user hits the Logon button, you'll receive the coordinates in the value of your hidden field.

DesignBais allows Javascript to be included via a number of methods:

1. The custom.js file in the website. (Be careful when upgrading not to over write the file).
2. Global General parameters. Script may be with a src reference or full script. Loaded for all systems.
3. System Parameters "General" button. Loaded for forms are loaded in the account.
4. Forms Designer "Additional script to include for Ajax".
5. On a field or button there is a "Field After Script", vs(event); will be added to call the database "Process after" e.g. in the example below "fred" is displayed after clicking the button.

Button Definition

Button Name	B.BUTTON16
Field Text	Submit
Section	Main
Row	60 Column
Process After	
Display Class	I2Button
Return to Field	
Z-Index Order	
Close Modal Window	☐ Field
Custom Attributes	
Field After Script	

alert("fred");

6. DBI.G.AJXCMD has an "addScript" function
7. The COMMON variable DBAJAXCMD can be utilised to add a function call
8. HTML and scripts can be embedded in a DesignBais field on a form.
9. Google geo-location is available in the Responsive Design tool to search for addresses.
10. Load a DesignBais field with a Custom HTML Attribute and use javacript/jquery to find the field and set its value. The next event will then pass back the value in the field.

Insert Button: [Ctrl+B]

Button Groups

The Button Groups feature is used to place a number of buttons close to each other and prevent wrapping.

To demonstrate this feature consider the snip below where there is one row with three columns and each column contains a button element. In order to place the buttons close together the developer may try to change the column width from the 4,4,4 as shown here.

The developer may think that setting column widths of 1,1,10 will place two of the buttons close together. In practice a width of 2 is usually the minimum to avoid the button element overflowing the column boundary. The result of setting 1,1,10 is shown here. Note that the buttons in the columns with width of 1 overflow into the adjacent column.
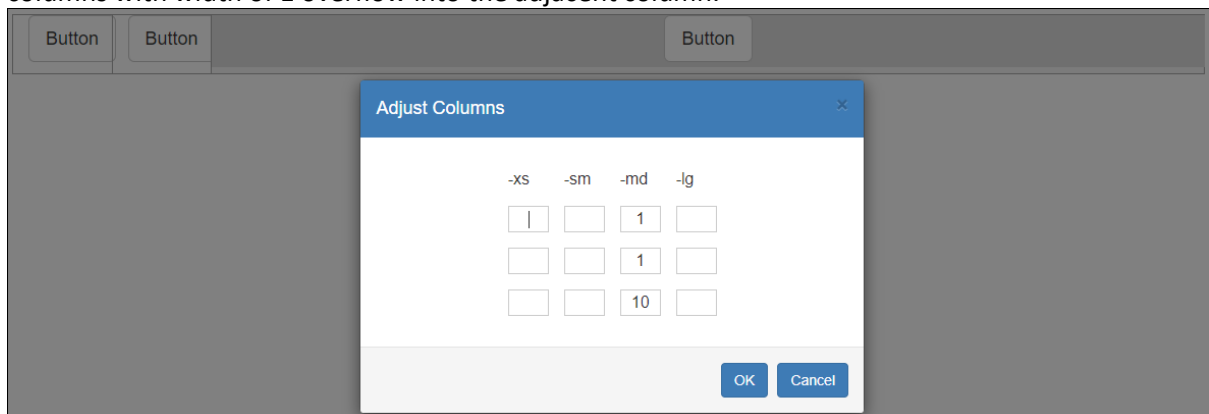
A better method is to use the *Grouping* option.

Select an empty column and press CTRL+B three times to create three buttons.
Right click > Column > Grouping > Right
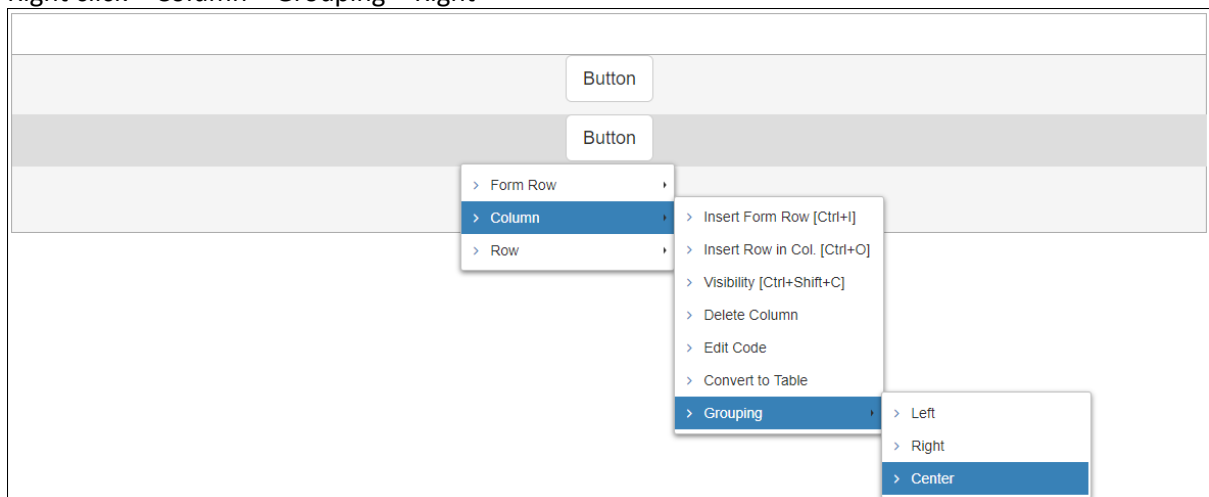
Three buttons will be centered in their column with a smaller margin between them.
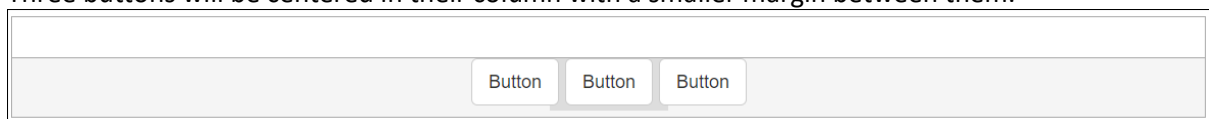
Note that too many buttons will exceed the screen width so developers should review/test their Button Groups in small screens (e.g. iPhone 4). A maximum of 3 buttons is recommended.

Only buttons can be grouped.

Inserting Buttons in the Same Row

Each form row can only contain one form element. If Grouping is not used then in order to have two buttons side by side you must create two columns in the form row.

You cannot create a column just for the button if there are multiple form rows within a single row. The effect is shown below.

There are multiple fields, each in its own form row, in the second row (*Title* through *No of Plays*).

If a column is inserted using the *Insert Column* option then the effect is as shown below.

Empty space is created below the new column.

So in order to achieve the intended result the new column needs to be inserted into a row that has only one form element in it. In this example the *Date Last Played* field occupies its own row so Ctrl-J will insert a column into the row.

Date Last Played	dd/mm/yyyy	
👤 Stats	⌃ Clear	💾 Save	▶ Play

By adding a form row and a button the required result can be achieved.

Date Last Played	dd/mm/yyyy	Button
👤 Stats	⌃ Clear	💾 Save	▶ Play

Insert Select: [Ctrl+S]

Use this element to create a list of valid entries to be selected from a dropdown list.

Attributes

Label Text   Enter the label for this select field. For example *Last Name*.

Id   The id for this form element. Must be unique within this form. Use the *auto* icon to generate the id.

Visible Options   The number of options to be displayed in and below the select field.

Event   Set to on if an event is to be associated with this field. Event type is *Change*.

Disabled   Set on to disable this field.

Multiple Selection
                  Set on to allow multiple selections from the list.

Autofocus   Set on to place focus to this field on entry to the form.

Mandatory   Check the box to make this field mandatory.

Max. Width   The maximum width of thie field in pixels.

Text Size   The font size, either *Large, Medium*, or *Small*.

