Enabling Point-and-Click Data Entry Assistance for Your BAPI Applications

Thomas G. Schuessler

Thomas G. Schuessler is the founder of ARAsoft, a company that specializes in integration between SAP and non-SAP systems, and offers products, consulting, custom development, and training to customers worldwide. Prior to founding ARAsoft in 1993, he worked with SAP AG and SAP America for seven years. Thomas is a prolific author of articles, books, and training materials. Thomas can be reached at tgs@arasoft.de or thomas.schuessler@sap.com.

If you are building BAPI-enabled applications as alternative front-ends to an SAP system, the GUIs for these applications should ideally provide facilities that help a user to enter data, like customer numbers and country codes, without requiring the user to actually remember all that data. These facilities can range from simple combo boxes to elaborate selection screens.

Within SAP's standard user interface, SAPGUI, these facilities are summarily known as *Input Help*. Input Help uses a pop-up window from which the user can select an entry. In simple cases (e.g., country codes), a list of codes and descriptions (e.g., country codes and country names) is sufficient. For more complex object types like customer, though, the user should be able to search on different sets of criteria. Sometimes the user wants to search for customers based on address attributes (e.g., city and zip code), sometimes based on the account group the customer belongs to. These more complex requirements are supported via Match Codes. A Match Code is a secondary index that combines multiple columns from one or more tables. There can be multiple Match Codes for one object type so that users can search on different combinations of criteria depending on their current requirements.

SAPGUI always offers the appropriate Input Help for a field. Straightforward lists with codes and descriptions are used for simple cases (e.g., countries), sophisticated selection dialogs using Match Codes for more complex business objects (e.g., customers).

How can we provide facilities similar to SAPGUI's Input Help in our BAPI-enabled applications? If specific BAPIs exist that allow

us to retrieve the required information (e.g., a list of all company codes can be accessed via `CompanyCode.GetList`), then we simply use these BAPIs. If the necessary BAPIs are missing, we have to use a service object type, called `Helpvalues`. Up to and including release 4.5, `Helpvalues` only supports entities that are based on Fixed Values Lists and simple check tables (configuration tables). The first part of this article will discuss how the `Helpvalues.GetList` BAPI can be used in order to retrieve data for these entities.

With 4.6, SAP has added facilities that allow us to utilize Match Codes in BAPI-enabled applications, as well. This allows us to provide the same level of user support in our GUIs that is available to SAPGUI users. The second part of the article will show you how to make use of these new capabilities.

The Helpvalues.GetList BAPI

There are many reasons why an application might need a list of all entries of an object type, one of them being the display of this list so that the user can select one of the entries. Sometimes an application needs more information about a specific entry — e.g., to display additional information in the user interface.

If you want to retrieve a list of all entries of a specific object type in an application, you will first look for an appropriate object type in the Business Object Repository (BOR). If such an object type can be found, you expect it to have a `GetList` BAPI that allows you to access a list of all or selected entries. To obtain a list of all sales orders for a customer, for example, you would expect to be able to use a BAPI called `SalesOrder.GetList`. To find out more details about one particular sales order (e.g., the line items contained in it), you would search for a BAPI of the `SalesOrder` object type called `GetDetail` or `GetStatus`. Since both `SalesOrder.GetList` and `SalesOrder.GetStatus` actually exist in the BOR, in this example we have no problem accessing the required information.

Quite often, though, we will be less fortunate. If we need a list of all countries, for example, in order to let a user assign a country code to a new customer, then we would search in vain for a `Country` object type in the BOR. Is there an alternative? Fortunately, there is. To retrieve information about entities for which there are no associated object types, SAP provides the Helpvalues mechanism, specifically the `Helpvalues.GetList` BAPI.

Before we start to look at the `Helpvalues.GetList` BAPI in more detail, let us summarize how we might use the information provided by the mechanism:

- We need a list of valid codes for an entity in order to validate a code entered by the user or passed by a client program. In order to ensure that a country code typed by the user is correct, for example, we need a list of all valid country codes.[1]

- In order to provide the user with a list of valid codes and associated descriptions, we need all the codes and their descriptive texts. One way of facilitating the entry of a valid country code, for example, would be a multi-column dropdown combo box with the country codes and names.

- Oftentimes, we want to display information returned by a BAPI. After a call to `Customer.GetDetail`, we might display the customer's address. It would definitely be a good idea to show the country name in addition to — or even instead of — the country code that was returned by the BAPI call.

---

[1] Just in case you feel tempted to simply accept whatever the user types and let the SAP system worry about the validation, I would like you to take the following two consequences into account: Firstly, this could be bad for the performance of the SAP system, if the user keeps on typing incorrect codes that SAP must verify. Using one BAPI call to retrieve the list of all country codes is much better than making many calls to verify different attempts by the user. Secondly, the user would probably much prefer a nice way of selecting a valid code rather than having to guess the correct code for a country.

