BSP Model – Data Binding Made Easy

SDN Community Contribution

(This is not an official SAP document.)

Disclaimer & Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.
Applies To:
ABAP stack of SAP NetWeaver starting from R6.20.

Summary

With more and more focus being put on BSP development, developers are beginning to explore more and more of the capabilities and features available within the systems. Described within this document are some of the main advantages of using model data binding within a BSP application.

By: Sergio Ferrari and Craig Cmehil

Company: ERPTech (Sergio Ferrari) / Hella KGaA Hueck & Co (Craig Cmehil)

Date: 20 July 2005

With the cooperation of Mr. Thomas Jung (Kimball International) regarding the chapter “Dynamic Model Binding”
Applies To: ........................................................................................................................................... 2
Summary .................................................................................................................................................. 2
Introduction ............................................................................................................................................... 4
Let’s introduce a practical example (basic setup) .................................................................................... 6
  Domains .................................................................................................................................................. 8
  Data Elements ....................................................................................................................................... 10
  Structures ............................................................................................................................................. 12
  BSP Application .................................................................................................................................. 13
  Model ................................................................................................................................................... 14
  Method: SEARCH .................................................................................................................................. 16
  Method: UPDATE_MESSAGE ................................................................................................................ 16
Data dictionary integration ...................................................................................................................... 25
  Data types - validations and visualization ............................................................................................. 25
  Enumeration validation .......................................................................................................................... 26
  conversion at domain level .................................................................................................................... 28
  Labels and tooltips ............................................................................................................................... 29
  Example ............................................................................................................................................... 29
How to react to formal errors? .................................................................................................................. 30
How to react to application errors in the same way as formal ones? ......................................................... 30
How to verify required fields? .................................................................................................................. 30
1 Dynamic Model Binding ...................................................................................................................... 31
  Model: YCL_BSP_MODEL_DYN_BIND ................................................................................................. 34
  Controller: ZMVC_BSP_DYNAMIC ...................................................................................................... 34
  View: mvc_dynamic.htm ...................................................................................................................... 36
Conclusions ............................................................................................................................................... 37
Author Bio .............................................................................................................................................. 38
Introduction

Craig¹ having finally completed his second series² on “Exploring BSP Development³, Sergio⁴ decided it was time to approach Craig and discuss the topic of “Data Binding”, something that Craig had left out of his two series. Sergio, being a fan of the topic and wanting to see it more visible proposed the two of them collaborate on weblog or article concerning the topic.

It was quite a funny meeting actually, it occurred right after Craig gave a presentation⁵ in Walldorf, during the “SDN Meets Labs⁶. Sergio caught up to Craig and told him “I think you left something out of your tutorial, it’s very good but you left something out that needs to be in”, Craig of course wondering “Uh?!” and thus began the discussion.

SAP has a done a wonderful job documenting “Data Binding”, the standard documentation⁷ located on http://help.sap.com explains the concept, theory and use behind the topic. However, we feel that it is not visible enough nor used enough and therefore what you will find contained within this document is our attempt to make the subject a bit more friendly and certainly more visible.

Participating in SDN since the beginning we have the feeling that this technique is not commonly used and in fact performing a search for the term binding up to now only 17 messages are found:

   Searching Binding: SDN 9,297 messaggi 1,606 Topics - Results: 17 Search Terms: binding

A few weblogs contain information about “Data Binding” and the special aspects of the technique but none really simplify the life of a beginner. Even after analyzing the delivered BSP applications contained in the system we can find just 2 views using the data binding:

1 Craig Cmehil - https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/com.sap.sdn.businesscard.SDNBusinessCard?u=rXr63bYDR6oXx0YFZhCX3w%3D%3D

© 2005 SAP AG  The SAP Developer Network: http://sdn.sap.com
• CRM_BSP_FRAME/tree.htm
• CRM_BSP_FRAME/tree.htm_m

Hardly what one would expect for such a great technique.
Let’s introduce a practical example (basic setup)

As usual we need a sample in order to explain the technique. So let's go ahead and get our example underway. To keep it simple we'll go ahead and play around with the whole “Flights” data.