Help text	Enter text, that will appear under this field, to assist a user to complete the entry of this field.
Tip text	Text causes a question mark character to be displayed after the *Label*. Clicking this question mark character at run time displays the *Tip text* in a help bubble overlay.
Alignment	The justification for this field, either *text-right* or *text-left* within the column.
Label span *Input span*	The proportion of this column, from a total of 12 units, that is to be assigned to the label with the balance being assigned to the actual input field. If the *Label span* is assigned a value of *X* then the *Input span* is set to *12 − X*. The above defines the default scenario. It is possible to assign values to the Label and Input spans that sum to a number greater then 12. In this case the Label will be placed above the input field. Similarly the values assigned can sum to a number less than 12 which leads to Label and Input occupying a smaller proportion of the available column width.

Test Area

After changing the above properties click the *Submit (simulation)* button. This Text Area input field allows the developer to test input in this form input field.

Use the Select Properties Page 2 *Options* fields to define the options for this *Select* element.

Options

value	in *value* enter the code to be used for the option.
text	in *text* enter the description to be associated with the option.

Test Area

After changing the above properties click the Submit (simulation) button. This Text Area input field allows the developer to test input in this form input field.

For the *Select* element the *Test Area* is also used to display the option to be actioned by the buttons at the base of the form.

Set Default Option   Highlight the option that is to be set as the default then click the *Set Default Option* button.

Remove Option   Highlight the option that is to be removed then click the *Remove Option* button.

To maintain existing selection options use the *Test Area*. Select an option from the dropdown and either set it as the default, or remove it.

Insert Radio: [Ctrl+R]

You must assign an *id*. Use the *Auto* icon to use the DesignBais default which is the *Label Text*. If required the text is followed by a number to ensure it is unique.

Click *In-line* to create the checkbox elements on the same line horizontally. Otherwise the elements are placed vertically.

Use the *Radio Options* fields to add each radio button to the form.

Using the same *id* displays the message *Option value is already used.*

Use the *Remove Option* button to remove a radio button from the group. In the *Test Area* tick the option(s) to be removed then click the *Remove Option* button.

Use the *Unselect* button to unselect elements in the Test Area.

Clicking an element in the Test Area displays the *id* of the element.

Insert Checkbox: [Ctrl+H]

You must assign an *id*. Use the *Auto* icon to use the DesignBais default which is the *Label Text*. If required the text is followed by a number to ensure it is unique.

Click *In-line* to create the checkbox elements on the same line horizontally. Otherwise the elements are placed vertically.

Use the *Add checkbox to Group* fields to add each checkbox to the group.

Using the same *id* displays the message:

Use the *Remove Option* button to remove a checkbox from the group. In the *Test* Area tick the option(s) to be removed then click the *Remove Option* button.

Use the *Unselect* button to unselect elements in the Test Area.

Clicking an element in the Test Area displays the *id* of the element.

Insert Text Area: [Ctrl+E]

Properties

Label	Enter the label for this input field. For example *Last Name*.
Id	The id for this form element. Must be unique within this form.
Placeholder	The text that appears in the field prior to entry of data by the user. Use this to indicate the nature of the required input.
Event	Set to on if an event is to be associated with this field. Event type is either *Click* or *Change*.
Disabled	Set on to disable this field.
Readonly	Set on to make this field *Read Only*.
Autocomplete	Set on to enable *Autocomplete* behaviour for this field.
Autofocus	Set on to place focus to this field on entry to the form.
Help text	Enter text, that will appear under this field, to assist a user to complete the entry of this field.
Max. Width	The maximum width of thie field in pixels.

*Height **	The height of this field in pixels. The * indicates that entry of a height greater then 100 will only take effect when viewing the field within the form itself, rather than in this Test Area.
Input Size	The font size, either *Large, Medium*, or *Small*.
Tip text	Text causes a question mark character to be displayed after the *Label*. Clicking this question mark character at run time displays the *Tip text* in a help bubble overlay.
Alignment	The justification for this field, either *text-right* or *text-left* within the column.
Label span *Input span*	The proportion of this column, from a total of 12 units, that is to be assigned to the label with the balance being assigned to the actual input field. If the *Label span* is assigned a value of *X* then the *Input span* is set to *12 – X*. The above defines the default scenario. It is possible to assign values to the Label and Input spans that sum to a number greater then 12. In this case the Label will be placed above the input field. Similarly the values assigned can sum to a number less than 12 which leads to Label and Input occupying a smaller proportion of the available column width.

Test Area

After changing the above properties click the *Submit (simulation)* button. This TextArea input field allows the developer to test input in this form input field.

Validation

Mandatory   Check the box to make this field mandatory.

Input Mask

Allow All	This field allows all characters to be input.
Allow [a-z]	Only lower case alphabetic characters can be input.
Allow [A-Z]	Only upper case alphabetic characters can be input.
Allow Numbers	Only number characters can be input.
Allow Chars.	Specify those characters that are permitted to be entered.
Min. Chars	Specify the minumum number of characters that must be entered.
Max. Chars	Specify the maximum number of characters that can be entered.

Use the above options *Allow [A-Z]* or *Allow [a-z]* for field properties that have an Input conversion setting of *Uppercase* or *Lowercase*.

Test Area

After changing the above properties click the *Submit (simulation)* button. This TextArea input field allows the developer to test input in this form input field.

Buttons

Submit (simulation)

        Apply changes to the Test Area only. Changes are not updated to the form.

OK   Exit the properties form and update all changes.

Cancel   Exit the properties form without updating any changes.

Insert File Upload: [Ctrl+F]

Use Preview to test the upload function. The Browse Folders icon will be active.
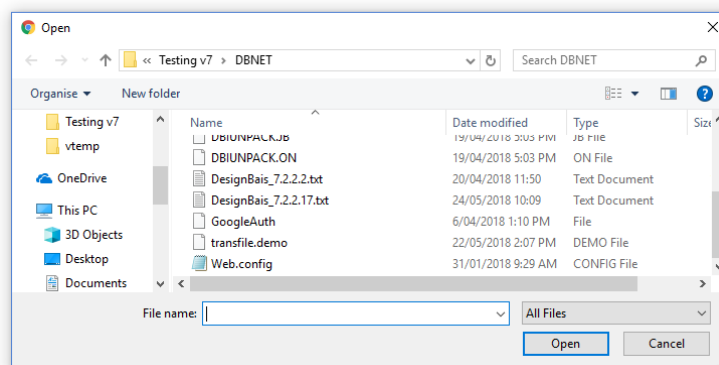
There are parameters in the setup node of *yourwebsite*/admin/db.config which control which file extensions are permitted, the maximum size of an upload file and the number of uploaded files. These can be set as required.

<RDallowedUploadExtensions>***.gif.jpg.jpeg.png.tif.tiff.pdf***</RDallowedUploadExtensions>
<RDallowedUploadSizeMB>***100***</RDallowedUploadSizeMB>
<RDallowedUploadFileCount>***10***</RDallowedUploadFileCount>

****IMPORTANT EXPLANATION ABOUT RD UPLOAD PARAMETERS*****

The web site's admin/db.config can have the following configuration parameters:

<RDallowedUploadExtensions>
<RDallowedUploadSizeMB>
<RDallowedUploadFileCount>

NOTE THAT THESE PARAMETERS APPLY ONLY IN DESIGN TIME.
This means, in RD designer, when you add a new FILE-UPLOAD element to the form, that element's attributes are set based on those parameters found in db.config. Once the page is published, the parameters found in db.config do not have any effect. Changing those parameters in db.config won't make any difference in run time. You must use rdSetAttribute() to set these parameters as needed at run time!

Note the following about the upload control:
- There is no EVENT that can be assigned to an UPLOAD control.
- The upload control itself DOES NOT send a validation.
- The upload control's state is not maintained from page to page.

It is therefore recommended that the developer have a button to control the upload function. For example the form could have a "*Continue*" button.

If a page has one or more *Upload* controls then it must also have a button with the *Event* and *Page Validation* attributes checked. In the *Form Data Link* option link the button to the subroutine that is to control the upload function.

This is the process:
- The user clicks the *Continue* button
- The back end subroutine checks the validity of each element. If there is a validation error then the error(s) are displayed and focus remains on the upload form.
- If all fields on the page validate then the subroutine performs the upload.
- Once all uploads are complete, all field values, together with the uploaded file names are sent to the back end.