Figure 1 *The Helpvalues Object Type*

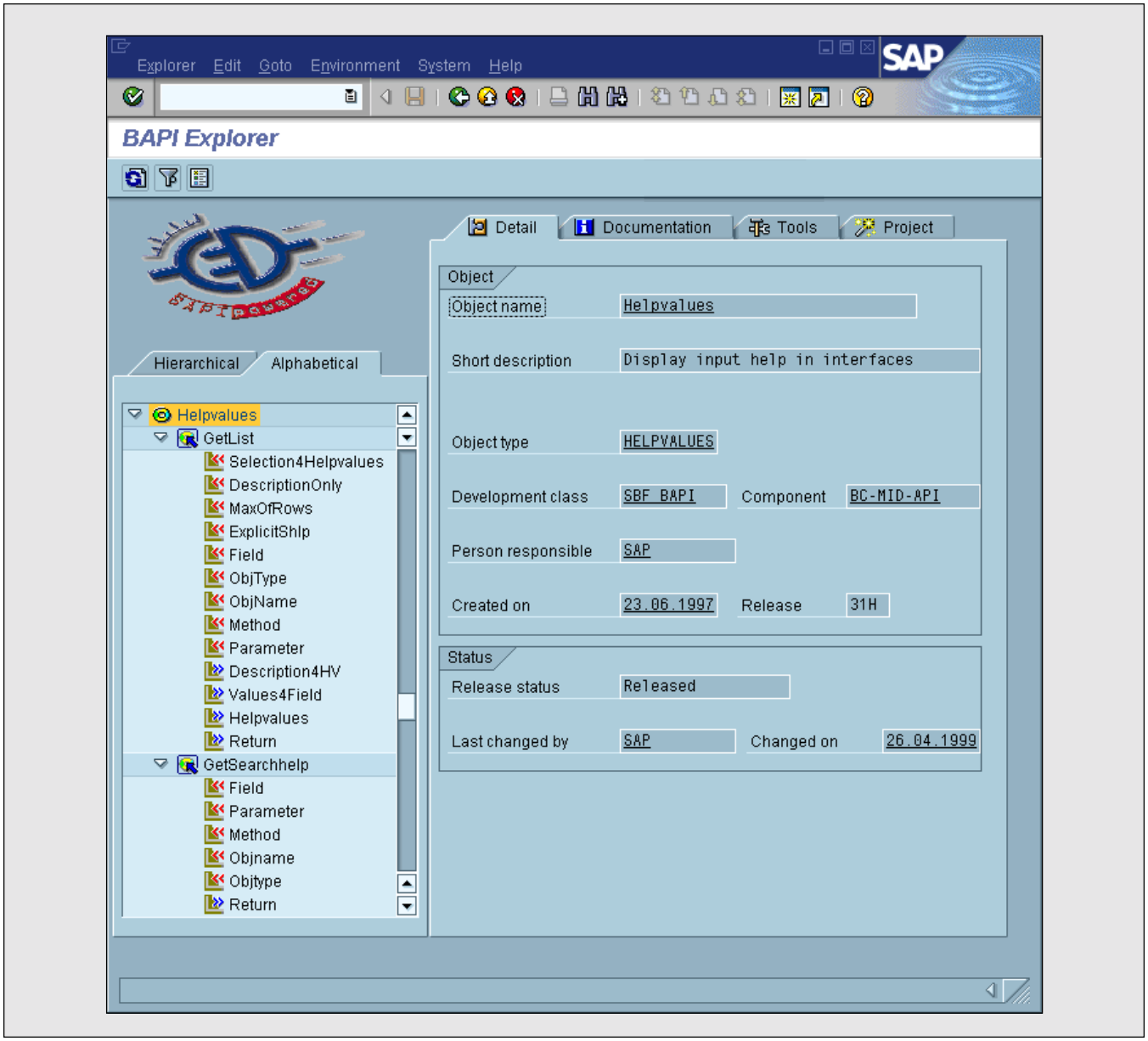


Figure 1 shows the `Helpvalues` object type displayed in the BAPI Explorer (transaction code BAPI). As you can see, there are two BAPIs. The `GetList` BAPI has been available since 3.1 and will be the focus of the first half of the article. The other BAPI, `GetSearchhelp`, was added in 4.6. This new BAPI, which will be discussed in the second part of the article, is the one that enables us to use Match Codes in our BAPI-enabled applications.

`Helpvalues.GetList` is the equivalent to SAPGUI's Input Help functionality for BAPI-enabled applications and uses the same underlying mechanisms. The BAPI returns a table with data for the entity as well as one with the structure information describing that data. The number of columns that will be returned for an entity can differ. For most entities, we will get a code column and a description column, but sometimes we get more columns and (rarely) we do not get a description. A detailed description of the returned information and how to interpret it will be given later.

✔ Tip

`Helpvalues.GetList` is a generic mechanism to retrieve information about entities that have no associated object type with suitable BAPIs. We should not use it in situations where the required information can be obtained by using specific BAPIs (like `SalesOrder.GetList` and `GetStatus`). There are two reasons for this recommendation that we can establish right now (and additional ones that will become clear as we explain the details of the generic Helpvalues mechanism):

- *The performance of any generic mechanism is inferior to a specific implementation.*
- *Writing client code using a generic mechanism is more work than to use a specific BAPI.*

Before we discuss the parameters required to use `Helpvalues.GetList`, let us talk about the prerequisites (in the SAP Data Dictionary) for SAPGUI's Input Help facilities. Since `Helpvalues.GetList` uses the same underlying mechanism they are also prerequisites for this BAPI to be able to access information about an entity.

Input Help and Helpvalues.GetList Prerequisites in R/3

SAPGUI allows a user to get Input Help for fields where this makes sense. The user can ask for a list of all defined sales order types when starting to enter a new sales order, for example. On the other hand, there is no input help for the purchase order number of a new sales order because a purchase order number is just an arbitrary string (as far as the sales order is concerned).

Input Help can only be provided if appropriate definitions were made in the SAP Data Dictionary. There are several ways of implementing Input Help, but the two most important and most common ones are:

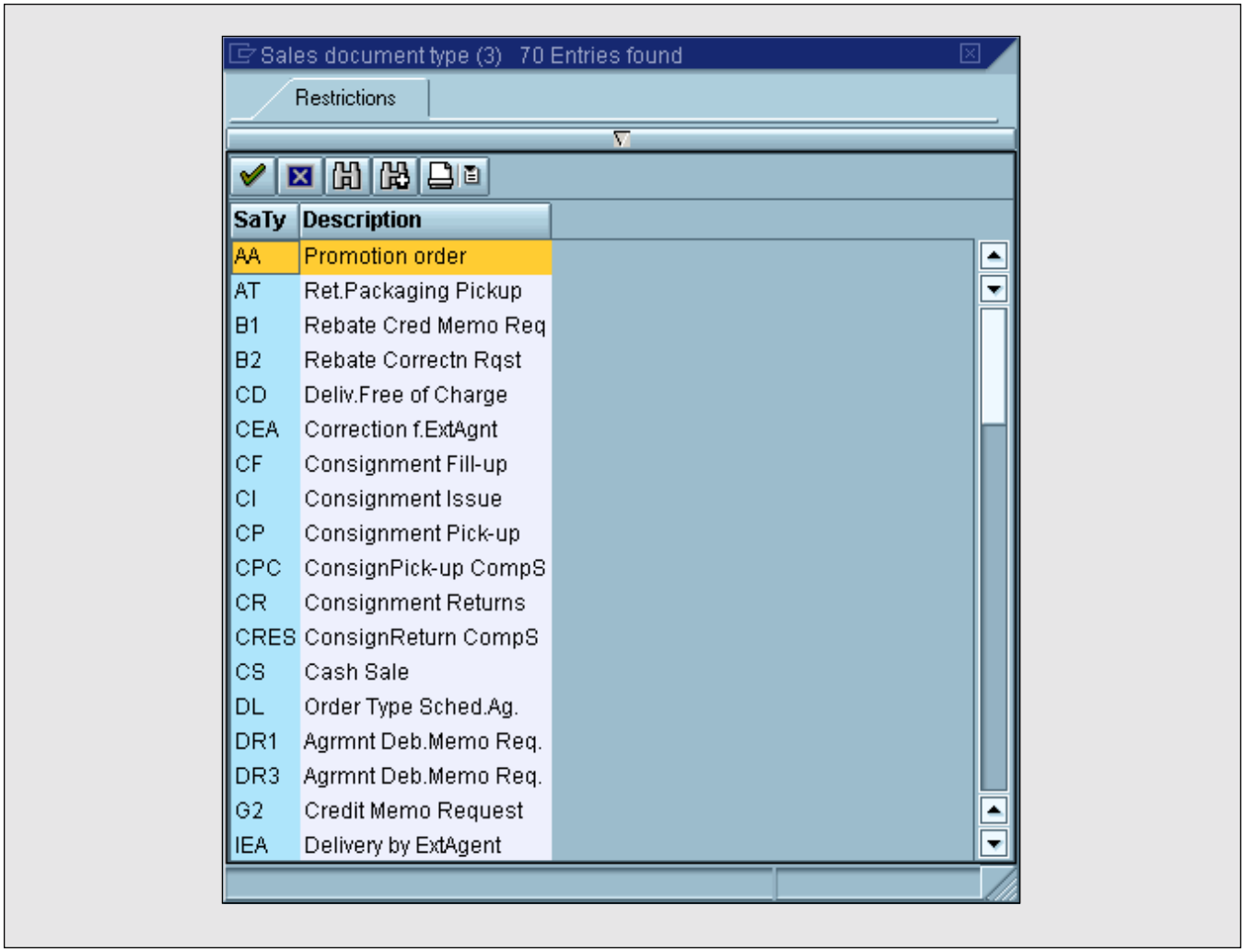
- The field is based on a domain with a Fixed Values List. This list is stored with the domain, not in a separate check table. This approach is used in SAP for comparatively small lists of values where the customer will never need to change the list of values. The `GESCH` domain (used for the gender code of a person) is a good example.
- A check table is defined for the field. This table contains all legal values for the field. An example would be table `TVAK`, which contains all valid sales order types.

Figure 2 shows Input Help at work using check table `TVAK`.

The user has asked for help concerning the list of all defined sales order types. As you can see, SAPGUI displays not just the codes, but also the associated descriptions. If you were to look up table `TVAK` in the Data Dictionary, though, you would notice two things:

- The table contains quite a few fields.
- None of the fields contains the description.