First of all verify via SE16 if you have any records table SCARR. If not create some new entries via SE16 via the following steps:

Note that it is not important to define the CURRCODE.
To get started please go ahead and login to your test system and start transaction SE80. Select “Package” from the menu and type in a new package name. The reason for doing this is to keep all of your pieces together so you can easily see them and work with them.

Our package name is “ZTUT_MODELBINDING”, but you can use whatever you like.

The example will now consist of several individual items that in the end will work together in harmony.

- DOMAINS
- DATA ELEMENTS
- STRUCTURES
- CLASSES
- BSP Application
Domains

Domains are part of the “Dictionary Objects”, simply right click there and “Create” - “Domain”.

- **ZSDN_DOMA_ENUM**

```plaintext
<table>
<thead>
<tr>
<th>Domain</th>
<th>ZSDN_DOMA_ENUM</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Text</td>
<td>Domain values</td>
<td></td>
</tr>
<tr>
<td>Formatting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data type</td>
<td>CHAR</td>
<td></td>
</tr>
<tr>
<td>No. characters</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Decimal places</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Output characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output length</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Convers. routine</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Single vals

- **Fix.Val.**
  - AA: First
  - BB: Second
  - CC: Third

Intervals

- **Lower limit**
  - M: $M
  - S: $S
- **Upper limit**
  - M: $M
  - S: $S
```

© 2005 SAP AG

The SAP Developer Network: [http://sdn.sap.com](http://sdn.sap.com)
Data binding in BSP Applications

- ZSDN_DOMA_CONV

```
<table>
<thead>
<tr>
<th>Domain</th>
<th>ZSDN_DOMA_CONV</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Text</td>
<td>Date</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Definition</th>
<th>Value range</th>
</tr>
</thead>
</table>

**Formatting**

- Data type: DATS
  - Date field (YYYYMMDD) stored as 8 characters
- No. characters: 8
- Decimal places: 0

**Output characteristics**

- Output length: 10
- Convers. routine: PDATE

- Sign
- Lowercase
```
Data binding in BSP Applications

Data Elements

Data Elements are part of the “Dictionary Objects”, simply right click there and “Create” - “Data Element”.

- **ZSDN_DOMA_ENUM**

---

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data Type</th>
<th>Further Characteristics</th>
<th>Field Label</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Elementary Type</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Domain</strong></td>
<td>ZSDN_DOMAENUM</td>
<td>Data Type CHAR Character String</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Length 2 Decimal Places</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data Type</th>
<th>Further Characteristics</th>
<th>Field Label</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data element</strong></td>
<td>ZSDN_DOMA_ENUM</td>
<td>Active</td>
<td></td>
</tr>
<tr>
<td><strong>Short Text</strong></td>
<td>Domain values</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Short</th>
<th>Length</th>
<th>Field Label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>Enum S</td>
</tr>
<tr>
<td>Medium</td>
<td>15</td>
<td>Enumerations</td>
</tr>
<tr>
<td>Long</td>
<td>20</td>
<td>Enumerations Long</td>
</tr>
<tr>
<td>Heading</td>
<td>15</td>
<td>Enumerations Hdr</td>
</tr>
</tbody>
</table>
Data binding in BSP Applications

- ZSDN_DOMA_CONV

<table>
<thead>
<tr>
<th>Data element</th>
<th>ZSDN_DOMA_CONV</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Text</td>
<td>Conversion</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data Type</th>
<th>Further Characteristics</th>
<th>Field Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elementary Type</td>
<td>ZSDN_DOMA_CONV</td>
<td>Domain</td>
<td>Data Type</td>
</tr>
<tr>
<td>Length</td>
<td>8</td>
<td>Decimal Places</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data element</th>
<th>ZSDN_DOMA_CONV</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Text</td>
<td>Conversion</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data Type</th>
<th>Further Characteristics</th>
<th>Field Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>10</td>
<td>Field Label</td>
<td>Conv.</td>
</tr>
<tr>
<td>Medium</td>
<td>15</td>
<td>Conversion</td>
<td></td>
</tr>
<tr>
<td>Long</td>
<td>20</td>
<td>Conversion Long</td>
<td></td>
</tr>
<tr>
<td>Heading</td>
<td>14</td>
<td>ConversionHdr</td>
<td></td>
</tr>
</tbody>
</table>
Structures