File upload example:
Assume the id of "the file upload component in the RD Form" is "bcup" and the id of the "actual file upload HTML element is "bcupFileDUpload". (The DesignBais Web Component adds the suffix "FileDUpload").

The xml string looks like this:
<input id="bcupFileDUpload" designer="true" type="file" class="form-control hidden" data-show-preview="false" allowedext=".gif.jpg.jpeg.png.tif.tiff.pdf" allowedsize="100" allowedcount="10" multiple="" filemandatory="false" targetFolder="uploads/boris">

The first three attributes are self-explanatory. They are the defaults set in the db.config. The developer can use rdsetAttribute on the upload element to change the default values for allowedext, allowedsize, allowedcount and also to set the targetFolder.

Note:
- If no changes are made using *rdsetAttribute* the system will use the db.config defaults

- The target folder is a relative path and is "*uploads*" by default so if the targetFolder attribute is missing then the "*uploads*" folder is used
- Missing folders are NOT created so in this example the *uploads/boris* folder must be created manually on the web server

The following **Upload Checklist** summarises the requirements for the Responsive Design Upload element:

- Make sure the page has at least one button with "Event" and "Page Validation" attributes checked (submit behaviour)

- In response to the page onload event (AFTER DISPLAY), use rdSetAttribute() to set the attributes listed below:
    - *allowedext*
    - *allowedsize*
    - *allowedcount*
    - *accept*
    - *multiple*
    - *targetfolder*

- DesignBais will use default values for the above attributes if they are not set using rdSetAttribute(). These default values come from web site's db.config file as specified when the page was published. If defaults have not been set then, for example, a file extension type may not be uploaded because it is missing from the DesignBais default settings.

- When the button with "submit behaviour" is clicked, DesignBais:
    - Validates all fields on the page
    - Uploads files if validation succeeds
    - Sends an event to the application to notify that the page has been validated and all files have been uploaded.
    - If the attributes are set in the database and any uploaded file is outside the parameters then all uploaded files will be deleted.
    - Please note that a malicious user might be able to upload files outside the parameters if the attributes are derived from the db.config defaults and NOT set in the database

Examples for an upload element with ID = "myupload"

allowedext:
rdSetAttribute("myupload","element","allowedext",".gif.jpg",0);

allowedsize(MB):
rdSetAttribute("myupload","element","allowedsize","10",0);

allowedcount:
rdSetAttribute("myupload","element","allowedcount","5",0)

accept:
rdSetAttribute("myupload","element","accept",".gif, .txt",0);

multiple:
rdSetAttribute("myupload","element","multiple","multiple",0);   à set this if the allowedcount is more than 1
rdRemoveAttribute("myupload","element","multiple",0); à remove the multiple attribute if the allowedcount is 1

targetfolder:
rdSetAttribute("myupload","element","targetFolder","uploads/myfolder",0);

Output Field Display Customisation in Responsive Design

Here is a method to display an output field value within a shaded input-type field element.

Set up an RD form and amend the File Properties to use the customized theme (css) file:

Add the HTML form element to the form. In this example the element Id is "costCustom".
Edit the html code as follows:

Basic subroutine code to display the value entered in this field.
Note that the output conversion mask is read from the field property on DBIPROP so that changes are automatically reflected.

```
        CASE THIS.PARAMETER = 'CUSTOMCSS' AND SCREEN.NO = 'CUSTOMCSS';* after read
                DISP.COST = DBRECORD<MUS.COST>
                SAVE.EVENTSOURCE = EVENTSOURCE
                EVENTSOURCE = 'MUS.COST'
                GOSUB GET.FIELD.PROP
                EVENTSOURCE = SAVE.EVENTSOURCE
                IF OUTPUT.CONV # '' THEN
                   DISP.COST = OCONV(DISP.COST,OUTPUT.CONV)
                END
                GOSUB DISPLAY.CUSTOMCOST

            CASE SCREEN.NO = 'CUSTOMCSS';* validation of the field to be displayed
               BEGIN CASE
                  CASE EVENTSOURCE = 'MUS.COST'
                     DISP.COST = DBVALUE
                     DBRECORD<MUS.COST> = DBVALUE
                     GOSUB GET.FIELD.PROP
                     IF OUTPUT.CONV # '' THEN
                        DISP.COST = OCONV(DISP.COST,OUTPUT.CONV)
                     END
                     GOSUB DISPLAY.CUSTOMCOST
               END CASE

*
GET.FIELD.PROP:
*
        OUTPUT.CONV = ''
        READ FORMREC FROM F.DBIFORMS,CHANGE(SCREENROOT,'_','*') ELSE RETURN
        LOCATE EVENTSOURCE IN FORMREC<DBIF.FIELD.NAME.LIST> SETTING PPOS ELSE RETURN
        PROPID = FORMREC<DBIF.FIELD.FILENAME,PPOS>:'*':EVENTSOURCE
        READ PROPREC FROM F.DBIPROP,PROPID THEN
           OUTPUT.CONV = PROPREC<DBIP.OUTPUT.CONVERSION>
        END
        RETURN

*
DISPLAY.CUSTOMCOST:
*
        AJX.ARG = 'costCustom'
        AJX.ARG<4> = "<p class='dboutput'>":DISP.COST:"</p>"
        AJX.FUNC = 'SV'
        CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
        RETURN
```

Copy a standard RD css file and amend it by adding the dboutput class (see below):

```css
/* HEADER */
#dnav {

}

/* NAVBAR */
#dnavbar {
    background-color:#cccdce;
    background-image: none;
    border-style:none;
    -webkit-box-shadow:none;
    box-shadow:none;
}

/* NAVBAR MENU ITEMS*/
#dnavbar a, #dnavbar span, #dnavbar ul, #dnavbar li{
    background-color:#cccdce;
    background-image: none;
    color:#000;
}

    #dnavbar ul {
        background-image:none;
    }

#dnavbar a:hover , #dnavbar a:hover span{
    color:#000;
    background-color:#dddddd;
}

    #dnavbar .dropdown-menu {
        background-color:#cccdce;
        border: 1px solid #cccdce;
    }

    #dnavbar .navbar-toggle {
        color:#fff;
        background-color:#fff;
    }

/* FOOTER */
#dfooter {
    background-color:#fff;
    background-image: none;
    padding-top:20px;
    border-style:none;
    border-top-style:solid;
    border-top-color:#cccdce;
    border-top-width:30px;
    margin-top:40px;
    padding-top:20px;
    -webkit-box-shadow:none;
    box-shadow:none;
}


/* TAGS */
h1, h2, h3, h4, h5, h6 {
    color:#555;
}

p {
    font-size:16px;  /* better for iOS - prevents zoom*/
```

```css
        color:#222;
    }

    .fa {
        /*color:#555;*/
    }

    .dbtable table th {
        background-color: #cccdce;
        color: #222;
        border: 1px solid #cccdce;
    }

    .dboutput{
        border:1px solid #ccc;
        border-radius:5px;
        padding:5px;
        padding-left:10px;
        padding-right:10px;
        text-align: right;
        background-color:#f5f5f5;
    }
```

A Method for Creating an Event using HTML

There is already have the ability to link an output field on the RD Form (HTML field) to a data field in the DBIFORMS record produced by the Form Data Link.

If the output field has no event then the contents may be set from program code e.g. AFTER DISPLAY or AFTER READ or VALIDATE. If the field is not a work or read variable then the contents will stay after a normal form clear. The contents may be simple text "Fred was here" or "Hello World" if you prefer, or indeed HTML:

```
DBOTHER.RECORD(11)<1> = '<input id="testHTML" type="button" onclick="dbFire(event);"
value="Click Me" />'
```
puts a button with an event despite no event in RD.

```
DBOTHER.RECORD(11)<1> = '<img src="images/db/dbLogo.jpg" height="40" id="testHTML"
onclick="dbFire(event);" />'
```
puts an image with an event despite no event in RD.

If there is an event it will be sent to the database subroutine as a BUTTON event.

If there is an event, unfortunately at the moment because the field is inside a <div> we get DBVALUE='' in the BUTTON code logic. This returns the field value (EVENTSOURCE) as an empty string and therefore clears the field. The developer can, to get around this, in the BUTTON logic have DBVALUE = DBOTHER.RECORD(11)<1> (in the sample above).