Figure 2 *Input Help for Sales Order Types*



So how does SAPGUI know how to suppress all the other fields from TVAK and where does the description text come from? Let me answer the second question first.

Since SAP supports multiple languages, language-dependent data (like the description texts) is stored in separate tables (in this case table TVAK), the key fields of which include a language code field.

The Input Help mechanism must now somehow extract the codes from table TVAK and combine them with the associated descriptions from table TVAK, using the user's logon language code in order to get the texts in the appropriate language. This is accomplished by defining a *Search Help*. A Search Help allows the developer to control all aspects of the Input Help offered for a specific field. In our example, the two tables involved must be joined in a specific way so that the user only sees sales order types and descriptions.

In almost all cases where Input Help works based on check tables, Search Helps are used.

All required Data Dictionary definitions are made by the ABAP applications developers. We will just reap the benefits of being able to use Input Help and

Helpvalues.GetList for entities that have been set up accordingly.

Identifying an Entity

In SAPGUI, the user selects Input Help by pressing the appropriate icon or the F4 key. What is the equivalent to this when we want to use Helpvalues.GetList? Since Helpvalues.GetList is a generic method, we must tell it which entity we are interested in. The requirement for a call to Helpvalues.GetList usually arises when we have called a BAPI whose returned data we need to interpret (for example, find the name of the country whose code was returned by the BAPI) or when we prepare to call a BAPI that requires a valid code in a parameter (for example, pass a valid sales order type in a call to SalesOrder.CreateFromDat1). Therefore, Helpvalues.GetList lets us identify the entity we are interested in by specifying a field in a BAPI parameter. We do this by specifying the business object, method, parameter, and field names that refer to a field in a dictionary structure. The field name is only required if the parameter is a structure or table, since a scalar parameter directly corresponds to a field in the dictionary.[2]

In order to receive a list of all sales order types and their descriptions we could pass the following parameters: Business object name "SalesOrder", method name "CreateFromDat1", parameter name "OrderHeaderIn", and field name "DOC_TYPE". This information enables Helpvalues.GetList to identify the check table and associated Search Help defined for the sales order type entity.

Different fields in different parameters can obviously use the same check table or Fixed Values List. Does that mean that there are multiple ways to retrieve the same data? Absolutely. What matters is the entity that is described by the parameters. If different sets of parameters reference the same entity, then we will get the same information for that entity regardless of how we identify it. A list of all sales order types could also be produced with the following parameters: Business object name "SalesOrder", method name "GetList", parameter name "SalesOrders", and field name "DOC_TYPE". We do not even have to have an intention to ever call one of the BAPIs SalesOrder.CreateFromDat1 or SalesOrder.GetList. All we need is a set of parameters identifying the entity we are interested in.

Authorization Issues

Normal BAPIs include appropriate authorization checks in order to ensure that only authorized people can call the BAPI. Helpvalues.GetList, on the other hand, is a generic mechanism that — as far as we have seen until now — allows us to access any entity that can be described as a field in a BAPI. Does that mean we have a security hole here? Of course not. First of all, Fixed Values Lists are not a security problem at all — the list of gender codes, for example, is not a secret (I hope). What about check tables, though? Does Helpvalues.GetList allow us to retrieve information about all employees, customers, sales orders, and so on? Before 4.6, you could only retrieve information for check tables that contain just configuration (or, as SAP usually calls it, customizing) data.[3] The new support for Match Codes — which contains a special authorization mechanism — will be discussed later.

This limitation to customizing tables only exists for the Helpvalues.GetList BAPI. SAPGUI users can get Input Help everywhere, since they are requesting it inside a transaction code, for which they needed proper authorization in the first place.

[2] In order to make the rest of this article a little more readable, I shall use the word *field* instead of *scalar parameter or field in a structure parameter or field in a table parameter* whenever I talk about the way to identify an entity for which we want information.

[3] The exact definition — in case you are interested — is this: A table, the delivery class of which is *not* "A" or "L" is considered to be a customizing table.

Figure 3 Helpvalues.GetList Parameters

Parameter Name	Data Type	Import/ Export	Mandatory	Description
ObjType	Character 10	Import		Internal name of object type (e.g., "BUS0002")
ObjName	Character 32	Import		Official name of object type (e.g., "CompanyCode")
Method	Character 32	Import	Yes	Name of the BAPI (e.g., "GetDetail")
Parameter	Character 32	Import	Yes	Name of the parameter (e.g., "CompanycodeDetail")
Field	Character 30	Import		Field name in structure or table parameter (e.g., "COUNTRY")
MaxOfRows	Integer	Import		Maximum number of rows returned (default (0): all rows)
DescriptionOnly	Character 1	Import		"X": Fill only table Description4HV " ": Fill tables Helpvalues and Values4Field, too (default)
Return	Structure BAPIRETURN	Export		Return code information
Selection4Helpvalues	Table BAPIF4B	Imp/Exp		Selection criteria
Helpvalues	Table BAPIF4C	Export		Data for the entity, the structure of the data is described in table Description4HV
Values4Field	Table BAPIF4D	Export		Codes for the entity
Description4HV	Table BAPIF4E	Export		Structure of the data in table Helpvalues
ExplicitShlp	Structure BAPISHLP	Import		Used to select a Search Help (Match Code) in a Collective Search Help

Is this constraint of `Helpvalues.GetList` a problem for us? Not really. First of all, entities like employee usually have their own object type and suitable BAPIs (with proper authorization checking) to retrieve all or selected entries. And secondly, in 4.6 SAP has added the new Match Code capabilities. JARTU!![4]

[4] Just Another Reason To Upgrade.

The Parameters of Helpvalues.GetList

Now that we understand the Helpvalues mechanism in general, we have to discuss the parameters of `Helpvalues.GetList` in detail. All parameters for `Helpvalues.GetList` are shown in **Figure 3**.

ObjType and ObjName

One of the parameters `ObjType` and `ObjName` needs to be specified. To make your programs easier to read, you should always use `ObjName` because "`CompanyCode`" is more meaningful for humans than "`BUS0002`". Be careful to provide the object name using the proper capitalization, though ("`CompanyCode`" instead of "`companycode`"), since this parameter is case-sensitive.

Method

Here you specify the name of the BAPI.

Parameter

Here you specify the name of the parameter.

Field

This parameter is required if the parameter is a structure or table, and must otherwise be left blank.

MaxOfRows

This parameter allows us to limit the number of rows to be returned. This (before 4.6, see below for the parameter's use with Match Code processing) is of doubtful value, though, since there is no way to retrieve a certain number of rows first and then, if required, get additional rows. The default (0) ensures that all rows are retrieved.

Return

This parameter tells our program whether the call to `Helpvalues.GetList` was successful.

Helpvalues

This table contains only one field, called `HELPVALUES` (see **Figure 4**). The `HELPVALUES`[5] field contains the data returned for the entity. The field contains one or more "columns" (separated by blanks). The structure of these columns is described in the `Description4HV` table.

Values4Field

The one field in this table parameter (see **Figure 5**) contains only the codes for the entity (one in each row). This can be useful if you just want to validate a code.

Description4HV

This parameter contains the metadata for all "columns" returned for the selected entity. This information allows you to interpret the data in table `Helpvalues`. `Description4HV` contains the fields shown in **Figure 6**.

`TABNAME` and `FIELDNAME` specify where the information returned was actually found in R/3. This could be real database table fields, fields in a database view, or "virtual" fields for a Fixed Values List.

The `LANGU` column contains the language code for the language-dependent columns of this table. This will always be the language in which we are logged on.

The `POSITION` field sometimes contains zero, sometimes the correct value, and sometimes a seemingly arbitrary number. This is probably a bug, but you need not worry since the fields in `Descriptions4HV` are returned in the correct order (provided that you do not set `DescriptionOnly` to "`X`", see below) so that you know the relative position of a field by looking at the row number of this field within table `Descriptions4HV`.