Structures are part of the “Dictionary Objects”, simply right click there and “Create” - “Structure”.

- `ZSDN_MESSAGE_BAR`

- `ZSDN_MODEL_BINDING`
BSP Application

The BSP Application is of course the main component which brings everything together under one roof so to speak.

In our example, the BSP application must be set statefull, so do not forget to flag it.

Now that the application is there and in place, we will need to go back one step and build our MODEL class for the entire application. In terms of tutorials it is always difficult to determine which to build first so in this case we have decided to build our application, leave it empty and then build our MODEL class, then return to the application components.
Model

The MODEL is a class we will create based on the class CL_BSP_MODEL, this will act as the connection between our views and our logic.
Inside of our MODEL, we will then create some ATTRIBUTES to handle our data.

- PARAMETERS
- RESULTS
- MESSAGE

Then we create our METHODS.

Once our METHODS are in place we of course need add the coding for each method.
Method: SEARCH

METHOD search.

* SELECT * FROM sflight INTO CORRESPONDING FIELDS OF TABLE me->results
  WHERE carrid = me->parameters-carrid.
  AND fldate = me->parameters-fldate
  AND seatsmax = me->parameters-seatsmax.

ENDMETHOD.

Method: UPDATE_MESSAGE

METHOD update_message.

* CLEAR: me->message.

DATA: o_message TYPE REF TO cl_bsp_messages.
DATA: l_messages_severity_int TYPE i.

* me->errors->get_message( EXPORTING index = 1
  IMPORTING message = me->message-text
  condition = me->message-connectedcontrol
  severity = l_messages_severity_int).

CHECK me->message-text IS NOT INITIAL.

CASE l_messages_severity_int.
  WHEN cl_bsp_messages=>co_severity_fatal_error.
    me->message-type = 'STOP'.
  WHEN cl_bsp_messages=>co_severity_error.
    me->message-type = 'ERROR'.
  WHEN cl_bsp_messages=>co_severity_info.
    me->message-type = 'OK'.
  WHEN cl_bsp_messages=>co_severity_warning.
    me->message-type = 'WARNING'.
  WHEN cl_bsp_messages=>co_severity_success.
    me->message-type = 'OK'.
  "<-- con NONE non esce il messaggio
ENDCASE.

ENDMETHOD.

Method: RESET_RESULT

METHOD reset_result.

* CLEAR: me->results.

ENDMETHOD.
Returning to our BSP application itself, it is now possible to create the VIEWS and CONTROLLER CLASSES necessary to complete our example.

**View: MAINWITH.HTM**

This of course is connected to a CONTROLLER CLASS called ZMVC_BINDING_WITH and has the following content in the PAGE LAYOUT.
Data binding in BSP Applications