The onclick="dbFire(event);" will be provided by the RD Page if the event option is ticked but the id will be fixed and the data cleared.

A better work around for the event clearing the data field – is to have a hidden button.

In the AFTER DISPLAY set your own html id:

DBOTHER.RECORD(11)<1> = ''

Then hide the button:

```
AJX.ARG = 'utility'
AJX.ARG<2>='element'
CALL DBI.G.AJXCMD('HD',AJX.ARG)
```

The tdx syntax allows the developer to identify which element on the form fires the click.

The hidden button on the form can be used to control many click functions.

The button logic:

```
BUTTON:
     *
     BEGIN CASE
         CASE SCREEN.NO = "MAINT" AND EVENTSOURCE="B.UTILITY"
             DBDS<-1>='DBREPORT.CELL=':DBREPORT.CELL
```

ID	Type	File	Field	Variable	Prop	Process After	Return To	Table/Group
clientCode	text	DBCLIENT	DBC.CLIENT.CODE	DBRECORD	INPUT			
name	text	DBCLIENT	DBC.CLIENT.NAME	DBRECORD	INPUT	DB.I.JL		
address	address	DBCLIENT	DBC.FULL.ADDRESS	DBRECORD	INPUT			
notes	textarea	DBCLIENT	DBC.NOTE	DBRECORD	INPUT			
testHTML	output	DBCLIENT	DBC.WORK1.WK	DBOTHER.RECORD(11)	OUTPUT			
utility	button		B.UTILITY		BUTTON	DB.I.JL		
testField	text	DBCLIENT	DBC.AMOUNT	DBOTHER.RECORD(1)	INPUT			
creditStop	select	DBCLIENT	DBC.CREDIT.STOP	DBRECORD	INPUT			
country	select	DBCLIENT	DBC.COUNTRY	DBRECORD	INPUT			
date	date	DBCLIENT	DBC.DATE.TEST	DBRECORD	INPUT			
time	time	DBCLIENT	DBC.TIME.TEST	DBRECORD	INPUT			
education	radio	DBCLIENT	DBC.RADIO.BUTTON	DBRECORD	RADIO			education
skills	checkbox	DBCLIENT	DBC.CHECKGROUP	DBRECORD	CHECK			skills
b-save	button		B.SUBMIT		BUTTON		DBC.CLIENT.CODE	
next	button		B.NEXT		BUTTON	DBCLIENT_NOTES		
b-clear	button		B.CLEAR		BUTTON		DBC.CLIENT.CODE	
b-delete	button		B.DELETE		BUTTON		DBC.CLIENT.CODE	

RD Designer:

Lorem ipsum..

Button

Run time:

RD Test   uvRHat   v7 UD   Pulse Test   A FCM 1   GLU Lic   BA.DEV   MyIP   iBais

My AFTER.DISPLAY.HEADER fred was here Text

192.168.199.194 says

THIS.EVENT=BUTTON EVENTSOURCE=B.UTILITY
DBREPORT.CELL=1.2

OK

Products   Services   Home

Client Code ❓*   Input

Name *   Input

Address   Input   📍

Notes   Input

DesignBais

No Button

Test Field   Test Field

Implementing a Slider

A slider can be included in both RD and non-RD DesignBais forms. Refer to the DesignBais Reference Manual [Slider Control](#) for a description of the method of implementing a slider in classic DesignBais.

DesignBais Responsive Design Slider form

There is a Responsive Design Slider demo page dbdemo-slider which links to the form DBDEMO*RDSLIDER in the DBINET.DEMO account. This form allows you to modify the options required for the slider and redisplay the slider which is shown here:

The form and the basic code behind it are available by clicking in column 2 of the *Responsive Design* row of the Demo Forms list which can be displayed by selecting the Demo Forms menu option on the top menu of the Development Tools form. This displays the form DBDEMO_RDDEMO. On the *Page* dropdown list select the *dbdemo-slider* option. You can then click either *Run Form* to run the form, or *Data Link* to view the Responsive Design Form Data Link settings.

The basic code used to drive the display of the carousel is contained in the DBLIB subroutine DBI.I.DEMO which is included in the DesignBais release in account DBINET.DEMO.

To create a slider in Responsive Design you simply create a Text Input element on your RD form.

In the Form Data Link function link this form element to the required database field.

Listed below is sample code to implement the slider displayed following the sample code. Note that FLD.NAME in the example below can be either the DBIPROP Field Name or the RD form element ID in the form data link.

```
    AFTER.DISPLAY:
    AJX.ARG = ''
    *
    FLD.NAME   = 'BKT.PROP.Q130L5'
    SLIDER.MIN = 5
    SLIDER.MAX = 20
    SLIDER.STEP = 0
    SLIDER.FROM = DBRECORD<BKT.PROP.Q130L5>+0
    SLIDER.VALUES = '[5,10,15,20]'
    GOSUB DISPLAY.SLIDER
    *
    FLD.NAME = 'BKT.PROP.Q23A'
    SLIDER.MIN = 1
    SLIDER.MAX = 20
    SLIDER.STEP = 0
    SLIDER.FROM = DBRECORD<BKT.PROP.Q23A>+0
    SLIDER.VALUES = '[1,2,5,10,20]'
    GOSUB DISPLAY.SLIDER
    *
    CALL DBI.G.AJXCMD('SL',AJX.ARG)

DISPLAY.SLIDER:
    LOCATE FLD.NAME IN AJX.ARG<1>,1 SETTING POS ELSE
       AJX.ARG<1,POS>  = FLD.NAME
    END
    AJX.ARG<3,POS,-1> = 'event'          ; AJX.ARG<4,POS,-1> = 'true'
    AJX.ARG<3,POS,-1> = 'min'            ; AJX.ARG<4,POS,-1> = SLIDER.MIN
    AJX.ARG<3,POS,-1> = 'max'            ; AJX.ARG<4,POS,-1> = SLIDER.MAX
    AJX.ARG<3,POS,-1> = 'step'           ; AJX.ARG<4,POS,-1> = SLIDER.STEP
    AJX.ARG<3,POS,-1> = 'from'           ; AJX.ARG<4,POS,-1> = SLIDER.FROM
    AJX.ARG<3,POS,-1> = 'type'           ; AJX.ARG<4,POS,-1> = 'single'
    AJX.ARG<3,POS,-1> = 'skin'           ; AJX.ARG<4,POS,-1> = 'big'
    AJX.ARG<3,POS,-1> = 'grid'           ; AJX.ARG<4,POS,-1> = 'true'
    AJX.ARG<3,POS,-1> = 'snap'           ; AJX.ARG<4,POS,-1> = 'true'
    AJX.ARG<3,POS,-1> = 'values'         ; AJX.ARG<4,POS,-1> = SLIDER.VALUES
    RETURN
```
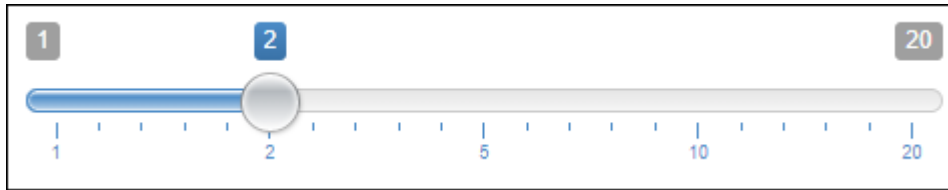
Note that the slider must only be displayed when the form is loaded. This is usually in the AFTER DISPLAY event. DesignBais will control the update of the slider in, for example, the field validation event. The slider will react automatically (from the developer's point of view) to any change in the field's *Variable to Use*, such as DBRECORD or DBOTHER.RECORD.

