Calling BAPIs from the SAP Web Application Server

Thomas G. Schuessler

Thomas G. Schuessler is the founder of ARAsoft, a company offering products, consulting, custom development, and training to customers worldwide, specializing in integration between SAP and non-SAP components and applications. Thomas is the author of SAP's BIT525 and BIT526 classes. Prior to founding ARAsoft in 1993, he worked with SAP AG and SAP America for seven years.

(complete bio appears on page 104)

The SAP Web Application Server is a full-blown development environment for building state-of-the-art web applications. In release 6.10, you can build your applications in ABAP, and starting with 6.20 you can choose between ABAP and Java. Some of the web applications that you want to build will be unrelated to the SAP application components, but in many cases you will need some access to SAP application functionality. This is usually accomplished by calling BAPIs in R/3, APO, BW, CRM, and other SAP application components.

This article will show you, step-by-step, how to find the relevant BAPI information in an SAP target system[1] (using a 4.6C R/3 system as an example), generate the client ABAP code for the BAPI calls in the SAP Web Application Server, and incorporate the generated code into a BSP[2] application written in ABAP using SAP Web Application Server release 6.10. A complete (albeit simple) sample application that displays customer information from R/3 in a BSP page will be discussed.

I presume that you are already somewhat familiar with ABAP, BSP applications, and BAPIs so that we can concentrate on how to use BAPIs in BSP applications.[3]

The BAPI Browser

The SAP Web Application Server contains a BAPI Browser that can

[1] That is, the system in which you want to invoke BAPIs from the Web Application Server.

[2] BSP stands for Business Server Pages. A BSP page is an HTML page with embedded ABAP scripting.

[3] For introductions to BSP applications and BAPIs, refer to the appropriate articles in this publication (see page 105 for a listing) or the SAP online documentation.

New to BAPIs?

If you are not that familiar with the BAPIs yet, here is a brief introduction:

BAPIs (Business Application Programming Interfaces) are the official interfaces to SAP for synchronous connectivity. BAPIs are defined in the SAP Business Object Repository (BOR) as methods of object types. The *Customer* object type, for instance, has methods (BAPIs) like *CreateFromData*, *GetDetail1*, and *GetSalesAreas*. So there is a certain amount of object-orientation. But the BAPIs are not implemented as methods of ABAP *classes*, since the object-oriented features of ABAP were added after release 3.1 in which we encountered the first few BAPIs. Instead, BAPIs are implemented as simple RFC-enabled Function Modules (RFMs). When you want to invoke a BAPI, you simply call the function module (the RFM) that implements it. It is still convenient to think about the BAPIs in an object-oriented fashion at design time, though. And the best way to find BAPIs, the BAPI Explorer*, represents the BAPIs that way, too.

* More about the BAPI Explorer below.

generate ABAP source code for BAPIs and other RFC-enabled Function Modules (RFMs). This BAPI Browser has a limited scope[4] (viz., list BAPIs and generate ABAP code). Hence it is advisable to first look up the BAPI metadata in the target system (R/3 in our case) so that is where we will start.

Exploring the BAPIs in the Target System

The most convenient way to find out about the BAPIs of a target system is to use the BAPI Explorer (transaction code "BAPI"). **Figure 1** shows the *Customer.GetDetail1* BAPI with its parameters.

Here are the steps you should execute in order to be ready for calling a BAPI from the SAP Web Application Server:

- Check that the object type to which the BAPI belongs is not obsolete.[5] BAPIs from obsolete object types should normally not be used. Usually, when SAP declares an object type obsolete, a new object type is created. If you cannot find the new object type by browsing the object type list in the BAPI Explorer, check the documentation of the BAPIs of the obsolete object type for hints.
- Check that the object type is not delegated.[6] If it is, find out to which object type, and continue your investigation with that one.
- Read the documentation of the object type[7]

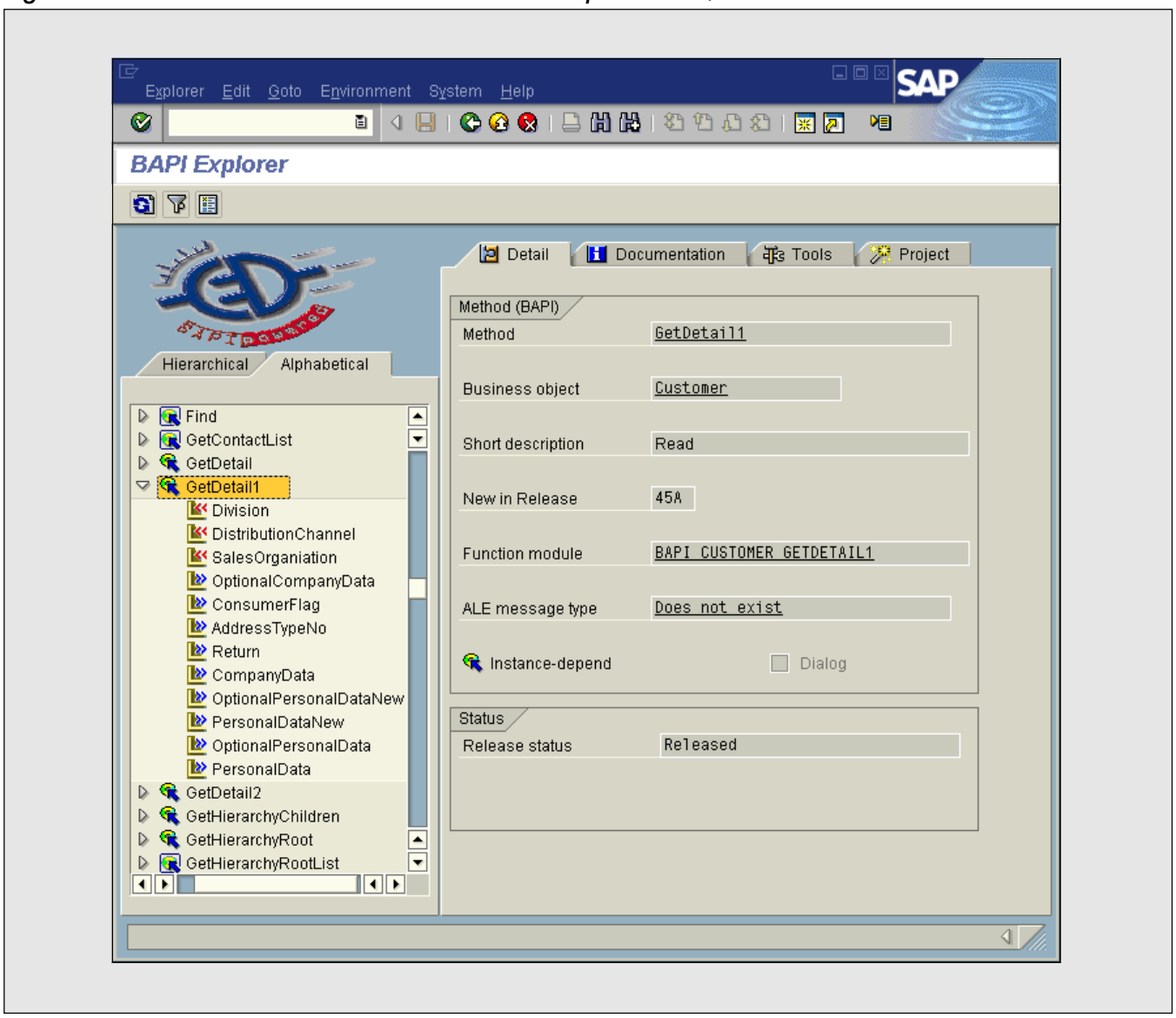
[4] Compared to the BAPI Explorer (transaction code "BAPI").

[5] The current version of the BAPI Explorer does not show whether an object type is obsolete, but you can use an external metadata server to access this information. In most cases, though, all BAPIs of an obsolete object type will be flagged as obsolete as well (see below). But there are exceptions to this rule. For your convenience, these are the obsolete object types in 4.6C: *Creditor, Debtor, InvestmentProgram, Qualification, RepManConfirmation*. (*Qualification* is obsolete, but its one BAPI is not. This is clearly an oversight by SAP.)

[6] If you do not know the concept of delegation in the BOR, you should read my BOR article in the January/February 2001 issue of this publication.