[5] Yes, this is not a typo. The object type `Helpvalues` has a parameter `Helpvalues` in its `GetList` BAPI. This parameter has a field called `HELPVALUES`. Try not to be too confused.

Figure 4 The Helpvalues Parameter

Field Name	Data Type	Description
HELPVALUES	Character 255	Contains a variable number of “columns” as one long string.

Figure 5 The Values4Field Parameter

Field Name	Data Type	Description
VALUES	Character 255	Contains one code for the entity.

Figure 6 The Description4HV Parameter

Field Name	Data Type	Description
TABNAME	Character 30	Table name
FIELDNAME	Character 30	Field name
LANGU	Character 1	Language code
POSITION	NumChar 4	Index of the column (not always filled correctly by `Helpvalues.GetList`)
OFFSET	NumChar 6	Offset of this column in the `Helpvalues` table (zero-based)
LENG	NumChar 6	Length of this column in the `Helpvalues` table
FIELDTEXT	Character 60	Descriptive text
REPTEXT	Character 55	Report header text
SCRTEXT_S	Character 10	Short label text
SCRTEXT_M	Character 20	Medium label text
SCRTEXT_L	Character 40	Long label text

`OFFSET` and `LENG` contain the (zero-based) offset and length for a column in table `Helpvalues`.

All the fields containing the string “TEXT” in their names can be utilized to display information about a field in a GUI. If, for example, you want to display all the data returned in table `Helpvalues` in a grid on a form, you can use one set of the text fields as the column headers.

We will discuss which of the fields in this parameter should be used and how in more detail a little later.

Selection4Helpvalues

This parameter allows you to define selection criteria for the data returned in tables `Helpvalues` and `Values4Field`. Other than in the context of Match Code processing (see below) I recommend that you do not use it, unless there could be a vast number of codes for an entity in R/3, which is extremely

Figure 7 The Selection4Helpvalues Parameter

Field Name	Data Type	Description
SELECT_FLD	Character 30	Field name
SIGN	Character 1	"I": Include information based on condition "E": Exclude information based on condition
OPTION	Character 2	Comparison operator
LOW	Character 30	Value for comparison or start of range (OPTION = "BT" or "NB")
HIGH	Character 30	Blank or end of range (OPTION = "BT" or "NB")

unlikely for customizing check tables. Normally, the best strategy is to retrieve all values from SAP once and keep the data buffered. Your client program can then display suitable subsets for the user.

How do you define selection criteria? Let us assume that you want to retrieve the regions for a particular country. The region entity is one of several examples of hierarchical relationships between entities. Each country can have regions. The code of a region (e.g., "CA" for California) is only meaningful if you know the country to which the regions belongs.

Adding a record to table `Selection4Helpvalues` allows you to specify that you only want the regions for the United States. The table contains the fields shown in **Figure 7**.

The field name that you specify in `SELECT_FLD` must be one of the field names returned in table `Description4HV`. So how can you find out which fields are returned for a particular entity and what their names are? The only way to find out is to call `Helpvalues.GetList` with suitable parameters for `ObjName`, `Method`, `Parameter`, and `Field` (if applicable) and then look at the structure in table `Description4HV`. This obviously has to be done at development time (using transaction code SE37, for example, see below) to establish the field names. Those field names will then be used in the runtime calls to define the selection criteria.

The `OPTION` field contains a comparison operator and can contain one of the values shown in **Figure 8**.

To retrieve only the regions for country code "US", you would add a row with the contents shown in **Figure 9** to the `Selection4Helpvalues` table.

Remember, we know that the country code field is called "`LAND1`" from a previous call with the same parameters for `ObjName`, `Method`, `Parameter`, and `Field`.

Multiple conditions can be specified by adding multiple rows to this table. If conditions specify the same field name, a logical *Or* relationship is used between them, if they specify different field names, a logical *And* relationship is used.

DescriptionOnly

This parameter could potentially be useful if you want to build a tool to help a programmer deal with `Helpvalues.GetList`. If you pass "X" as the value, only table `Description4HV` is filled, tables `Helpvalues` and `Values4Field` are returned empty. This would allow you to display just the structure of the information that would be returned in table `Helpvalues` if you left `DescriptionOnly`

Figure 8 The Comparison Operators for Selections

Operator	Fill HIGH Field	Description
EQ		**EQ**ual
NE		**N**ot **E**qual
GT		**G**reater **T**han
LT		**L**ess **T**han
LE		**L**ess or **E**qual
CP		**C**ontaining **P**attern (e.g., "S*")
NP		**N**ot containing **P**attern
BT	Yes	**B**e**T**ween
NB	Yes	**N**ot **B**etween

Figure 9 Retrieving the Regions of the USA

Field Name	Contents
SELECT_FLD	`LAND1`
SIGN	`I`
OPTION	`EQ`
LOW	US
HIGH	

blank, without the overhead of actually retrieving the data for the entity.

Unfortunately, there is a little problem with this parameter. Specifying "`X`" does not always yield the same structure information as leaving the parameter blank. The following problems can occur:

- Different number of fields returned.
- Different names used for a field.
- Different order of fields within table `Description4HV`.
- Incorrect offsets returned.

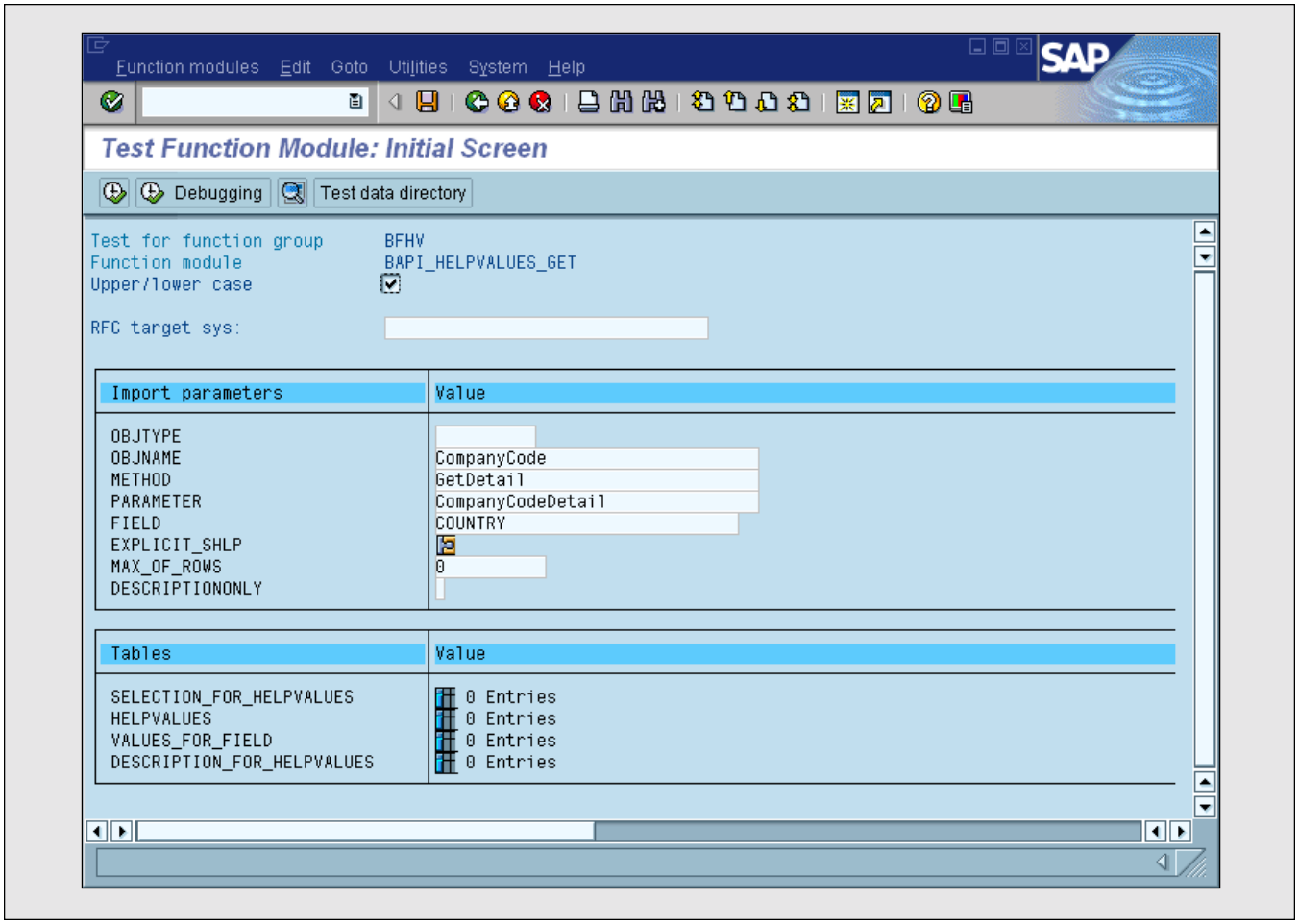
Most of these problems seem to have been fixed in 4.6, but you should still be very careful with this parameter. An alternative is to set `MaxOfRows` to "`1`", which will limit the number of returned rows to one.