You can use javascript to vary the display of the slider. See DBI.G.AJXCMD for the full list of ajax functions. For example the slider labels can be amended using the *prefix* and *postfix* options.

```
AJX.ARG<3,POS,-1> = 'prefix'        ; AJX.ARG<4,POS,-1> = '$'
AJX.ARG<3,POS,-1> = 'postfix'       ; AJX.ARG<4,POS,-1> = ' M'
```
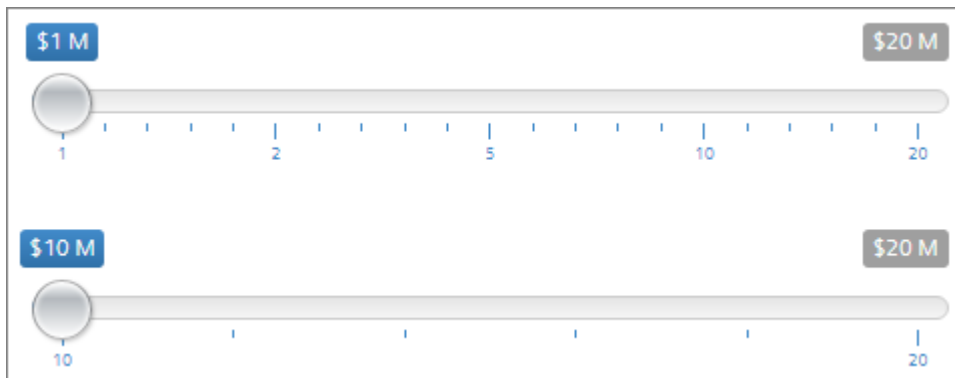
Implementing a Slider Panel

A Slider Panel can be included in both RD and non-RD DesignBais forms. Refer to the DesignBais Reference Manual [Slider Panel Control](#) for a description of the method of implementing a slider panel in classic DesignBais.

DesignBais Responsive Design Slider Panel form

There is a Responsive Design Slider Panel demo page dbdemo-slidepanel which links to the form DBDEMO*RDSLIDEPANEL in the DBINET.DEMO account. This form allows you to modify the options required for the slide panel.

To create a slider panel in Responsive Design you use a HTML output field. The slidePanel function moves the element into the panel and it disappears from the initial location. This means that the output field should be hidden up front but displayed when the function is called. Use the normal rdHide & rdShow.

In the demo form the HTML form element has an id of *slidePanel*.

The call to DBI.G.AJXCMD will set the custom attribute value Custom Attributes = 'dbslidepanel="1"' automatically if it is not present.

The basic code can be found in the dropdown list of Basic Code Segments in the *Code Editor* form DBIUSERS_D80. Select the *Code to support a Slide Panel in an RD form* in the *Select Code Segment* grid.

```
35          * Add CSS
36          AJX.DATAIN = ''
37          AJX.DATAIN<1>   = 'dbspScript1'
38          AJX.DATAIN<2>   = 'js/dbslidePanel-min.js'
39          CALL DBI.G.AJXCMD('AS',AJX.DATAIN)
40          * Add Javascript
41          AJX.DATAIN = ''
42          AJX.DATAIN<1>   = 'dbspCSS1'
43          AJX.DATAIN<2>   = 'js/css/dbslidePanel.css'
44          CALL DBI.G.AJXCMD('addCSS',AJX.DATAIN)
45          * Slide Panel options can be in any order
46          DBWORK<DB.TEMP.PANEL.WK>    = 'content'        ; DBWORK<DB.TEMP.PANEL.VALUE.WK>    = 'Quote Details'
47          DBWORK<DB.TEMP.PANEL.WK,2>  = 'titleWidth'     ; DBWORK<DB.TEMP.PANEL.VALUE.WK,2>  = '30px'
48          DBWORK<DB.TEMP.PANEL.WK,3>  = 'titleFontSize'  ; DBWORK<DB.TEMP.PANEL.VALUE.WK,3>  = '14px'
49          DBWORK<DB.TEMP.PANEL.WK,4>  = 'titleFontFamily'; DBWORK<DB.TEMP.PANEL.VALUE.WK,4>  = 'sans-serif'
50          DBWORK<DB.TEMP.PANEL.WK,5>  = 'width'          ; DBWORK<DB.TEMP.PANEL.VALUE.WK,5>  = '240px'
51          DBWORK<DB.TEMP.PANEL.WK,6>  = 'height'         ; DBWORK<DB.TEMP.PANEL.VALUE.WK,6>  = '200px'
52          DBWORK<DB.TEMP.PANEL.WK,7>  = 'rightOffset'    ; DBWORK<DB.TEMP.PANEL.VALUE.WK,7>  = '50px'
53          DBWORK<DB.TEMP.PANEL.WK,8>  = 'topOffset'      ; DBWORK<DB.TEMP.PANEL.VALUE.WK,8>  = '100px'
54          DBWORK<DB.TEMP.PANEL.WK,9>  = 'backgroundColor'; DBWORK<DB.TEMP.PANEL.VALUE.WK,9>  = '#5bc0de';*#4477cc'
55          DBWORK<DB.TEMP.PANEL.WK,10> = 'borderColor'    ; DBWORK<DB.TEMP.PANEL.VALUE.WK,10> = '#4477cc';*#5E25D8'
56          DBWORK<DB.TEMP.PANEL.WK,11> = 'color'          ; DBWORK<DB.TEMP.PANEL.VALUE.WK,11> = 'white'
57          DBWORK<DB.TEMP.PANEL.WK,12> = 'sticky'         ; DBWORK<DB.TEMP.PANEL.VALUE.WK,12> = 'false'
```

The slider panel options are shown in the example basic code above.
The *topOffset* option is used as follows:

sticky=true	topOffset determines the stick position of the panel
sticky=false or absent	topOffset determines the offset position of the panel with respect to its container

You can use the parameters "open", "close" and "toggle" to control a slide panel. Here is an example of the basic code. The slider panel is displayed using the work field DEM.WORK4.WK. Refer to the Demo Form DBDEMO_RDSLIDEPANEL and the basic code in DBI.I.DEMO.

```
CASE EVENTSOURCE = "B.OPEN"
    AJX.ARG = 'DEM.WORK4.WK'
    AJX.ARG<2> = 'open'
    CALL DBI.G.AJXCMD('SP',AJX.ARG)

CASE EVENTSOURCE = "B.CLOSE"
    AJX.ARG = 'DEM.WORK4.WK'
    AJX.ARG<2> = 'close'
    CALL DBI.G.AJXCMD('SP',AJX.ARG)

CASE EVENTSOURCE = "B.TOGGLE"
    AJX.ARG = 'DEM.WORK4.WK'
    AJX.ARG<2> = 'toggle'
    CALL DBI.G.AJXCMD('SP',AJX.ARG)
```

Implementing a Carousel

A Carousel can be included in both RD and non-RD DesignBais forms.

Refer to the DesignBais Reference Manual for a description of the steps required when Implementing a Carousel in a Classic (non-RD) DesignBais form.

DesignBais Responsive Design Carousel Demo form

There is a Responsive Design Carousel demo page dbdemo-carousel which links to the form DBDEMO*RDCAROUSEL, which is shown below, in the DBINET.DEMO account.

The form and the basic code behind it are available by clicking in column 2 of the *Responsive Design* row of the Demo Forms list which can be displayed by selecting the Demo Forms menu option on the top menu of the Development Tools form. This displays the form DBDEMO_RDDEMO. On the *Page* dropdown list select the *dbdemo-carousel* option. You can then click either *Run Form* to run the form, or *Data Link* to view the Responsive Design Form Data Link settings.

The basic code used to drive the display of the carousel is contained in the DBLIB subroutine DBI.I.DEMO which is included in the DesignBais release in account DBINET.DEMO.

While you do not strictly need to use basic code to create the carousel, the practical way for DesignBais developers is to set up the carousel is via a basic subroutine. In order to make the setup as easy as possible there is a template form accessible from the *Subroutine* button on the Code Editor form DBIUSERS_D80. This is referenced below.

First you will need to create or modify your RD form. To add a carousel element to your RD form:
- Create an HTML segment (CTRL+K)
- Open the properties for this element (CTRL+P) and set a meaningful Id for the element – this is useful in the data link process to tell you what the element is.
- Edit the HTML segment (CTRL+Q), click Source, remove the existing code (<p>Lorem ipsum...</p>) and replace it with <div id = "*mycar4*"></div> where *mycar4* is the name by which you will reference the carousel. This div will hold the carousel when the page is displayed.
- Save and publish the page – for this example the page is called *mycar4* in work folder *test*

Then, assuming that you have used folder *test* and page *mycar4*:
- Open the *Responsive Design Form Data Link* form *DBIFORMS_RD10* and select "*test-mycar4*" from the Page Name dropdown.
- Enter the file to which you want to link the form. E.g. *DBCLIENT*
- Enter a new Form Name. This is the name of the DesignBais "Classic" mode form that resides in DBIFORMS. E.g. *MYCAR4*
- The Process after Display field will be empty so we need to create a new subroutine to hold the code that controls the carousel setup and display.
- Click the Process after Display hyperlink.