[7] Not every object type comes with documentation.

Figure 1 *The BAPI Explorer in R/3*



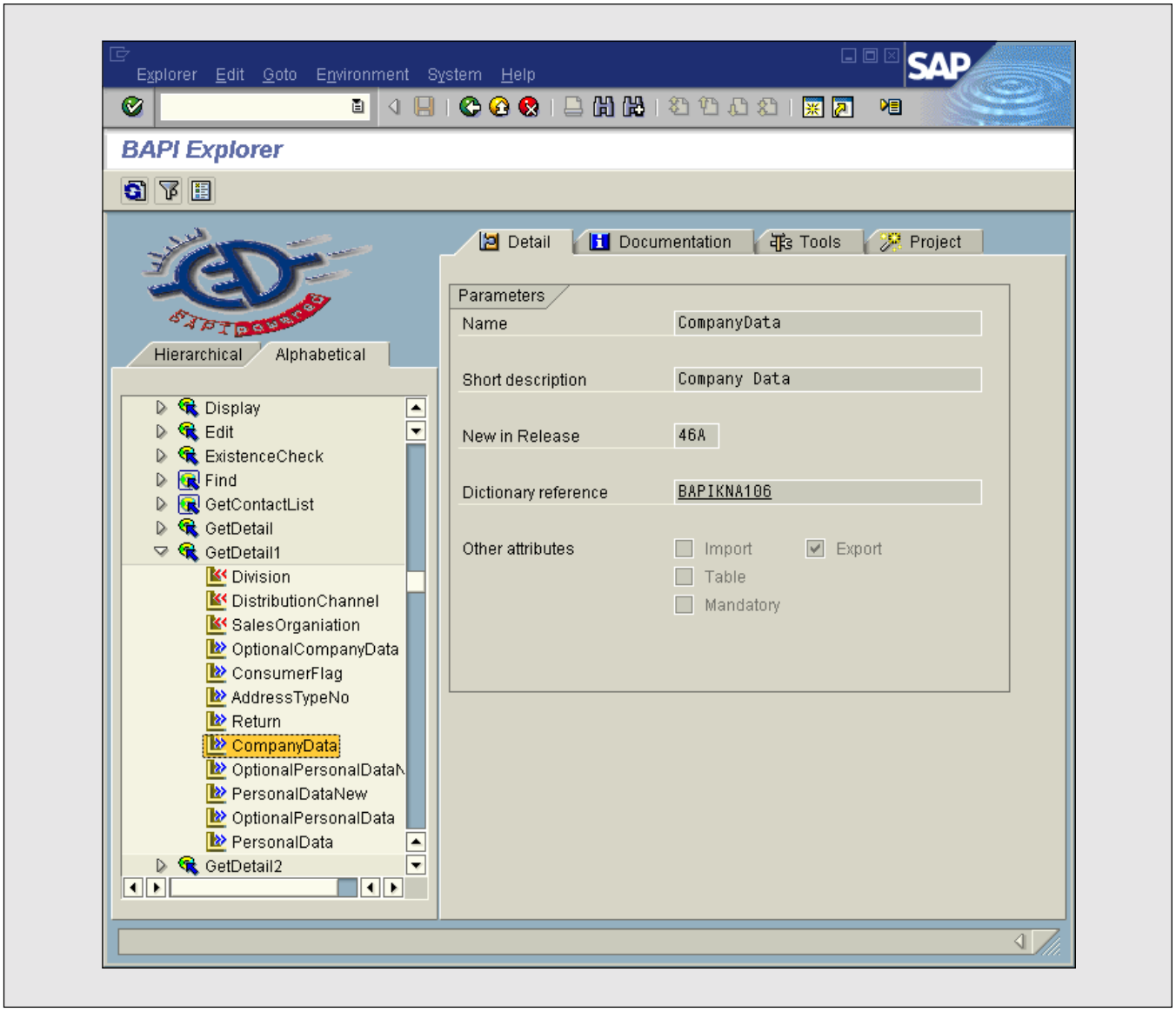
(in this case *Customer*). This should give you some background as to the functional scope of the object type.

- Check that the BAPI is released. Unreleased BAPIs should only be used if you absolutely need the new functionality and are willing to change your application when SAP makes inconsistent changes to the BAPI.

- Check that the BAPI is not a dialog BAPI. Dialog BAPIs are usually only used in applications run in SAPGUI.

- Check that the BAPI is not obsolete. If it is, look for a BAPI with a name that is almost the same, but ends with a (higher) number. *Customer.GetDetail*, for example, is obsolete and has been superceded by *Customer.GetDetail1*. In some cases, you may decide to use an obsolete BAPI because it is easier to deal with. I still prefer *SalesOrder.CreateFromDat1* over

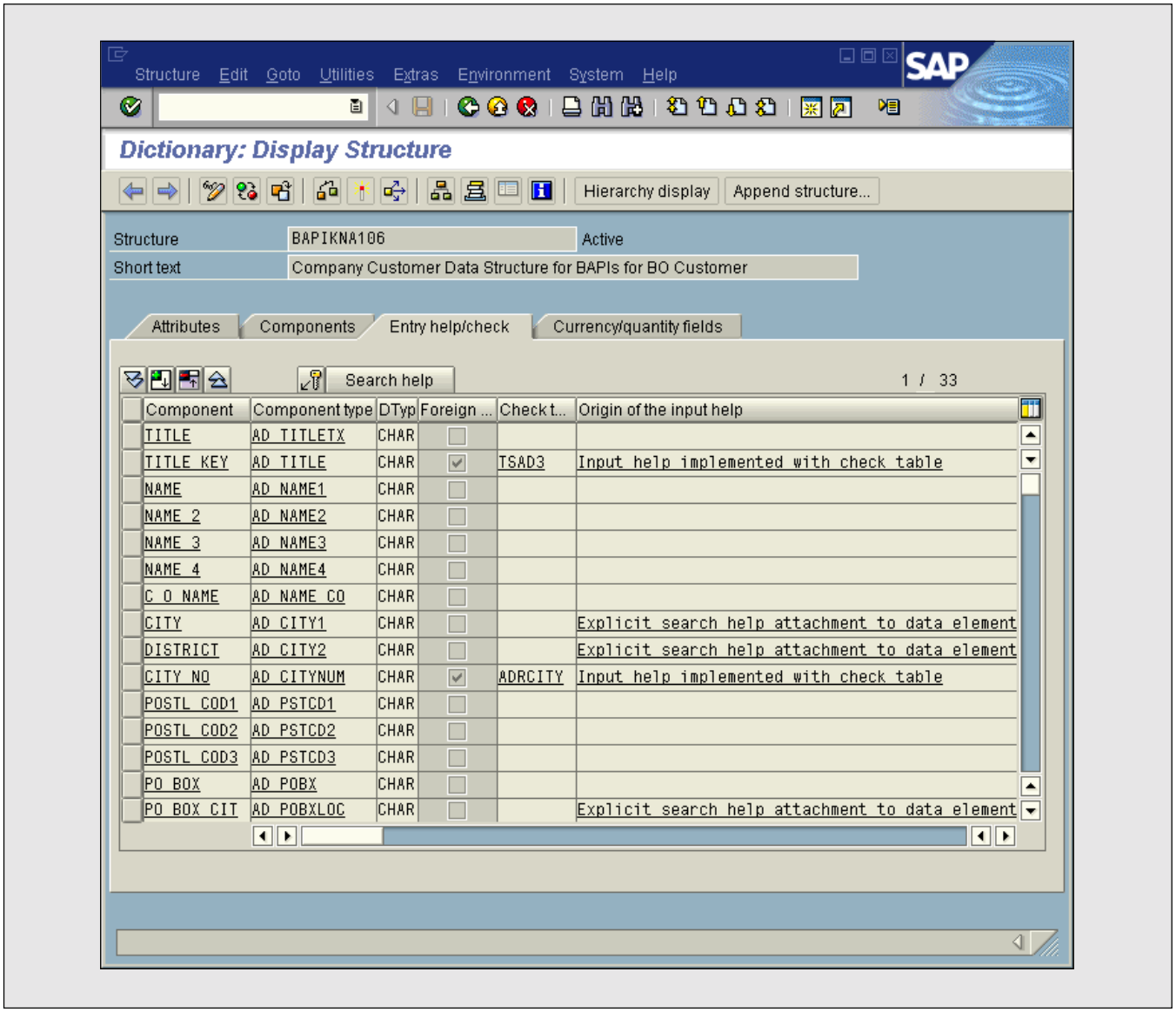
Figure 2 *A BAPI Parameter*



SalesOrder.CreateFromDat2, for instance. Obsolete BAPIs are guaranteed to continue to exist (and work) for at least two functional releases, and in many cases are still around long after that.