Unwanted Features

There are a couple of caveats that you should be aware of before you go off and build your first program using the Helpvalues mechanism.

Sometimes `Helpvalues.GetList` returns useless information. If you want to display the description of the marital status of an employee after calling `Employee.GetList`, for example, you will find that `Helpvalues.GetList` returns too much information for the `MAR_STATUS` field in the `PersonalData` table. You get all the marital status codes, albeit not once, but once for each language in the system. There is no language code field in the returned information, so you have no chance to identify the entry that contains the text in your language. Fortunately, in most cases `Helpvalues.GetList` works just fine, and the few problems will hopefully be resolved by SAP in the near future!

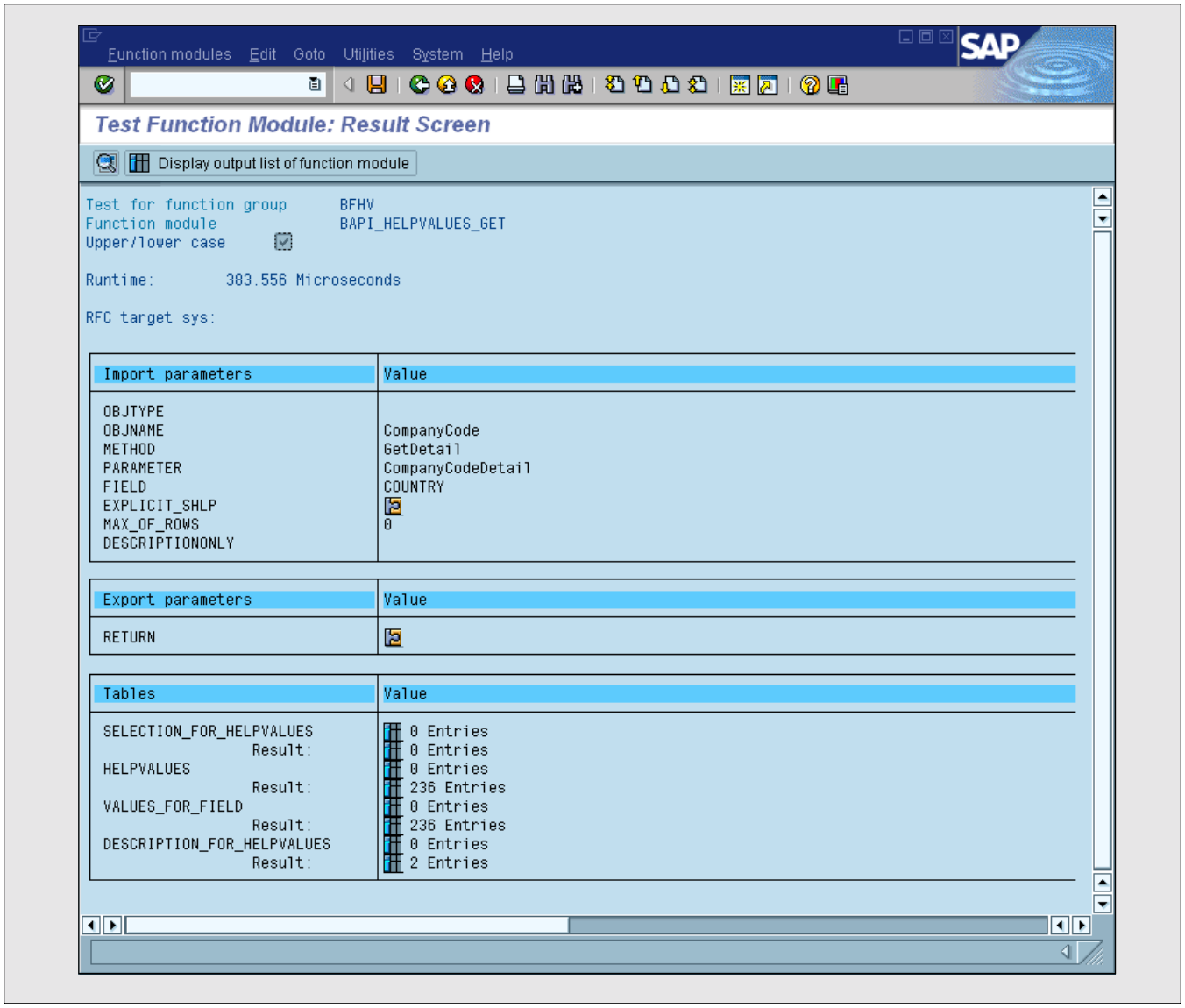
Figure 10 About to Retrieve All Countries



Since the structure of the data to be returned is determined dynamically when you call `Helpvalues.GetList`, the only way to find out what kind of information you get is to actually call the BAPI for the entity you are interested in. One caveat applies here: it is possible that the structure of the data that you receive changes between releases. This will certainly not happen very often, but you should be aware of this possibility. In most cases, the data returned by the BAPI consists of a column for the code and one for the descriptive text, respectively, but there are exceptions. And if more than one column is returned, the descriptive text is not always contained in the second column. Different releases of SAP return different columns when you retrieve all country codes, for example. We will discuss recommendations for a component that provides release independence later.

At this point you might say: Are the BAPIs not supposed to be upward-compatible? A lawyer would argue that `Helpvalues.GetList` is upward-compatible since its parameters have not changed incompatibly. Just the data returned has changed and the data is not covered by any compatibility guarantee. This is technically true, but not very satisfactory since part of the data that we get is the structure information required to use the other data in a meaningful way. Again, only a component that encapsulates all access to `Helpvalues.GetList` is a practical answer to this issue.

Figure 11 *All Countries Retrieved*



Some Real-World Examples

If this is your first exposure to the Helpvalues phenomenon and you still feel a little confused, I cannot blame you. This stuff is not exactly trivial. Therefore, I will now take you through two examples of how this works in the real world. I will retrieve a list of all countries, followed by a list of all regions. The `Helpvalues.GetList` BAPI will be invoked inside SAP's Function Builder test environment (transaction code SE37).

Figure 10 shows one set of parameters that identify the country entity. As we saw earlier, any combination of parameters identifying that entity could have been used instead.