The *Create New Subroutine & Insert Basic Code Segment* form *DBIUSERS_D85* will display.

Enter the required fields shown in yellow:
- File Name of the basic subroutine library – E.g. DBLIB
- Item Name of the subroutine – E.g. MY.I.CAR
- From the Select Code Segment dropdown select *Code to support a Carousel in an RD form* (to view the skeleton code click on the header text Select Code Segment)

Click the *Define Parameters for Carousel* hyperlink at the top of the form. This opens the *Set Parameters for Basic Code for Carousel* form *DBIUSERS_D86* mentioned above.

Enter the parameters to display and control the carousel that are to be incorporated into the form subroutine:

- Form File name – *DBCLIENT*
- Equates From File – this will default to the value from DBIFILES for the file
- Form Name (for the DBIFORMS form record) – *MYCAR4*
- RD Element Id (this must match the element id used in the RD page) – *myCar4*
- Res Image Path (the website path to the location of the images) – *images/carousel* for example. You need to determine where images will reside. It is good practice to place images for RD forms in the *res/websitename/images* folder.
- Image Names (as many image names as are required for the carousel)
- Aspect Ratio (will depend on the size of the images used)
- Button Width (the size of the scroll buttons on the carousel)
- Field Properties (field names from the Form File which will be used to pass the carousel attributes)

The images to be displayed in the carousel must be present in the folder that you specify in the *Res Image Path* field.

Click the *Accept* button.

Click the *Code Editor* Button to create the subroutine.

The code will present in the DesignBais Code Editor. The text in red highlights the code that is created from the details entered on the Create New Subroutine form:

```
SUBROUTINE MY.I.CAR
* Description:
*
* This program may not be reproduced, disclosed, used or sold in any
* form or by any means without written authorisation from :-
*
* Written : garb 04/02/2021
* Amended :
*
$INCLUDE DBI DBI.COMMON
$INCLUDE DBEQU E.DBCLIENT
      *
      FILENAME = FIELD(SCREENROOT,"_",1)
      SCREEN.NO = FIELD(SCREENROOT,"_",2)
      SCREEN.NO = CHANGE(SCREEN.NO,".DESIGNER","")
      *
      THIS.EVENT = PROCESS.EVENT
      THIS.PARAMETER = PROCESS.PARAMETER
      EVENTSOURCE = CHANGE(PROCESS.EVENTSOURCE,".DESIGNER","")
*
* Event Table
*
      BEGIN CASE
         CASE THIS.EVENT = "AFTER DISPLAY"
            GOSUB AFTER.DISPLAY
      END CASE
      RETURN
*
* Code to support a Carousel in an RD form
* "MYCAR4" is the designated DesignBais form name (from DBIFORMS Filename*Formname)
* "myCar4" is the designated RD form element id in which the carousel is located
* DBC.WORK1.WK & DBC.WORK2.WK are the designated work fields for replacement in this skeleton
```

```
*
AFTER.DISPLAY:
      BEGIN CASE
         CASE SCREEN.NO = "MYCAR4";* this is the name of the form record in DBIFORMS to which the RD page is linked
            GOSUB SET.CAROUSEL
      END CASE
      RETURN
*
SET.CAROUSEL:
*
      * Carousel options can be in any order
      DBWORK<DBC.WORK1.WK> = ""
      DBWORK<DBC.WORK2.WK> = ""
      DBWORK<DBC.WORK1.WK,-1> = "aspectRatio"; DBWORK<DBC.WORK2.WK,-1> = "3.8" ;* bigger number reduces the height of the image
      DBWORK<DBC.WORK1.WK,-1> = "imgPath"    ; DBWORK<DBC.WORK2.WK,-1> = "res/video/images"
      DBWORK<DBC.WORK1.WK,-1> = 'images'     ; DBWORK<DBC.WORK2.WK,-1> = '["smallCar.jpg","trolley.png","harvest.jpg"]'
      DBWORK<DBC.WORK1.WK,-1> = "buttonWidth"; DBWORK<DBC.WORK2.WK,-1> = "30px"
      *
      AJX.ARG = "myCar4"
      AJX.ARG<2> = "script"    ;* needed to add the dbCarousel script to the HTML page
      * Also send the options for this carousel
      JMAX = DCOUNT(DBWORK<DBC.WORK1.WK>,VM)
      FOR J = 1 TO JMAX
         IF DBWORK<DBC.WORK1.WK,J> # "" AND DBWORK<DBC.WORK2.WK,J> # "" THEN
            AJX.ARG<3,1,-1> = DBWORK<DBC.WORK1.WK,J>
            AJX.ARG<4,1,-1> = DBWORK<DBC.WORK2.WK,J>
         END
      NEXT J
      CALL DBI.G.AJXCMD("CAR",AJX.ARG)
      RETURN
```

Compile the code using the *Compile* option from the Code Editor *File* menu.

In the *Responsive Design Form Data Link* form *DBIFORMS_RD10* the name of your new subroutine will display in the *Process after Display* field.

Save the record by clicking Submit.

Recall the page from the Page name dropdown and click the Test button.

The carousel will display.

The full list of options that can be used to tailor the carousel is shown below.

Option	Allowed Values	Description
imgPath	String	Relative path to the folder containing the carousel images. Default: "images"
images	String array	Comma delimited image names. Default: ["1-min.png", "2-min.png", "3-min.png"]
showButtons	Boolean	Show left/right navigation buttons Default: true
showDots	Boolean	Show navigation dots Default: true
skin	String	Various preset skins. Possible values are: blue, red, green, orange, gray, dark and white. Default: "blue"
buttonType	String	"arrowhead" or "awesome". Default: "arrowhead". **Note: "awesome" can be used if FontAwesome is loaded on the page.**
buttonWidth	String	Width of navigation buttons in px. Default: "40px"
buttonColor	A color value (e.g. blue)	Fore color of navigation buttons. Default: #eff
buttonBackgroundColor	A color value (e.g. blue)	Background color of navigation buttons. Default: #2af
buttonBorderColor	A color value (e.g. blue)	Border color of navigation buttons. Default: transparent
buttonActiveOpacity	A decimal number (0 to 1)	Button opacity when not active Default:0.7
buttonOpacity	A decimal number (0 to 1)	Button opacity when active (mouse over) Default:1.0
buttonOffset	String	Offset of navigation buttons from the left and right edges. Negative values will make the buttons appear outside by making the images narrower. Note that images are not distorted but the aspect ratio is lost.
dotWidth	String	Width of dots. Default: "10px"
dotsTopOffset	String	Offset of navigation dots from the bottom edge. Default: "-20px"

Option	Allowed Values	Description
		Negative values will make the dots appear outside by making the images shorter. Note that images are not distorted but the aspect ratio is lost.
dotColor	A color value (e.g. blue)	Color of naviagtion dots. Default: "#2af"
dotActiveOpacity	A decimal number (0 to 1)	Dot active opacity (i.e. the corresponding image is being displayed) Default:1.0
dotOpacity	A decimal number (0 to 1)	Dot opacity when not active Default:0.5
animationSpeed	Number	Transition time from one image to the next in millisconds. Default: 700
loopPeriod	Number	Wait time between image transitions. Default:2000 **Set this number to 0 to stop animations.**
callback	String	Callback function name. The function is called on tap, click, swipe etc. events. Default: "" (i.e. no callback).
height	String	Carousel height in px Do not specify if responsive
width	String	Carousel width in px Do not specify if responsive
leftOffset	String	Offset of the carousel from the left of its relatively positioned container or document body. Default: "0px"
topOffset	String	Offset of the carousel from the top of its relatively positioned container or document body Default: "0px"
aspectRatio	Number	The width/height ratio of the carousel when the width is specified in pixels OR when the carousel is placed in a responsive grid column without specifying a width.

Implementing a Day Picker

Refer to the Demo Form *dbdemo-daypicker* in the DBINET.DEMO account.