- Read the documentation for the BAPI itself and all of its parameters. In some cases, all the required documentation is attached to the BAPI, in others, the individual parameters have their own documentation.
- Check the important attributes of all parameters. A parameter can be mandatory or optional. It can be a simple field, a structure, or a table. A parameter can be passed to a BAPI ("import"), sent back from the BAPI ("export"), or both. For a table parameter, the information about import/export exists only on the BOR level, but not on the level of the implementing RFM.
- Decide which parameters will be important for your specific application scenario.

Figure 3 *Dictionary Information*



- Look up the RFM parameter names for all the BAPI's parameters that you are interested in. See below for details.

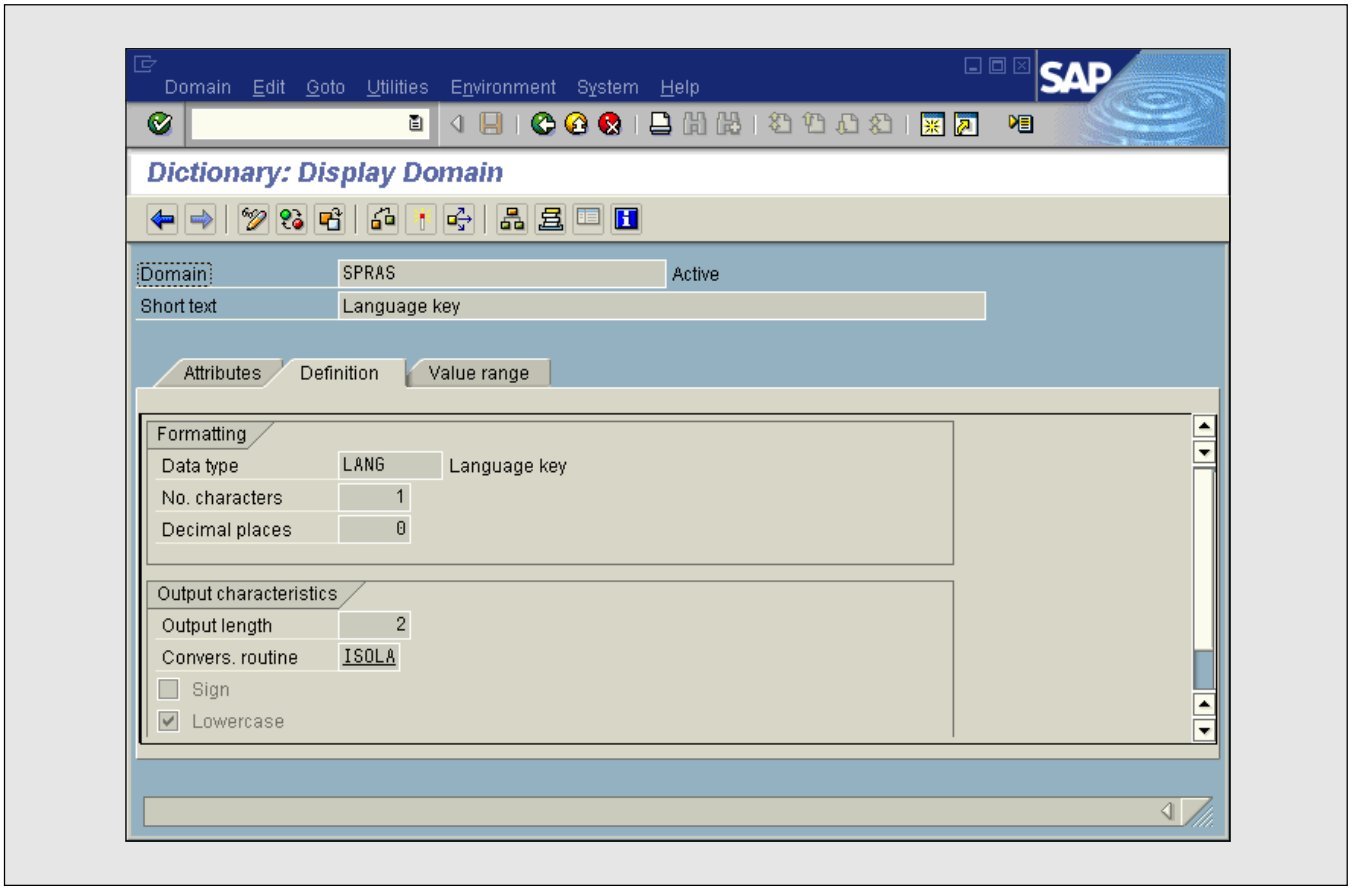
- Find out which fields within each structure or table parameter you are interested in. **Figure 2** shows the "CompanyData" parameter of *Customer.GetDetail1*. After double-clicking on the "Dictionary reference" field, you see the data dictionary information depicted in **Figure 3**.

- For each field you are interested in, check whether input help is available (cf. the column "Origin of the input help" in Figure 3). We will discuss the relevance of this later.

- For each field you are interested in, check whether a conversion routine is used. You have to drill down to the domain[8] underlying the field

[8] Remember: Fields are (usually) based on data elements, which in turn are based on domains. If you are not familiar with these terms in SAP, read the online documentation of the SAP data dictionary.

Figure 4 *Domain Information*



to see this. **Figure 4** shows the domain on which field "LANGU" is based. Note that there is a conversion routine defined, viz. "ISOLA". More details on this also later.

Parameter Information in the Dictionary

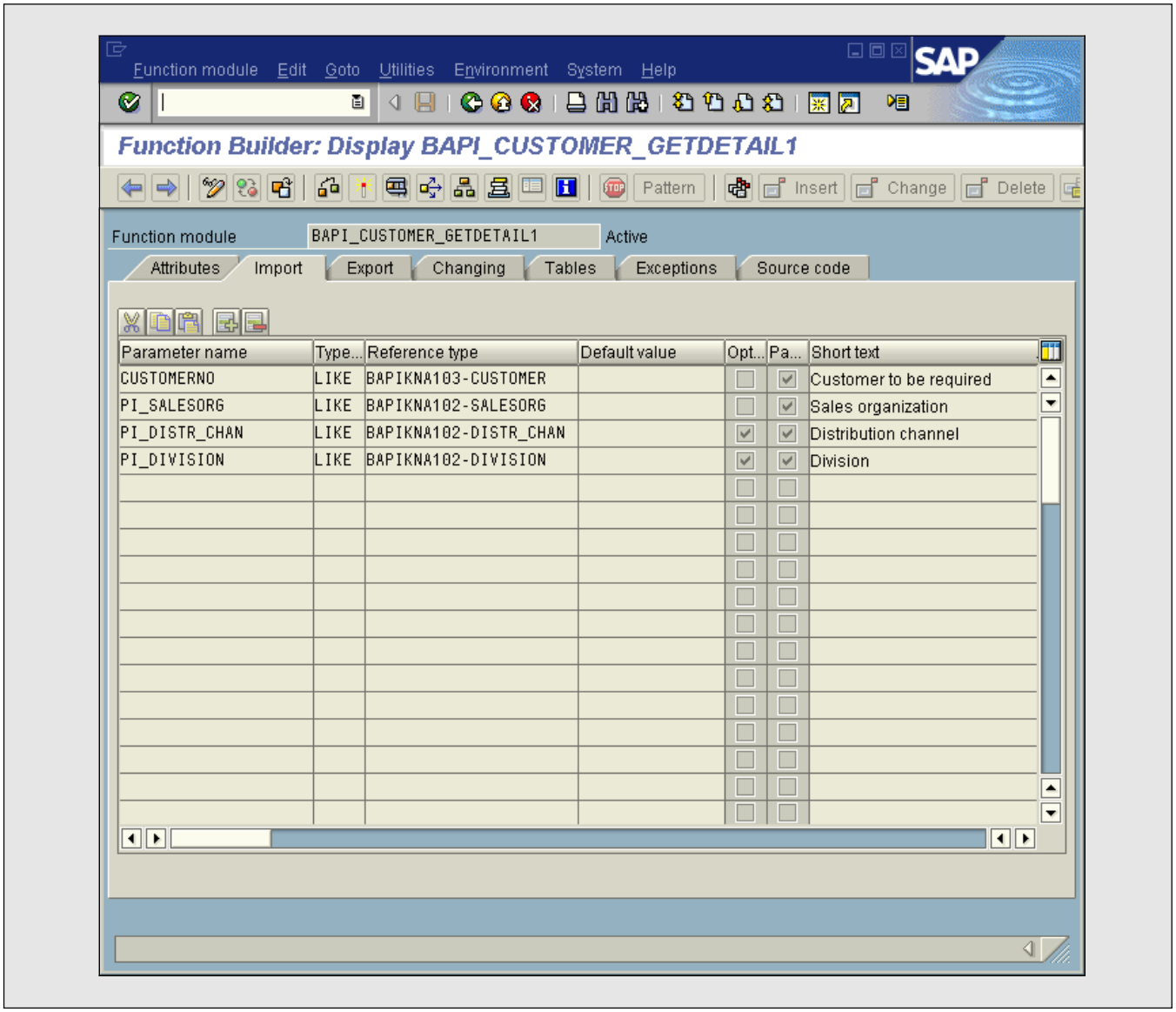
For each field that you want to use in your application, you obviously want to know the data type and length information. But I mentioned earlier that you should also check whether a field has input help and/or a conversion routine defined. Let me explain why.