PAGE LAYOUT

```html
<!--page language="abap" -->
<%@extension name="bsp" prefix="bsp" %>
<%@extension name="htmlb" prefix="htmlb" %>
<%@extension name="phtmlb" prefix="phtmlb" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<htmlb:content design = "DESIGN2003"
    themeRoot = "sap_standard">
    <htmlb:document>
        <htmlb:documentHead title="SDN - Data binding">
            <script language=JavaScript>history.forward();</script>
        </htmlb:documentHead>
        <htmlb:documentBody marginBottom = "0"
            marginLeft   = "0"
            marginRight  = "0"
            marginTop    = "0">
            <htmlb:form id     = "f"
                method = "POST">
                <htmlb:tray id                = "tray"
                    width             = "100%"
                    hasMargin         = "false"
                    title             = "With Data Binding"
                    hasContentPadding = "true">
                    <phtmlb:formLayout labelAlignment         = "LEFT"
                        design                 = "TRANSPARENT"
                        verticalLineSeparation = "TRUE"
                        fieldToLabelFactor     = "2.5"
                        customizationKey       = "SDN">
                        <phtmlb:formLayoutColumnHeader text="Labels" />
                        <phtmlb:formLayoutInputField id       = "CarrId"
                            value    = "//msdn/parameters.fldate"
                            required = "true" />
                        <phtmlb:formLayoutColumnHeader text="Enumerations" />
                        <phtmlb:formLayoutItem idOfItem = "PDomaEnumDDLBCarrid"
                            label    = "CARRID (Value Table)">
                            <htmlb:dropdownListBox id         = "DomaEnumDDLBCarrid"
                                selection  = "//msdn/parameters.carrid"
                                helpValues = "//msdn/parameters.carrid" />
                        </phtmlb:formLayoutItem>
                        <phtmlb:formLayoutItem idOfItem = "PDomaEnumDDLB"
                            label    = "EnumerationDomain">
                            <htmlb:dropdownListBox id         = "DomaEnumDDLB"
                                selection  = "//msdn/parameters.domaenum"
                                helpValues = "//msdn/parameters.domaenum" />
                        </phtmlb:formLayoutItem>
                        <phtmlb:formLayoutInputField id    = "Seatsmax"
                            value = "//msdn/parameters.seatsmax" />
                        <phtmlb:formLayoutInputField id    = "domaconv"
                            value = "//msdn/parameters.domaconv" />
                    </phtmlb:formLayout>
                    <htmlb:button id      = "Search"
                        onClick = "Search"
                        text    = "Search" />
                </htmlb:tray>
            </htmlb:form>
        </htmlb:documentBody>
    </htmlb:document>
</htmlb:content>
```

© 2005 SAP AG
The SAP Developer Network: [http://sdn.sap.com](http://sdn.sap.com)
Followed by having one PAGE ATTRIBUTE.

- MSDN

<table>
<thead>
<tr>
<th>Attribute</th>
<th>TypingMethod</th>
<th>Associated Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>msdn</td>
<td>TYPE REF TO</td>
<td>EMVC_M_SDN</td>
</tr>
</tbody>
</table>

With the VIEW now completed, the CONTROLLER CLASS needs to be created and modified to meet our needs.
This can be done by first creating a CONTROLLER.

Then simply double clicking on the CONTROLLER CLASS name you typed in, the system will prompt to create the class if it is not found.
Within this new CONTROLLER CLASS we need to add a single attribute:

- **M_SDN**

| M_SDN | Instan... Priv... | Type Re... ZMVC_M_SDN | Model Binding - MODEL |

Then the following METHODS need to be redefined:

- **DO_REQUEST**

  ```plaintext
  METHOD do_request .
  
  * Init models
  IF me->m_sdn IS NOT BOUND.
   me->m_sdn := create_model( class_name = 'zmvc_m_sdn' model_id = 'msdnid' ).
  ENDIF.
  *
  dispatch_input( ).
  *
  * if any of the controllers has requested a navigation, do not try to display, but leave current processing
  IF is_navigation_requested( ) IS NOT INITIAL.
   RETURN.
  ENDIF.
  
  DATA: o_view   TYPE REF TO if_bsp_page.
  o_view = create_view( view_name = 'main_with_data_binding.view' ).
  o_view->set_attribute( name = 'msdn' value = me->m_sdn ).
  
  call_view( o_view ).
  
  ENDMETHOD.
  ```
• **DO_HANDLE_EVENT**

  METHOD do_handle_event .
  *  
  me->m_sdn->update_message( ).
  IF      me->m_sdn->message-type = 'STOP'
  OR     me->m_sdn->message-type = 'ERROR'.
  me->m_sdn->reset_result( ).
  RETURN.
  ENDIF.
  *
  DATA: o_page_context TYPE REF TO if_bsp_page_context.
  o_page_context = me->get_page_context( ).
  cl_htmlb_manager=>dispatch_event_ex( request = request
  page_context =
  o_page_context
  event_handler = me ).
  *
  * In case the Search does not work, try to uncomment the following
  * statement.
  * IF htmlb_event->server_event = 'search'.
  *  me->m_sdn->search( ).
  *  ENDIF.
  *
  ENDMETHOD.
Then a new method called SEARCH needs to be created.