After pressing the execute button (the icon that looks like a clock with a check mark) you will get the results shown in **Figure 11**. Table `Description4HV` contains two rows now, and tables `Helpvalues` and `Values4Field` 236 rows each.

✔ Tip

If you want to try out these examples yourself, there are two things to note.

Do not forget to check the "Upper/lower case" field because what you type will otherwise all be translated to upper case, which will not work since the object name parameter is case-sensitive.

*The screen shots of the examples show slightly different parameter names than the ones we discussed above. This is due to the fact that the BAPI parameter names can be different from the parameter names of the underlying function module (*BAPI_HELPVALUES_GET*). Fortunately, they are similar enough that you will easily make the necessary mental connections.*

Let us first inspect the metadata in table `Description4HV` (see **Figure 12**). In 4.6, this table contains two rows, in previous releases there were sometimes three. The offset and length information for each "column" will be vital later to split the string returned in table `Helpvalues` into the proper columns. Note that the `POSITION` field for both "columns" contains "`0001`", which should not bother us since we can safely ignore this field. The LANGU field is displayed as having two characters, while above we indicated that the language code was just one byte long. How so? What we see here is a conversion exit at work. SAP uses one-byte language codes internally, but converts them to their two-byte ISO equivalent for display in SAPGUI. When we call the BAPI externally, the language code will just be one byte, as defined in Figure 6.

Figure 13 shows the first few rows of table `Values4Field`. As you can see, only the country codes are available in this table, which makes it suitable for a quick validation of a country code since we do not have to split strings to get at the codes.

In this figure, you can also see the 255-byte string containing all the data described by the metadata information in Figure 12. This string now has to be split using the offset and length information for each "column".