Many fields in SAP are codes (country codes, currency codes, and many more). When a BAPI returns a code to the client program it usually does not supply the associated description as well. *Customer.GetDetail1*, for instance, returns the country code for the customer, but not the name of the country. When displaying information in your web page you usually want to show the country name in addition to or even instead of the code since users cannot be expected to have memorized all codes in the SAP system. If input help is available for a field you can call the *Helpvalues.GetList* BAPI to get the description text for a code.

Similarly, when a user has to enter a code, it makes sense to supply a list of all available choices. *Helpvalues.GetList* can be used to provide the required information.

For some fields, there are two representations, an internal one used by the SAP applications and stored

Figure 5 *The RFM for BAPI Customer.GetDetail1*



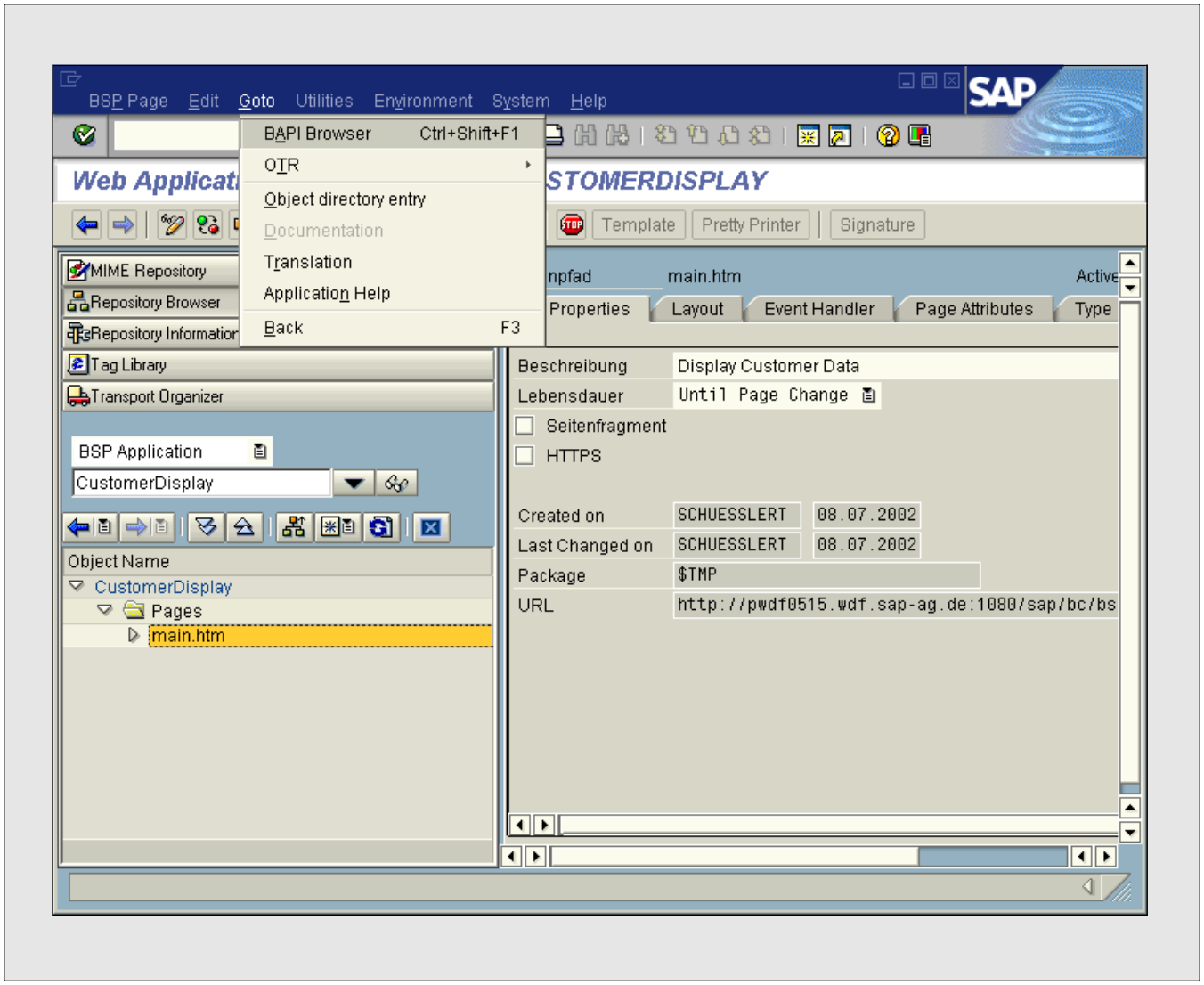
in the database, and an external one utilized in SAPGUI. Conversion routines defined on the domain level of the data dictionary convert between the two formats. Most BAPIs only use the internal representation. When passing an all-numeric customer code to a BAPI, for example, the code has to have leading zeroes, which the user normally does not want to have to type. If a conversion routine is defined for a field, you can use the conversion BAPIs of the *BapiService* object type to convert between the internal and external formats.

The sample program discussed later will exemplify both the input help and conversion routine issues.

Different Parameter Names

The parameter names of a BAPI in the BOR (cf. Figure 2) use mixed case. The parameter names of an RFM (see **Figure 5**, which shows the RFM for