Define the EVENT_OBJECT parameter:

And fill the source with:

```plaintext
METHOD search .
  me->m_sdn->search( ).
ENDMETHOD.
```

With all of the pieces to the BSP application now in place, a simply test of the CONTROLLER mainwith.do will result in the application loading.
How to receive user input from the HTTP request

In the SEARCH method of the controller class the selected CARRID (e.g. LH for Lufthansa) is automatically available in the model attribute MSDN->PARAMETERS-CARRID.

Note that we are making use of one approach described in the weblog “BSP Programming: Event Handling in Composite Elements” (https://weblogs.sdn.sap.com/pub/wlg/663).

Please note that without Data Binding it will be necessary retrieve from the HTTP Request the user inputs.
**Data dictionary integration**

“Data Binding” provides nice features, which leads one to sometimes think of the “Elves” lurking around the halls of the SAP Labs.

**Data types - validations and visualization**

In the first picture of our example:

As you can see the fields are automatically formatted depending on the data type, max capacity is aligned to right, date fields are provided with the “Help” icons and of course a calendar that pops up.

Error handling is also taken care of, if a user enters invalid data for any of the types the MODEL will report the error by highlighting the incorrect field. Later on we will go into more detail on the error management.

Can you imagine what will happen when typing 29.02.2005 in the first field (Date *) and pressing the Search button? If not have a look to the following picture:

![Image of Data Binding example](image-url)
Enumeration validation

The following htmlb extension:

```html
<htmlb:dropDownListBox id="DomaEnumDDLb"
    selection="/msdn/parameters.domaenum"
    helpValues="/msdn/parameters.domaenum" />
```

knows Data Dictionary very well. It will generate the list of values from the Domain definition of the binded attribute.

Both fixed values and check tables are managed obviously taking into consideration the login language for text tables.

You can see the list of airlines that comes directly from the table SCARR defined as Value table of the domain S_CARR_ID.
Note that in case of fixed values if you specify intervals you will get the lower limit key in the selection field.

Have a look in the example what is generated for the Enumeration Domain:

Unfortunately the limits declared at domain levels are not managed by other htmlb objects. It would be nice to have the same check performed against input fields.
conversion at domain level

Every SAP R/3 user knows that a material code can be written omitting leading zeros. If you want to provide the same functionality Data Binding do it. It calls the Conversion Exit defined at domain level.

Let's have a look to the conversion field of our sample: we type 25-12-2004 (please note the separator)
Try also with 25/12/2004 (please note the separator).

Please note that the conversion exit associated at domain level calls the Function Module: CONVERSION_EXIT_PDATE_INPUT. In this case it will not accept dates in the future.

Try with 25/12/2005.

We think that conversion exits can play an important role in developing BSP Web Applications.

**Labels and tooltips**

You may have already noticed that labels and column headers are filled automatically from the data element texts. As well as the tooltips.

*Note: it is important to maintain the DDIC descriptions in the same language used when logging on to the BSP application. Remember that usually the language depends on how your browser is set.*

**Example**
How to react to formal errors?

The MODEL class fills in its ERRORS attribute (that is PROTECTED) when it recognizes a formal error.

In the DO_HANLDE_EVENT we suggest to invoke a method (we called it UPDATE_MESSAGE) that copies the ERRORS attribute into a MESSAGE attribute (PUBLIC).

Therefore stopping the event processing if there are any error in your models.