Figure 12 The Metadata for the Countries

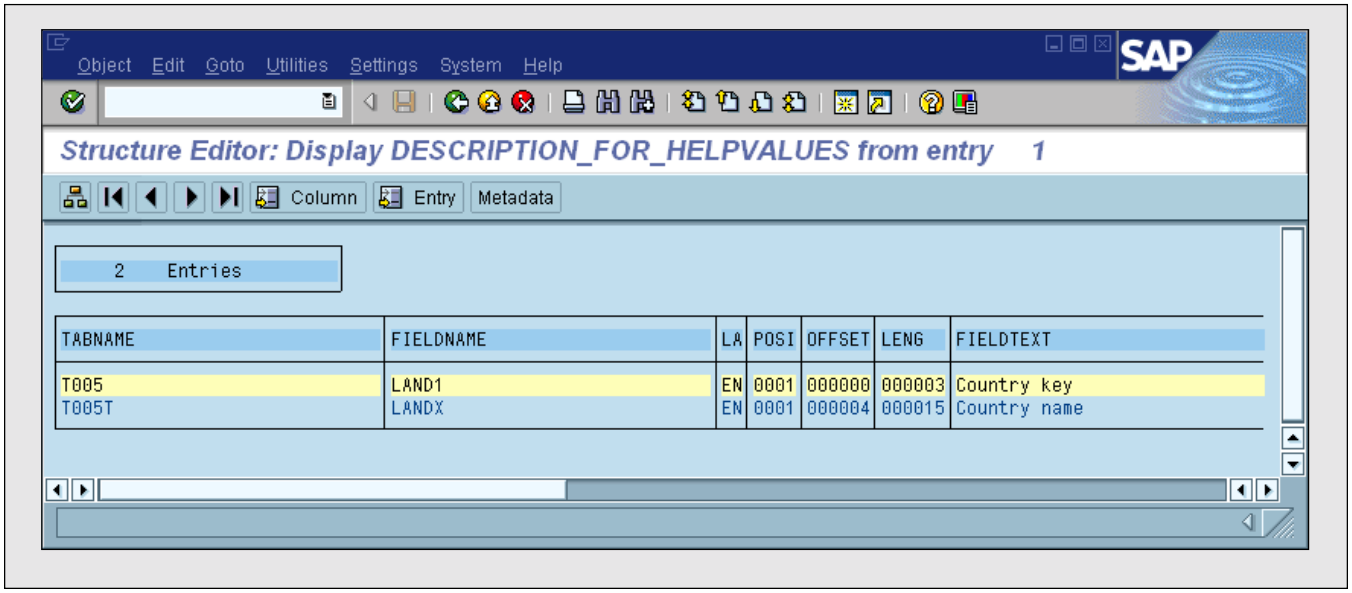


Figure 13 All Country Codes and All Country Data

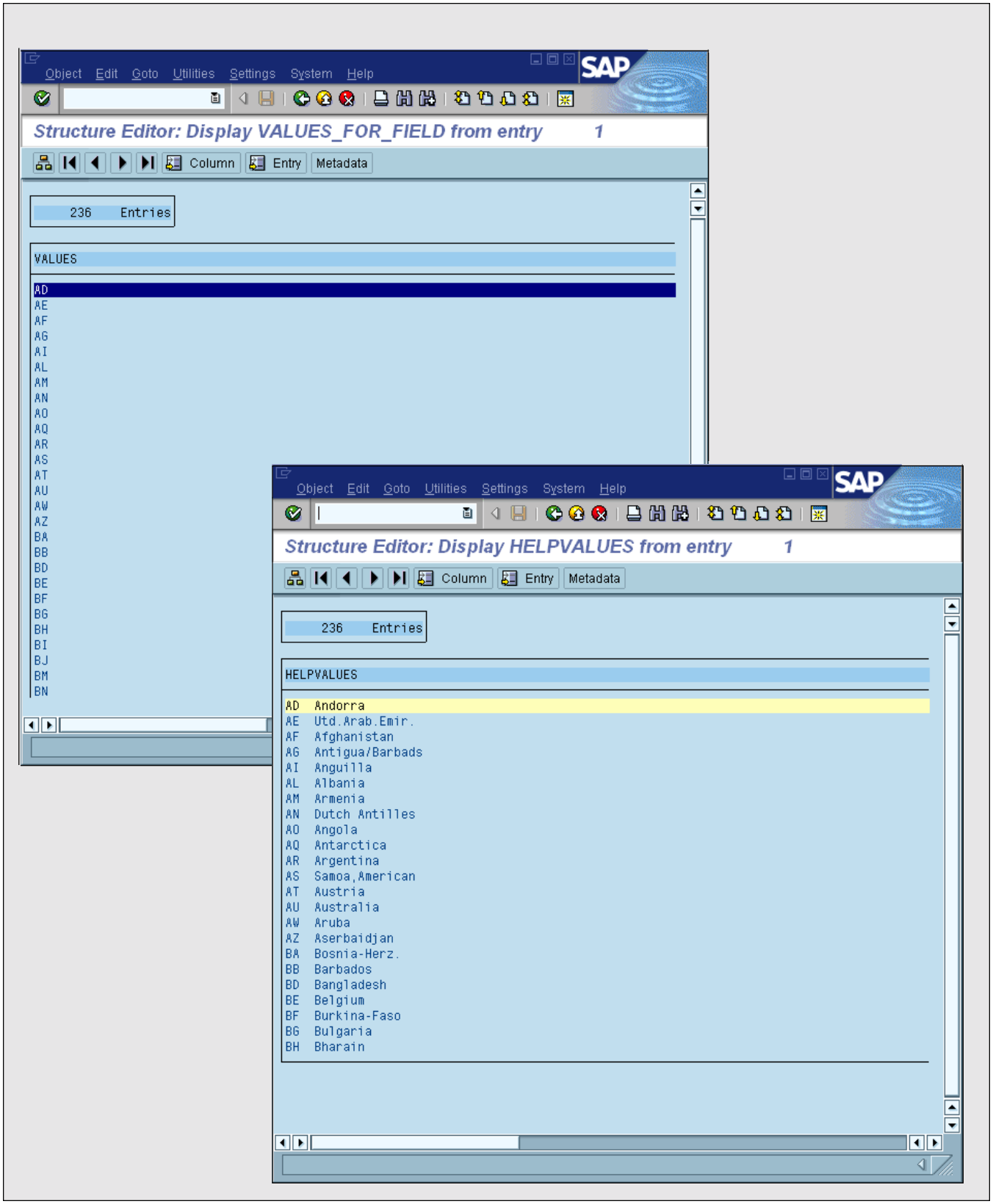
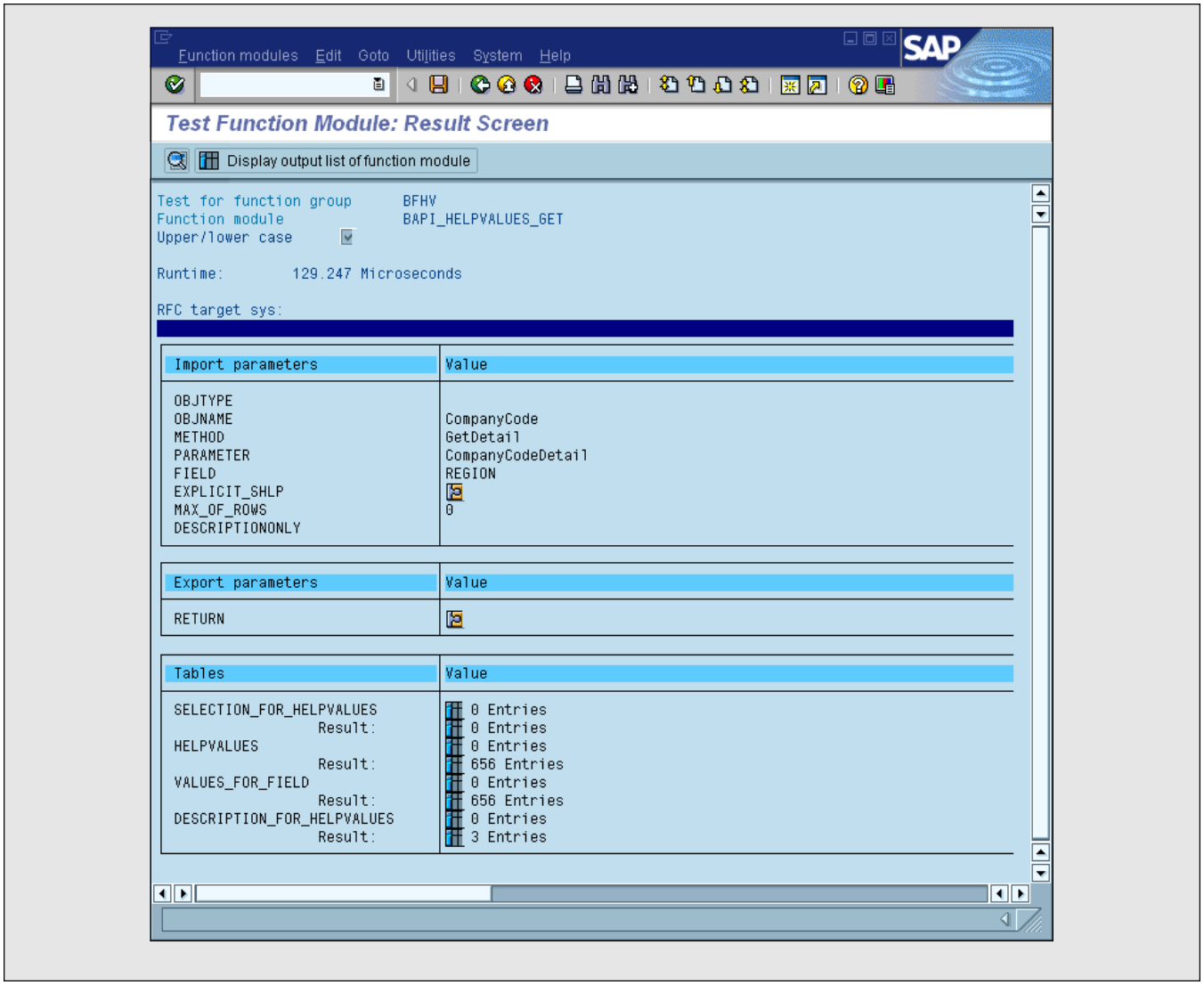


Figure 14 *All Regions Retrieved*



As a human being, you are able to easily discover the columns without any metadata. This is due to the fact that humans are good at detecting patterns. In addition, the BAPI helps a little bit by inserting blanks between the "columns". Does that mean that we do not need the metadata after all and can simply split the string after each blank? This would not be a good idea. Firstly, sometimes we will get more than two columns (see the region example below). The description "column" might be between two other "columns" and might contain blanks itself, which would yield invalid data for the subsequent "columns". Secondly, remember that the sequence of "columns" is not necessarily the same across releases. The only way to protect ourselves against surprises is to hard-code the field names in our Helpvalues component (these field names are extremely unlikely to ever change) and use the associated metadata.

In the second example, we want to retrieve all regions from SAP. The result of the BAPI call is shown in **Figure 14**. This time, we have three "columns" and 656 codes.

In the metadata (**Figure 15**) we see the three "columns" region code, country code, and region