Figure 6 Invoking the BAPI Browser



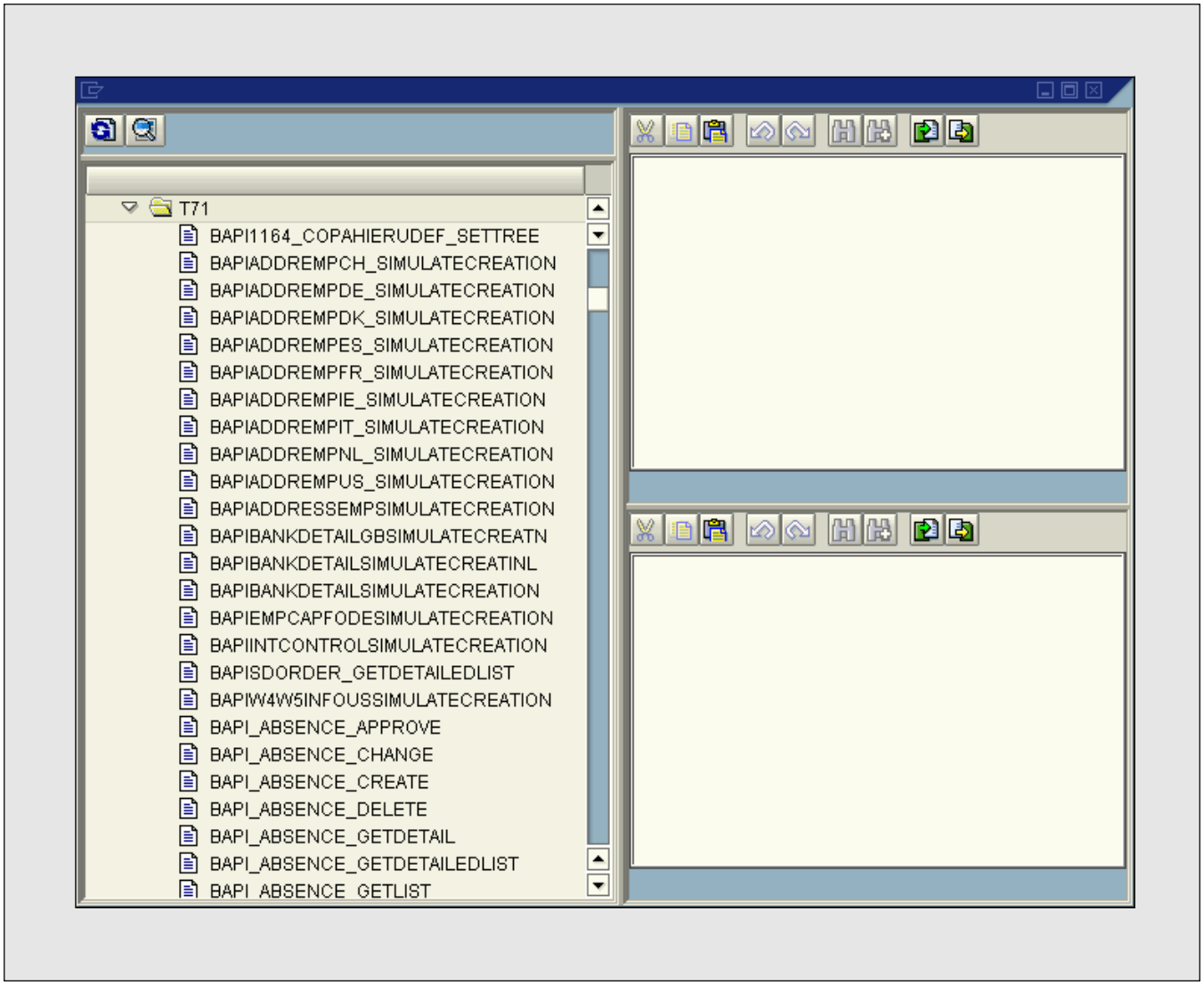
the *Customer.GetDetail1* BAPI from Figure 2) use upper case. And if you compare the two figures more closely, you will notice that there are differences beyond that. Figure 2 shows that there is a parameter called "DistributionChannel". Where is this parameter in Figure 5? It seems to be called "PI_DISTR_CHAN" here. This is due to the fact that in order to make upper case names more readable, most developers use underscores to separate different parts of the name. SAP allows the developer of a BAPI to redefine the parameter names of an RFM when it is added to the BOR as a BAPI.

Since the client code calling the BAPI in the Web Application Server actually calls the RFM that implements the BAPI, you need to know the RFM parameter names.

Generating ABAP Code for a BAPI

After you have identified and examined the BAPIs for your application you will now turn to the SAP

Figure 7 *The List of BAPIs in the BAPI Browser*



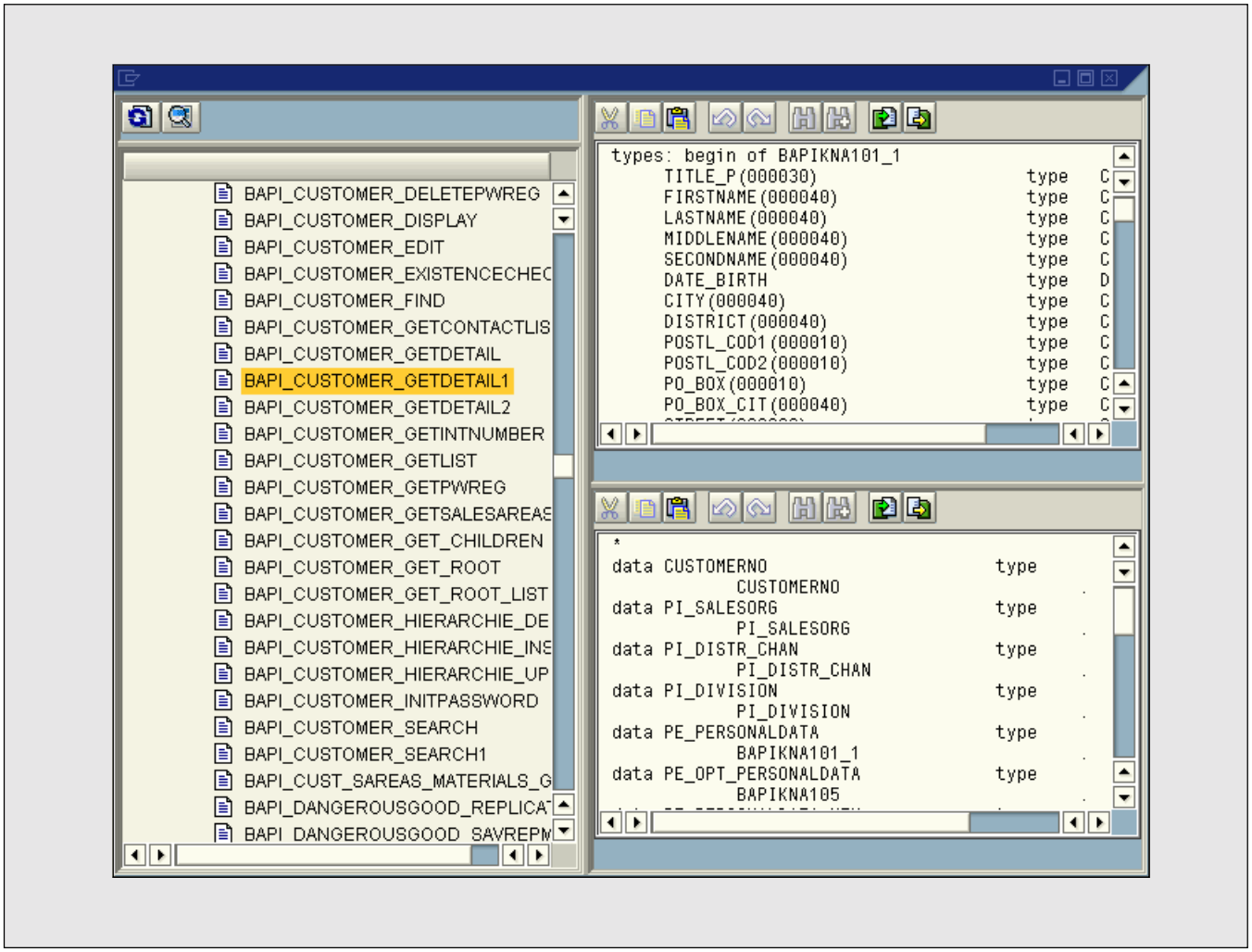
Web Application Server to build your application and invoke said BAPIs. To facilitate the latter, the SAP Web Application Server offers a BAPI Browser that can generate code for BAPIs and other RFMs[9]. This is very convenient because otherwise you would have to type all the type and data definitions yourself.

Figure 6 is a screenshot of invoking the BAPI Browser in the 6.10 Web Application Builder.

A new window is opened, which is shown in **Figure 7**. In the left pane of this window, you will initially see a list of all SAP systems defined as RFC destinations in transaction code "SM59". In Figure 7, we have already scrolled down to our chosen target system ("T71") and opened the list of BAPIs. This list is very different from what we have seen earlier in the SAP BAPI Explorer. Instead of an object-oriented view, the BAPI Browser displays a simple, alphabetical list of the RFMs implementing the BAPIs. Since you have explored all relevant metadata earlier (see above), this does not present a

[9] In this article, we will only deal with BAPIs.