Have a look at what takes place in the example.

How to react to application errors in the same way as formal ones?

In case of errors, a field is highlighted (red borders).

If you want to highlight a field that has no formal error but maybe application errors you will have to insert into the ERROR attribute of your model an error message (TYPE) and build the right value for the CONDITION field (it’s the name of the field to be highlighted)

How to verify required fields?

Here we discovered an unresolved issue. HTMLB tags usually provide the attribute required but the Data binding do not take it into consideration.

So you should verify them for example via a method of your model called in the DO_HANLDE_EVENT. If a required field is initial you should generate a new entry in the ERROR attribute
1 Dynamic Model Binding

We have already seen how powerful data binding can be. Besides all the benefits already put forward for using it, data binding really shines when it comes to creating dynamic UI elements.

For an example of the power of the dynamic model binding, let us examine a situation that would be very difficult to reproduce in classic ABAP Dynpro. We will start with a structure that represents a reduced number of fields in a database table. We want to expose each one of these fields as individual input fields with their own labels.

Of course we could manually design the UI for our structure, creating each individual element by hand. This could become time consuming depending upon the size of the structure at question. Also every time we add or remove fields from the structure, we have to return to the user interface and adjust it as well.

Would it not be much simpler if we could just supply the data object for the structure to the user interface and let it dynamically build all the necessary input fields, with Meta data pulled from the structure and automatic field input retrieval? Well that is exactly what model binding makes possible.

We will start this example by creating a structure that is a sub set of the fields in the table SFLIGHT. For now we will throw out MANDT and all the fields that break down first and business class. Our structure leaves us with about ¾ of the original fields.
We will keep everything from this point forward as dynamic as possible so that if we want to extend our user interface, all we have to do is add or remove fields from this YBSP_SFLIGHT_LITE structure. Our model class will have a public attribute of type of the structure we just created. It will also have the logic to select a single record from the database table SFLIGHT into the corresponding fields of this attribute.

Inside of our view we are ready to start our dynamic element creation. The first thing we will need to do is retrieve a listing of the fields in our structure using the ABAP RunTime Type Services or RTTS.

```abap
data: descriptor type ref to CL_ABAP_STRUCTDESCR.
  descriptor ?= CL_ABAP_STRUCTDESCR=>describe_by_data(
    model->isflight ).

data: flddescr type DDFIELDS.
  flddescr = descriptor->GET_DDIC_FIELD_LIST( ).
```

Now we are going to be able to loop through our field listing and create a label and input field for each entry. What we would like to do is just build our binding string into a variable and give that variable to the BSP elements. However when working with BSP elements within pages or views, they do not expose separate attributes for the bound and unbound values. Therefore if we send a dynamic binding string into BSP element attribute as a variable, it will incorrectly interpret that. The element will assume that we are taking the value directly from the variable instead of trying to read it as a binding string.
One might also think that completing the binding string dynamically like in the following example would be possible as well.

```html
<htmlb:label
   for="/model.isflight.<%= <wa_field>-fieldname %>">
</htmlb:label>
```

Unfortunately the BSP runtime also incorrect identifies this example as a direct value assignment as well.

Luckily there is a solution that works. If you create the BSP element directly via the ABAP class, separate attributes are exposed for bound and unbound values. For instance in the `<htmlb:label>` the implementing class, CL_HTMLB_LABEL, has two attributes – `for` and `_for`. The attribute that will expect a binding string always beings with the underscore.