RD DesignBais form

The procedure to follow to add a day picker element to your RD form is:
- Create a new page
- CTRL+K to create an HTML segment
- CTRL+Q to edit that segment. Replace <p>Lorem…</p> with <div id="delivery"></div> and save

The form element will appear like this:

The code behind this HTML element looks like this:

```
i  1▾ <div class="dbcol col-md-12 dbcolhilite">
   2▾     <div class="dbform form-horizontal">
   3▾         <div class="dbformgroup form-group">
   4▾             <div dbtype="segment" class="html-text" id="d370216">
   5                 <div id="delivery"></div>
   6             </div>
   7         </div>
   8     </div>
   9 </div>
  10
```

- From the File menu select Properties and add a hidden field to the form. In this example the id used is *hiddenId*.
- Publish the page.
- In *Form Data Link* option link your page to a form. The demo form setup is shown below. The work field DEM.WORK1.WK is used to pass the date selected from the web page to the database.

| | | Recent | Copy Form | Report | Clear | Review | Submit |

Page Name: dbdemo-daypicker — dbdemo-daypicker Demo Day Picker ⌄

Page Description: Demo Day Picker

Filename: DBDEMO — DBDEMO Demo Client File ⌄

Form Name: DAYPICKER

CSS Filename default
Header undefined
Footer undefined

Full Description: Day Picker demo form

Preserve Common ☐   Sub Form ☐

Button to Action when Enter is pressed   -- Select -- ⌄

Process Before Display: [ ]   Parameter [ ]
Process after Display: DBI.I.DEMO   Parameter [ ]
Modal Close via X Process: [ ]

Default Key Value: [ ]
Form Read Group: [ ]   Form Read Variable -- Not Used -- ⌄
Form File Name: --No File Selected-- ⌄
Form Read Type: No Lock ⌄

Search for ID: [ ]

ID	Type	Unlink	File	Field	Variable	Prop	Event	Process After
d370216	output	✖						
hiddenId	hidden	✖	DBDEMO	DEM.WORK1.WK	DBWORK	INPUT		DBI.I.DEMO

In your basic subroutine you will need code similar to the following in the AFTER DISPLAY event processing:

```
CASE SCREEN.NO = "DAYPICKER"
        * "delivery" is the ID of the <div> container in the RD page to be used for the day picker
          HEAD.TEXT = "Please select the day on which you wish to visit"
          AJX.FUNC = "DP"                            ;* dbDayPicker
          AJX.ARG = "delivery"
          AJX.ARG<3,1,-1> = 'startDate'           ; AJX.ARG<4,1,-1> = DATE()
          AJX.ARG<3,1,-1> = 'endDate'             ; AJX.ARG<4,1,-1> = DATE()+90
          AJX.ARG<3,1,-1> = 'selectedDate'        ; AJX.ARG<4,1,-1> = '""'
          AJX.ARG<3,1,-1> = 'pickerTitle'         ; AJX.ARG<4,1,-1> = HEAD.TEXT
          AJX.ARG<3,1,-1> = 'displayDate'         ; AJX.ARG<4,1,-1> = '27-Oct-2020'
          AJX.ARG<3,1,-1> = 'disabledArray'       ; AJX.ARG<4,1,-1> = '["14-Nov-2020","12-Nov-
2020"]'
          AJX.ARG<3,1,-1> = 'customAttributeArray' ; AJX.ARG<4,1,-1> = '[3001, 9010, 2111, 3101,
1100, 2010, 3001, 9111, 2111, 1111, 1100, 2110, 2010, 2110]'
        * hiddenId is the ID of the hidden field in the RD page to pass data back to the database
          AJX.ARG<3,1,-1> = 'onselect'            ; AJX.ARG<4,1,-1> = 'function ($el, d) {}'
          AJX.ARG<3,1,-1> = 'onanimate'           ; AJX.ARG<4,1,-1> = 'function (b) { }'
          AJX.ARG<3,1,-1> = 'hiddenID'            ; AJX.ARG<4,1,-1> = 'hiddenId'
          *
          CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

In the VALIDATION event you will need code to process the selected date:

```
CASE EVENTSOURCE="DEM.WORK1.WK" AND SCREEN.NO = "DAYPICKER"
          AJX.FUNC = "DPSH"          ;* dbDayPickerSetHTML
          AJX.ARG = "delivery"     ;* the ID of the <div> container in the RD page to be used for the
day picker
          AJX.ARG<3,1,-1> = '<div style="color:green;">You have selected date: ':DBVALUE:'</div>'
          CALL DBI.G.AJXCMD(AJX.FUNC,AJX.ARG)
```

The day picker displays allowing the user to select a day:

Please select the day on which you wish to visit

‹   ›

Sunday	Monday	Tuesday	Wednesday
Jul 16th	Jul 17th	Jul 18th	Jul 19th

You have selected date: 17-Jul-2023

Hit Blocker in RD Forms

Hit blocker settings are shown in the figure below.

```
<option value>-- Inherit Setting --</option>
<option value="0">0 Element not disabled. Subsequent events are sent.</option>
<option value="1">1 Element not disabled. Subsequent events are blocked.</option>
<option value="2">2 Element disabled but enabled on return. Subsequent events sent.</option>
<option value="3">3 Element disabled but enabled on return. Subsequent events blocked.</option>
<option value="4">4 Element disabled & kept disabled. Subsequent events sent.</option>
<option value="5">5 Element disabled & kept disabled. Subsequent events blocked.</option>
```

In the AFTER DISPLAY event of a form the desired hit blocker setting can be set for all form elements using the following, where 'n' is a value 0 through 7:

> JST = ';':STR(' ',3);* this is the DesignBais javascript terminator (; followed by 3 spaces)
> HBVAL = 'n'
> DBAJAXCMD<-1> = "dbhbMode=":HBVAL:JST

The hit blocking action takes effect as each input field or button element is executed.

For example if HBVAL is set to 4 then after entering a value in an input field the input field will be hidden. After clicking a button the button will be hidden.

Alternatively hit blocker can be set for selected form elements using the rdSetAttribute function.

RD.FUNC = 'SA'	*SA is the abbreviation for 'rdSetAttribute'*
RD.INP = 'save'	*the id of the form element*
RD.INP<2> = 'element'	*the element type*
RD.INP<3> = 'dbhb'	*the CSS property*
RD.INP<4> = HBOPT	*the hit blocker value to use*
CALL DBI.G.AJXCMD(RD.FUNC,RD.INP)	

Attribute	Content	Format
1	RD form element id	multi-valued
2	Type of RD Form element (see list below)	multi-sub valued
3	CSS property	multi-sub valued
4	CSS property setting or value	multi-sub valued
5	mSec delay time	multi-sub valued

Type of RD Form element
> row
> col
> formRow
> element
> label
> option
> optionlabel
> error
> tipicon
> tiptext
> help
> segment   (use for HTML output element)

Example of HTML Solution for Progress Icons

Create a HTML row form item on each page for example:
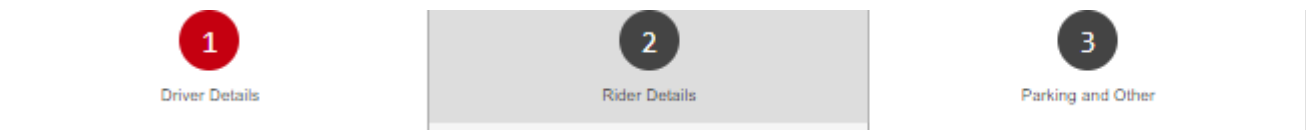
For each column set up the following HTML code. Note that if column 1 is the current form then the HTML code is as below, so *breadon* highlights the current page:

```
<div class="dbformgroup form-group dbrowcolhilite">
        <div dbtype="segment" class="html-text" id="step1" existingid="step1">
                <p class="bread breadon">1</p>
                <p style="width:100%; text-align:center;font-size:11px">Driver Details</p>
        </div>
</div>
```

In the above case column 2 is <u>not</u> the current form so the HTML code is:

```
<div class="dbformgroup form-group dbrowcolhilite">
        <div dbtype="segment" class="html-text" id="d377293" existingid="d377293">
                <p class="bread">2</p>
                <p style="width:100%; text-align:center;font-size:11px">Rider Details</p>
        </div>
</div>
```

Columns 3 and 4 are the same as column 2 when column 1 is the current page.

In the css file (e.g. famous.css) the *bread* class is defined as:

```
.bread{
   width: 40px;
   text-align: center;
   border: 1px solid #0000;
   margin-right: auto;
   margin-left: auto;
   border-radius: 50%;
   background-color: #444;
   color: #fff;
   padding: 0px;
   font-size: 20px;
   height: 40px;
   line-height: 40px;
   font-family: monospace;
}
```

Example of Table Row with Checkbox

This demonstrates a method for placing a checkbox into a table row. The final result is shown below. The requirement was to display a fixed set of risks with an associated cost and allow the user to select one or more of these items. The solution uses the Table Row feature of Responsive Design (refer to #Convert to Table).

net/dbnet.aspx?dbpage=bob-excess

Description	Amount	Select
All Other Claims	5000	☑
All Other Claims Increased	3000	☐
Submersion Excess	2000	☐
Named Cyclone	1000	☑
Age Under 25 / Inexperience	700	☑
Theft	1000	☑
Personal Effects	1500	☑
Whist Racing	900	☑
		Submit

Method:

1. Set up three RD form elements in a *single* form row – they can be all HTML outputs, but in this example the second element is a disabled input.

The Submit button allows for testing the updating of the selections on a database file.

2. Then use the Convert to Table option to create a table.

The table look like this in Designer mode:

Description	Amount	Select
Lorem ipsum..	Excess Amount	Lorem ipsum.
		Submit

3. Save and publish the page.
4. Open the Form Data Link option on the Responsive Design menu.

Responsive Design Form Data Link — Recent Copy Form Export Clear Review Submit

Page Name	bob-excess	bob-excess
Page Description		
Filename	DBCLIENT	DBCLIENT Client Test & File
Form Name	RDEXCESS	

CSS Filename default
Header undefined
Footer undefined

Full Description	Excess Table	
Preserve Common	☐	Sub Form ☐
Button to Action when Enter is pressed	-- Select -- ∨	

Process Before Display		Parameter	
Process after Display	DB.I.RG	Parameter	EXCESS
Modal Close via X Process			

Default Key Value 43
Form Read Group 1   Form Read Variable DBRECORD ∨
Form File Name DBCLIENT Client Test & File ∨
Form Read Type Lock ∨

Search for ID

ID	Type
excessDesc	table
submit	button

Table Definition   Table Name excessDesc

ID	Type	Unlink	File	Field	Variable	Prop	Event	Process After
d623384	output	✗	DBCLIENT	DBC.EXCESS.WK	DBWORK	OUTPUT		
excessAmount	text	✗	DBCLIENT	DBC.EXCESS.AMT.WK	DBWORK	INPUT		
excessSelect	output	✗	DBCLIENT	DBC.EXCESS.SELECT.WK	DBWORK	OUTPUT	click	DB.I.RG

This example uses a default key value to read a designated record. This is just for testing purposes. In your form you may be inserting the table row into an existing form with its own read and write logic.

Copyright © 2018 DesignBais | DesignBais Responsive Design Release 9.2.1.1   203

5. Click on the table row id *excessDesc*.

6. Link the three form elements to DBWORK fields. In your basic code populate the work fields as required.

The basic code to handle the excessSelect column which contains the checkbox is as follows:

The AFTER DISPLAY is the same as the AFTER READ (forms with a Default Key Value don't have an after read event) and sets up the checkbox on each row.

```
*
AFTER.DISPLAY:
*
    BEGIN CASE
      CASE SCREEN.NO = "RDEXCESS"
          DBWORK<DBC.EXCESS.WK> = DBRECORD<DBC.EXCESS>
          DBWORK<DBC.EXCESS.AMT.WK> = DBRECORD<DBC.EXCESS.AMT>
          DBWORK<DBC.EXCESS.SELECT.WK> = ''
          *
          CHKBX.ON  = '<p style="text-valign:middle;"><input type="checkbox" class="dbchkstyle-lg"
checked></p>'
          CHKBX.OFF = '<p style="text-valign:middle;"><input type="checkbox" class="dbchkstyle-lg"
></p>'
          *
          SMAX = DCOUNT(DBRECORD<DBC.EXCESS>,VM)
          FOR II=1 TO SMAX
             IF DBRECORD<DBC.EXCESS.SELECT,II> = 'Y' THEN
                DBWORK<DBC.EXCESS.SELECT.WK,II> = CHKBX.ON
             END ELSE
                DBWORK<DBC.EXCESS.SELECT.WK,II> = CHKBX.OFF
             END
          NEXT II
```

The click on the checkbox triggers a button event – look for the " checked" setting.

```
*
BUTTON:
*
    BEGIN CASE
      CASE SCREEN.NO = "RDEXCESS" AND EVENTSOURCE="DBC.EXCESS.SELECT.WK"
          CHKBX.ON  = '<p style="text-valign:middle;"><input type="checkbox" class="dbchkstyle-lg"
checked></p>'
          CHKBX.OFF = '<p style="text-valign:middle;"><input type="checkbox" class="dbchkstyle-lg"
></p>'
          * Just toggle values
          IF DBWORK<DBC.EXCESS.SELECT.WK,DBMVCOUNT> = CHKBX.ON THEN
             DBWORK<DBC.EXCESS.SELECT.WK,DBMVCOUNT> = CHKBX.OFF
          END ELSE
             DBWORK<DBC.EXCESS.SELECT.WK,DBMVCOUNT> = CHKBX.ON
          END
```

Update the record based on the " checked" setting. The PROCESS.STACK in the example below is for testing only, to allow the save record to be re-displayed.

```
*
BEFORE.WRITE:
*
     BEGIN CASE
         CASE SCREEN.NO = 'RDEXCESS'
             DBRECORD<DBC.EXCESS.SELECT> = DBWORK<DBC.EXCESS.SELECT.WK>
             SMAX = DCOUNT(DBWORK<DBC.EXCESS.SELECT.WK>,VM)
             FOR II=1 TO SMAX
                 WVAL = DBWORK<DBC.EXCESS.SELECT.WK,II>
                 IF INDEX(WVAL,' checked>',1) THEN
                     DBRECORD<DBC.EXCESS.SELECT,II> = 'Y'
                 END ELSE
                     DBRECORD<DBC.EXCESS.SELECT,II> = 'N'
                 END
             NEXT II
             PROCESS.STACK = "DBCLIENT_RDEXCESS"
```

Changing the color of buttons in Responsive Design

The RD primary color looks like this: #36B2AD

Create 5 more colors by darkening the primary color. In this case, the new colors will be from #36B2AD down to #00524D as shown below;

A good way to make darker color is as follows:

> #XXYYZZ: one shade darker is (X-1)X(Y-1)Y(Z-1)Z
> e.g. #36B2AD becomes #26A29D

Copy and paste the following block into your css file and change the colors with your new set of colors. Of course, this works only if you used the "Primary Button" type in designing your page.

```css
.btn-primary{
color: #fff;
background-color: #36B2AD;
border-color: #16928D;
}

.btn-primary.active, .btn-primary:active, .open>.dropdown-toggle.btn-primary {
  color: #fff;
  background-color: #16928D;
  border-color: #06827D;
}

.btn-primary.focus, .btn-primary:focus {
  color: #fff;
  background-color: #06827D;
  border-color: #00726D;
}


.btn-primary:hover {
  color: #fff;
  background-color: #00726D;
  border-color: #00625D;
}

.btn-primary.active.focus, .btn-primary.active:focus, .btn-primary.active:hover, .btn-primary:active.focus, .btn-primary:active:focus, .btn-primary:active:hover, .open>.dropdown-toggle.btn-primary.focus, .open>.dropdown-toggle.btn-primary:focus, .open>.dropdown-toggle.btn-primary:hover {
  color: #fff;
  background-color: #00625D;
  border-color: #00524D;
}
```

Make sure to refresh the browser so that the new css is active. Navigate to the url and press F12 to bring up Chrome developer tools. Press CTRL+F5 to refresh the page (stylesheets are hard to refresh). Press F12 to close the developer tools.