Figure 8 Generating Source Code for a BAPI



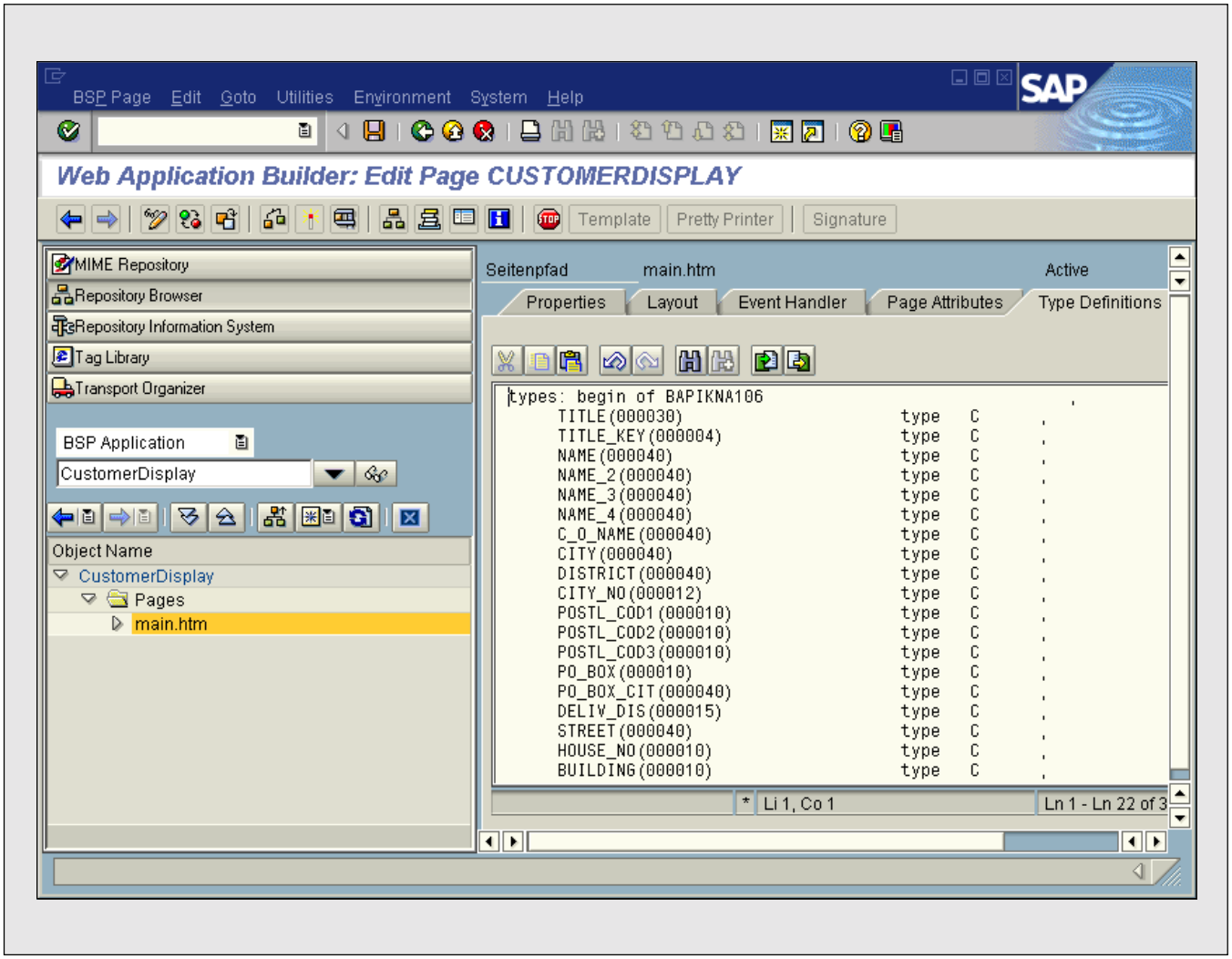
problem. All you have to do is remember (or look up again) the names of the RFMs underneath the BAPIs you want to use and locate them in the list.

In **Figure 8**, I have scrolled down to the first BAPI we will need for our sample application, *Customer.GetDetail1*, implemented by RFM "BAPI_CUSTOMER_GETDETAIL1". Then I double-clicked on the RFM name in order to generate the ABAP source code in the two panels on the right side of the window. The upper panel contains type definitions for all structure and table parameters of the BAPI, the lower panel data definitions and sample code to execute the BAPI. You could copy all type definitions to the appropriate area of your BSP page, but it is probably better to copy only the ones you need.[10] Since you have examined each required BAPI and figured out which parameters you want to use this will be easy. The same applies to the data definitions in the lower panel. Copying only the required ones will make it easier to understand the completed application later on.

Since the BAPI Browser window is modal (i.e., you cannot work in the Web Application Builder before you close the BAPI Browser) it is best to

[10] This should make future maintenance of your application easier.

Figure 9 Type Definitions for the BSP Page



copy the contents of the upper and lower panes into a local editor (Notepad, for example) so that you do not have to return to the BAPI Browser again for the same BAPI.

Building the Sample Application

Our sample application has a humble goal: let the user enter a customer code and display some information for that customer. Since we just finished generating ABAP source code for our main BAPI call, let us utilize the generated code before discussing other details of the application.

Each BSP page has an area for type definitions. Select the type definitions you require and paste them into that area (see **Figure 9**).

The BAPI invocation itself needs to happen when the user has entered the customer code and pressed the submit button of the page. In terms of a BSP application this means that we will call the BAPI from within the InputProcessing event of the BSP page.

Figure 10 Code for the InputProcessing Event

```
* event handler for checking and processing user input and
* for defining navigation
data PE_COMPANYDATA                  type
            BAPIKNA106                        .
data RETURN                          type
            BAPIRETURN1                       .

CALL FUNCTION 'BAPI_CUSTOMER_GETDETAIL1'
          DESTINATION 'T71'
   EXPORTING
     CUSTOMERNO                     = CustNum
   IMPORTING
     PE_COMPANYDATA                 = PE_COMPANYDATA
     RETURN                         = RETURN.
ErrorMessage = RETURN-MESSAGE.
CompanyName = PE_COMPANYDATA-NAME.
City        = PE_COMPANYDATA-CITY.
Country     = PE_COMPANYDATA-COUNTRY.
```

Figure 10 lists the ABAP source code for the first version of our sample program. The data definitions were copied (selectively) from the code generated by the BAPI Browser. The function call was copied and subsequently modified. Unused parameters were discarded and the "CUSTOMERNO" parameter set to the customer code entered by the user (more about that in a minute). Finally, after calling the BAPI, we copy some of the data into fields used in the actual HTML page.

This page is defined in the layout area of the BSP page. **Listing 1** contains the complete page definition with its HTML tags and ABAP scripting parts. The latter are identified by the "<% %>" tags. As you can see there is precious little actual scripting in this page. All we do is copy some variable data into the resulting HTML stream.[11]

[11] It is highly recommended that you have as little ABAP code in the layout as possible since in a normal, i.e., more sophisticated, application the page is designed by a web designer who normally does not know ABAP and does not like to have to work around tons of code. Obviously, most applications will require more scripting in the layout than our sample application, but it should be the minimum possible under the circumstances, nevertheless. All application logic belongs in the other events of the page or even — for more complex applications — in ABAP classes.