So now within our view we will generate the BSP elements directly via code much like when creating a composite BSP element. We can then render the BSP element into a string and output that string within our view.

```abap
field-symbols: <wa_field> like line of flddescr.
data: label type ref to cl_htmlb_label.
data: input type ref to CL_HTMLB_INPUTFIELD.
data: binding_string type string,
   label_string type string,
   input_string type string.

"Loop through each field in the structure Definition
loop at flddescr assigning <Wa_field>.
clear: label, input.
concatenate '//model/isflight.' <wa_field>-FIELDNAME
   into binding_string.
create object label.
label-_for = binding_string.
label_string = label->IF_BSP_BEE-RENDER_TO_STRING(
   page_context ).
create object input.
input-_value = binding_string.
input_string = input->IF_BSP_BEE-RENDER_TO_STRING(
   page_context ).
```

We will use the flexibility of the `<phtmlb:matrix>` to support our dynamic UI.
With only a handful of lines of code, we have generated our 9 fields from our simplified SFLIGHT structure. However the same number of lines of code could have just as easily created 90 input fields and their labels.

![Figure 1.2: Dynamic Model Binding Output](image)

Here is the whole code:

**Model:** YCL_BSP_MODEL_DYN_BIND

| Attribute: ISFLIGHT Instance Public Type SFLIGHT. |
| method init. (Instance Public) |
| select single * from sflight into corresponding fields of isflight. |
| endmethod. |

**Controller:** ZMVC_BSP_DYNAMIC

| Attribute: MODEL Instance Private Type Ref To YCL_BSP_MODEL_DYN_BIND. |
| method do_init. |
| if model is initial. |
model ::= create_model( model_id = 'MD'
        class_name = 'YCL_BSP_MODEL_DYN_BIND' ).
endif.
model->init( ).
endmethod.

method do_request.
    data: view type ref to if_bsp_page.

    * if input is available, dispatch this input to subcomponent.
    * this call is only necessary for toplevel controllers.
    * ( if this is not a toplevel controller or no input is present,
    *   this call returns without any action)
    dispatch_input( ).

    * if any of the controllers has requested a navigation,
    * do not try to display, but leave current processing
    if is_navigationRequested( ) is not initial.
        return.
    endif.

    view = create_view( view_name = 'mvc_dynamic.htm' ).
    view->set_attribute( name = 'MODEL'
            value = me->model ).
    call_view( view ).
endmethod.
Data binding in BSP Applications

View: mvc_dynamic.htm

Attribute: MODEL Type Ref To YCL_BSP_MODEL_DYN_BIND.

Layout:

<%@page language="abap" %>
<%@extension name="htmlb" prefix="htmlb" %>
<%@extension name="phtmlb" prefix="phtmlb" %>
<htmlb:content design="design2003" >
<htmlb:page title=" " >
<htmlb:form>
<phtmlb:matrix width="100%" >
<%
data: descriptor type ref to CL_ABAP_STRUCTDESCR.
descriptor ?= CL_ABAP_STRUCTDESCR->describe_by_data( model->isflight ).
data: flddescr type DDFIELDS.
flddescr = descriptor->GET_DDIC_FIELD_LIST( ).
field-symbols: <wa_field> like line of flddescr.
data: label type ref to cl_htmlb_label.
data: input type ref to CL_HTMLB_INPUTFIELD.
data: binding_string type string,
label_string type string,
input_string type string.
"Loop through each field in the structure Definition
loop at flddescr assigning <Wa_field>.
clear label.
clear input.
concatenate '//model/isflight.'
<wa_field>-FIELDNAME
into binding_string.
create object label.
label->_for = binding_string.
label_string = label->IF_BSP_BEE~RENDER_TO_STRING( page_context ).
create object input.
input->_value = binding_string.
input_string = input->IF_BSP_BEE~RENDER_TO_STRING( page_context ).
%
  <phtmlb:matrixCell row    = "+1"
vAlign = "TOP" />
  <%= label_string %>
  <phtmlb:matrixCell col    = "+1"
vAlign = "TOP" />
  <%= input_string %>
%
endloop.
%
</phtmlb:matrix>
</htmlb:form>
</htmlb:page>
</htmlb:content>
Conclusions

Hopefully after all that typing, your example worked as well as ours and our extreme hope is that we have shed some light onto “Data Binding” and it's benefits!
Author Bio

is a Senior NetWeaver Consultant and the R&D manager of www.ERPTech.it.

is a web developer for Hella KGaA Hueck & Co.