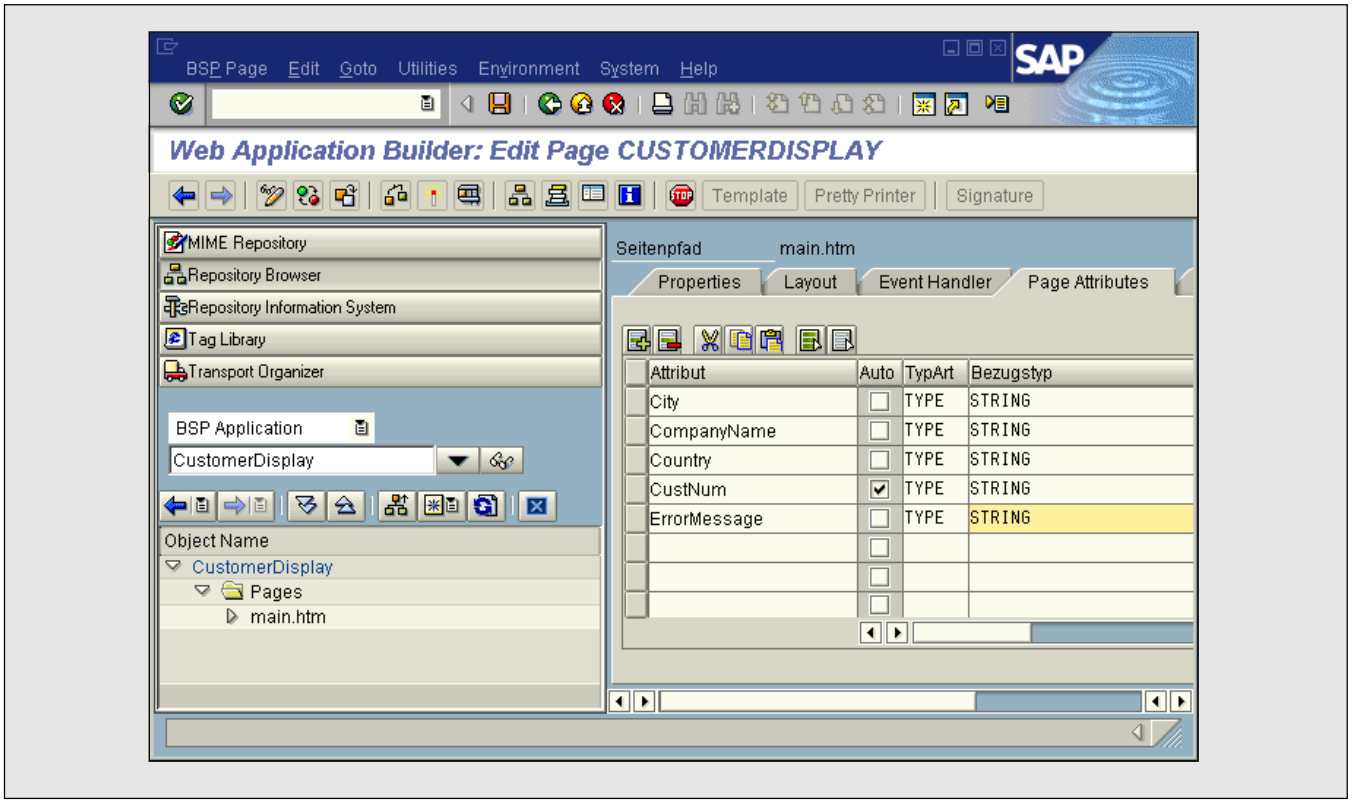
Listing 1: Layout Coding

```
<%@page language="abap"%>
<html>
  <head>
```



```
    <link rel="stylesheet" href="../../sap/public/bc/bsp/styles/sapbsp.css">
    <title> Display Customer Data </title>
  </head>
  <body class="bspBody1">

    <%=ErrorMessage%>
    <form>
      <table width="100%" cellspacing="0" cellpadding="0" border="0">

        <tr>
          <td> Customer No.: </td> <td>
            <input type="text"
                   name="CustNum"
                   size="10"
                   value="<%=CustNum%>">
          </td> <td>
            <input type="submit"
                   name="OnInputProcessing"
                   value="Retrieve Data">
          </td>
        </tr>

        <tr>
          <td> Company Name: </td>
          <td> <%=CompanyName%> </td>
        </tr>

        <tr>
          <td> City: </td>
          <td> <%=City%> </td>
        </tr>

        <tr>
          <td> Country: </td>
          <td> <%=Country%> </td>
        </tr>

      </table>
    </form>

  </body>
</html>
```

Figure 11 The Page Attributes



One more thing before we are ready to test: we need to define the page attributes, as shown in **Figure 11**.

"CustNum" is the customer code entered by the user. It is marked as an Auto attribute so that it is automatically copied from the HTML form field to our variable.

"City", "CompanyName", and "Country" are used to copy the pertinent information from the BAPI parameter to the HTML code. See Figure 10 for the ABAP code that fills these fields after the BAPI call.

"ErrorMessage" is used to display the text of the error message that the BAPI has returned. This field will be empty after a successful BAPI call.

We are ready to go. After activating our BSP application, we invoke the test function of the Web Application Server and see **Figure 12** in the browser of our choice.

After entering customer code "0000001400" and pressing the "Retrieve Data" button, we see the result shown in **Figure 13**. So far, so good. Most users — especially those "spoiled" by SAPGUI — would not dream of entering leading zeroes for a customer code. **Figure 14** contains the sorry result of omitting the leading zeroes. An error message is all we get.

Figure 12 The Sample Application Ready to Go

Figure 13 The Sample Application Works

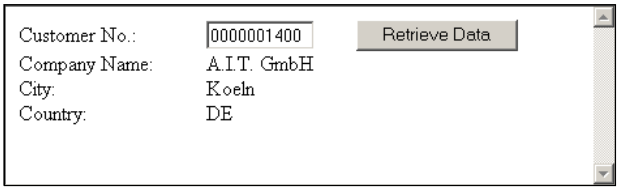
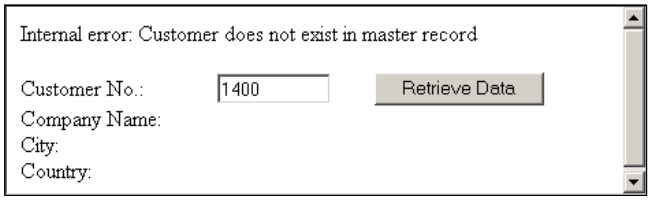


Figure 14 The Sample Application Does Not Work



Earlier, I introduced the subject of conversion routines. Now would be a good time to utilize this knowledge and call a conversion BAPI before calling *Customer.GetDetail1*.

Figure 15 contains the second iteration of the code for the InputProcessing event. I called the BAPI Browser again to generate code for

Figure 15 Conversion Added to the InputProcessing Code

```
* event handler for checking and processing user input and
* for defining navigation
data PE_COMPANYDATA                  type
            BAPIKNA106                          .
data RETURN                          type
            BAPIRETURN1                         .

data: t_data   TYPE TABLE OF bapiconvrs.
data: wa_data  LIKE LINE  OF t_data.

wa_data-objname    = 'Customer'.
wa_data-method     = 'Create'.
wa_data-parameter  = 'Customer'.
wa_data-ext_format = CustNum.
APPEND wa_data TO t_data.

CALL FUNCTION 'BAPI_CONVERSION_EXT2INT1'
          DESTINATION 'T71'
  TABLES
    data   = t_data.

  READ TABLE t_data INDEX 1 INTO wa_data.
  CustNum = wa_data-int_format.

CALL FUNCTION 'BAPI_CUSTOMER_GETDETAIL1'
          DESTINATION 'T71'
   EXPORTING
     CUSTOMERNO                      = CustNum
   IMPORTING
     PE_COMPANYDATA                  = PE_COMPANYDATA
     RETURN                          = RETURN.

ErrorMessage = RETURN-MESSAGE.
CompanyName = PE_COMPANYDATA-NAME.
City        = PE_COMPANYDATA-CITY.
Country     = PE_COMPANYDATA-COUNTRY.
```

"BAPI_CONVERSION_EXT2INT1" (the BAPI that converts from the external to the internal representation used by *Customer.GetDetail1* and most other BAPIs). Explaining the strange values I pass to the fields in work area "wa_data" is unfortunately beyond the scope of this article.

The customer code entered by the user is converted by the conversion routine called by the conversion BAPI. The result has the required leading zeroes and after passing the converted value to *Customer.GetDetail1* our application also works where Figure 12 failed.

We could be happy now, but are we? Look at Figure 13 again. The country value displayed is the code and nothing but the code. Do you expect your users to remember all 238 country codes defined in a vanilla R/3 system? *Helpvalues.GetList* will allow us to do better and display the code and the name of the country.

Figure 16 The Country Name Added to the Page

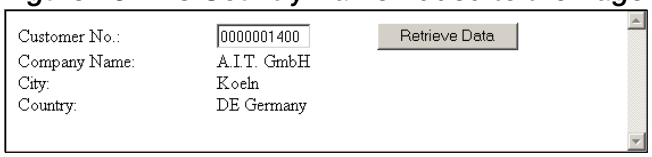


Figure 16 shows the completed application in action and **Listing 2** contains the complete source code for the InputProcessing event.[12] In **Listing 3**, you will find all type definitions used by the sample application.

[12] For details on using the Helpvalues object type, see my article on this subject in the March/April 2002 issue of this publication.

Listing 2: Complete InputProcessing Coding

```
data PE_COMPANYDATA                  type
            BAPIKNA106                        .
data RETURN                          type
            BAPIRETURN1                       .
data SELECTION_FOR_HELPVALUES        type
            SELECTION_FOR_HELPVALUES          .
data HELPVALUES                      type
            HELPVALUES                        .

data: wa_HELPVALUES LIKE LINE OF HELPVALUES.
data: wa_SELECTION_FOR_HELPVALUES LIKE LINE OF SELECTION_FOR_HELPVALUES.
data: t_data   TYPE TABLE OF bapiconvrs.
data: wa_data  LIKE LINE  OF t_data.

wa_data-objname    = 'Customer'.
wa_data-method     = 'Create'.
wa_data-parameter  = 'Customer'.
wa_data-ext_format = CustNum.
APPEND wa_data TO t_data.

CALL FUNCTION 'BAPI_CONVERSION_EXT2INT1'
          DESTINATION 'T71'
```



```
  TABLES
    data   = t_data.

READ TABLE t_data INDEX 1 INTO wa_data.
CustNum = wa_data-int_format.

CALL FUNCTION 'BAPI_CUSTOMER_GETDETAIL1'
          DESTINATION 'T71'
   EXPORTING
     CUSTOMERNO                    = CustNum
   IMPORTING
     PE_COMPANYDATA                = PE_COMPANYDATA
     RETURN                        = RETURN.

ErrorMessage = RETURN-MESSAGE.
CompanyName = PE_COMPANYDATA-NAME.
City        = PE_COMPANYDATA-CITY.

wa_SELECTION_FOR_HELPVALUES-SELECT_FLD = 'LAND1'.
wa_SELECTION_FOR_HELPVALUES-SIGN       = 'I'.
wa_SELECTION_FOR_HELPVALUES-OPTION     = 'EQ'.
wa_SELECTION_FOR_HELPVALUES-LOW        = PE_COMPANYDATA-COUNTRY.
APPEND wa_SELECTION_FOR_HELPVALUES TO SELECTION_FOR_HELPVALUES.

CALL FUNCTION 'BAPI_HELPVALUES_GET'
          DESTINATION 'T71'
   EXPORTING
     OBJNAME                       = 'Customer'
     METHOD                        = 'GetDetail1'
     PARAMETER                     = 'CompanyData'
     FIELD                         = 'COUNTRY'
   TABLES
     SELECTION_FOR_HELPVALUES      = SELECTION_FOR_HELPVALUES
     HELPVALUES                    = HELPVALUES.

READ TABLE HELPVALUES INDEX 1 INTO wa_HELPVALUES.

Country     = wa_HELPVALUES-HELPVALUES.
```

Listing 3: Type Definitions

```
types: begin of BAPIKNA106                                  ,
     TITLE(000030)                    type   C      ,
     TITLE_KEY(000004)                type   C      ,
     NAME(000040)                     type   C      ,
```

(continued on next page)

(continued from previous page)

```
      NAME_2(000040)                    type   C      ,
      NAME_3(000040)                    type   C      ,
      NAME_4(000040)                    type   C      ,
      C_O_NAME(000040)                  type   C      ,
      CITY(000040)                      type   C      ,
      DISTRICT(000040)                  type   C      ,
      CITY_NO(000012)                   type   C      ,
      POSTL_COD1(000010)                type   C      ,
      POSTL_COD2(000010)                type   C      ,
      POSTL_COD3(000010)                type   C      ,
      PO_BOX(000010)                    type   C      ,
      PO_BOX_CIT(000040)                type   C      ,
      DELIV_DIS(000015)                 type   C      ,
      STREET(000040)                    type   C      ,
      HOUSE_NO(000010)                  type   C      ,
      BUILDING(000010)                  type   C      ,
      FLOOR(000010)                     type   C      ,
      ROOM_NO(000010)                   type   C      ,
      COUNTRY(000003)                   type   C      ,
      LANGU(000001)                     type   C      ,
      REGION(000003)                    type   C      ,
      TEL1_NUMBR(000030)                type   C      ,
      TEL1_EXT(000010)                  type   C      ,
      FAX_NUMBER(000030)                type   C      ,
      FAX_EXTENS(000010)                type   C      ,
      E_MAIL(000241)                    type   C      ,
      COUNTRYISO(000002)                type   C      ,
      LANGU_ISO(000002)                 type   C      ,
      CURRENCY(000005)                  type   C      ,
      CURRENCY_ISO(000003)              type   C      ,
        end of   BAPIKNA106                                .

types: begin of BAPIRETURN1                              ,
      TYPE(000001)                      type   C      ,
      ID(000020)                        type   C      ,
      NUMBER(000003)                    type   N      ,
      MESSAGE(000220)                   type   C      ,
      LOG_NO(000020)                    type   C      ,
      LOG_MSG_NO(000006)                type   N      ,
      MESSAGE_V1(000050)                type   C      ,
      MESSAGE_V2(000050)                type   C      ,
      MESSAGE_V3(000050)                type   C      ,
      MESSAGE_V4(000050)                type   C      ,
        end of   BAPIRETURN1                               .

types: begin of BAPICONVRS,
```



```
         OBJTYPE(000010)               type C,
         OBJNAME(000032)               type C,
         METHOD(000032)                type C,
         PARAMETER(000032)             type C,
         FIELD(000030)                 type C,
         ROWNUMBER                     type I,
         INT_FORMAT(000255)            type C,
         EXT_FORMAT(000255)            type C,
         CONV_LEN(000003)              type N,
       end of   BAPICONVRS                          .

types: begin of BAPIRET2,
         TYPE(000001)                  type C,
         ID(000020)                    type C,
         NUMBER(000003)                type N,
         MESSAGE(000220)               type C,
         LOG_NO(000020)                type C,
         LOG_MSG_NO(000006)            type N,
         MESSAGE_V1(000050)            type C,
         MESSAGE_V2(000050)            type C,
         MESSAGE_V3(000050)            type C,
         MESSAGE_V4(000050)            type C,
         PARAMETER(000032)             type C,
         ROW                           type I,
         FIELD(000030)                 type C,
         SYSTEM(000010)                type C,
       end of   BAPIRET2                          .

types: begin of BAPIF4B                                ,
     SELECT_FLD(000030)                type   C      ,
     SIGN(000001)                      type   C      ,
     OPTION(000002)                    type   C      ,
     LOW(000030)                       type   C      ,
     HIGH(000030)                      type   C      ,
       end of   BAPIF4B                             .

types: begin of BAPIF4C                                ,
     HELPVALUES(000255)                type   C      ,
       end of   BAPIF4C                             .

types: SELECTION_FOR_HELPVALUES        type table of
          BAPIF4B                                        .

types: HELPVALUES                      type table of
          BAPIF4C                                        .
```

Personal Conclusion

Developing web applications with the SAP Web Application Server (in ABAP) is easier than in any other environment I have tried. If you add the support that SAP provides to facilitate the calling of BAPIs in the SAP application components like R/3 you have a sure winner. Provided that you can live with some of the idiosyncrasies of ABAP (and you probably would not have finished this article if you could not) the decision between ABAP and Java at least for business web applications should be an easy one.[13]

[13] I am well aware that some readers will disagree with this statement, but I will not go into more details for now. A thorough justification would require a complete article on its own…

Thomas G. Schuessler is the founder of ARAsoft (www.arasoft.de), a company offering products, consulting, custom development, and training to a worldwide base of customers. The company specializes in integration between SAP and non-SAP components and applications. ARAsoft offers various products for BAPI-enabled programs on the Windows and Java platforms. These products facilitate the development of desktop and Internet applications that communicate with R/3. Thomas is the author of SAP's BIT525 "Developing BAPI-enabled Web Applications with Visual Basic" and BIT526 "Developing BAPI-enabled Web Applications with Java" classes, which he teaches in Germany and in English-speaking countries. Thomas is a regularly featured speaker at SAP TechEd and SAPPHIRE conferences. Prior to founding ARAsoft in 1993, he worked with SAP AG and SAP America for seven years. Thomas can be contacted at thomas.schuessler@sap.com or at tgs@arasoft.de.