Figure 15 The Metadata for the Regions

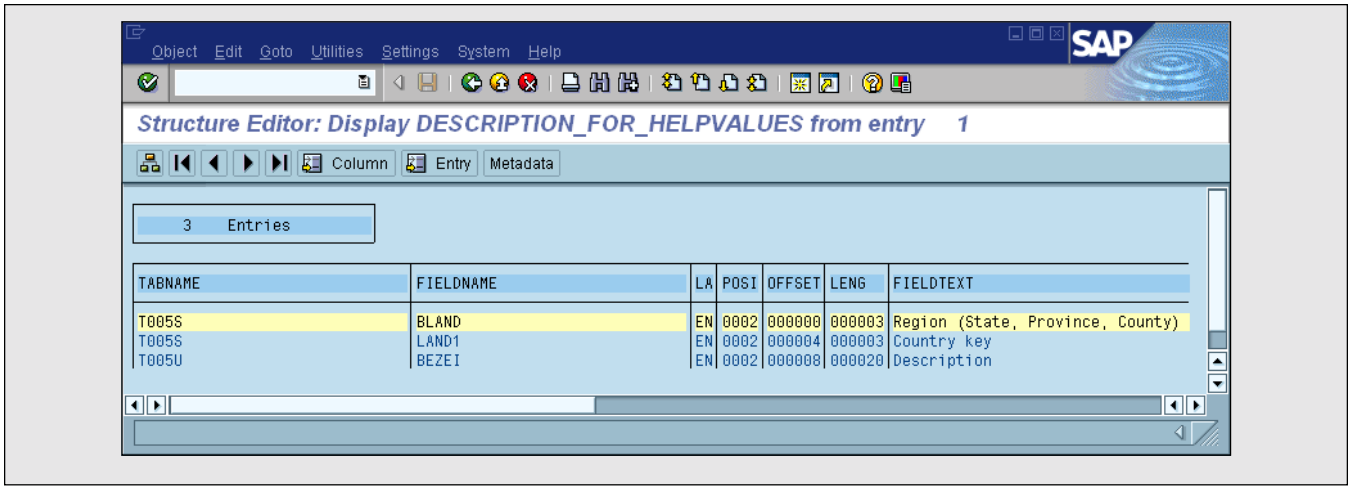
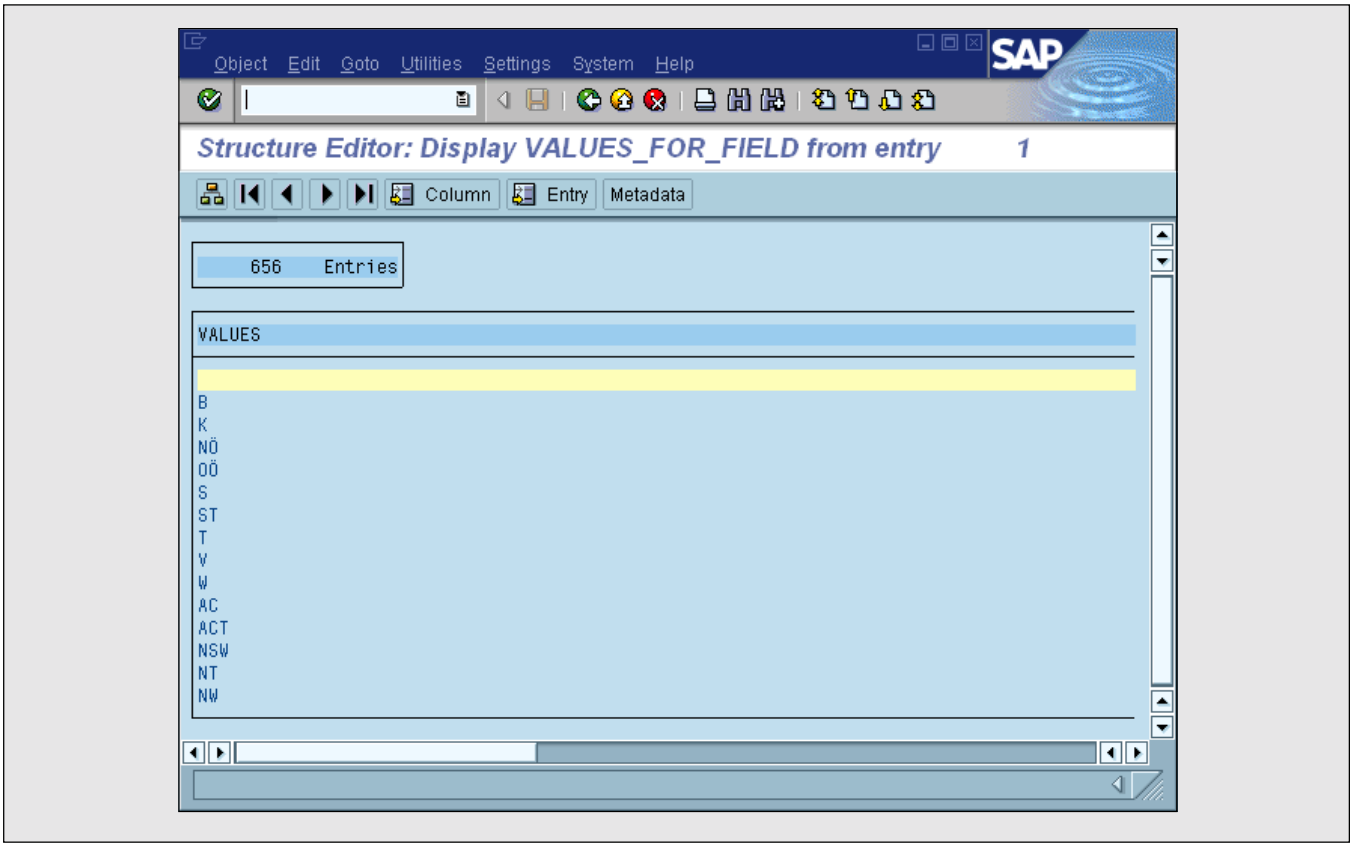


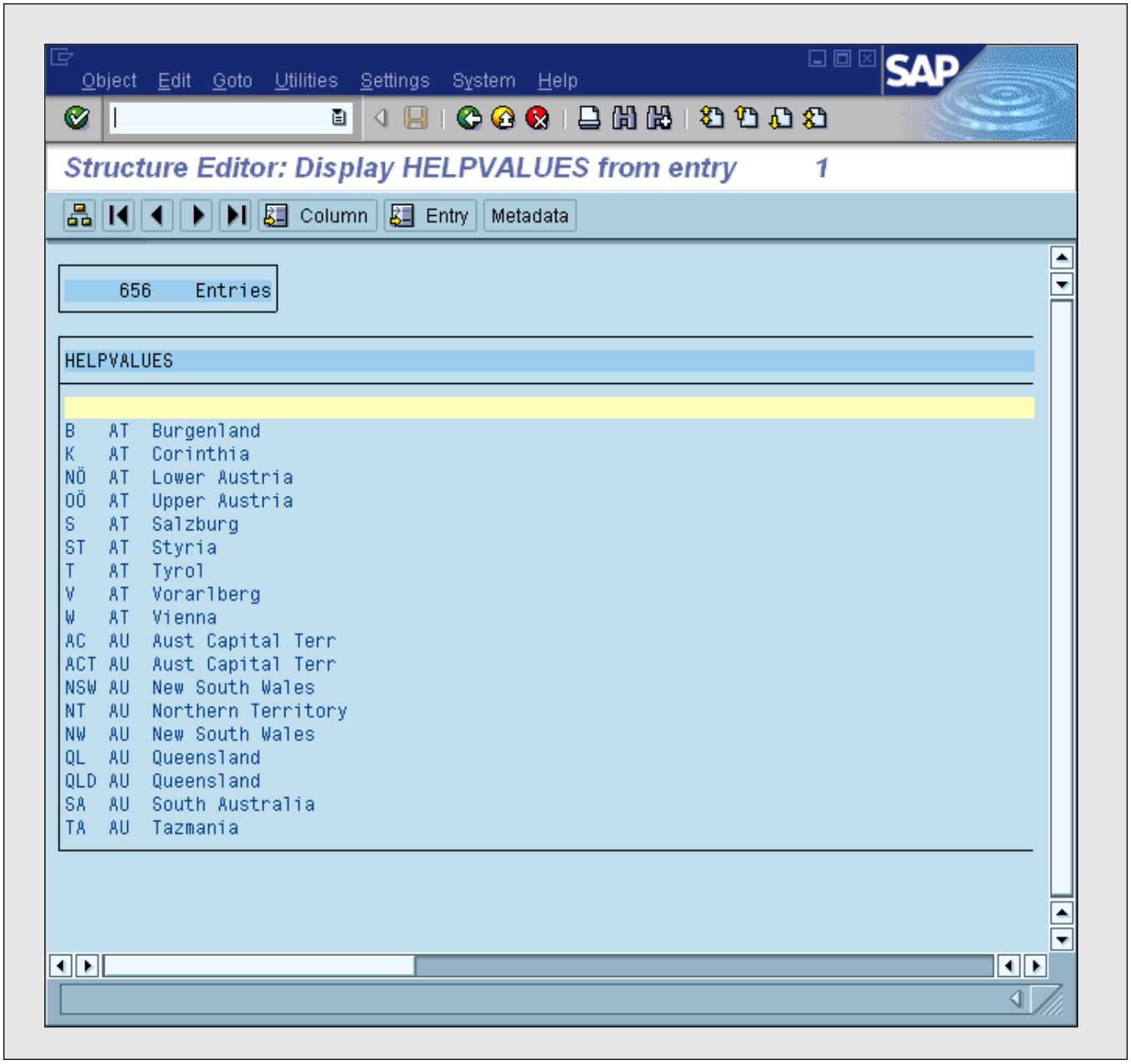
Figure 16 All Region Codes



name (description). This time, the `POSITION` field for all three contains "`0002`", but again we will simply ignore this bug. The sequence of these "columns" (as far as I can remember) was different in previous releases. This emphasizes the importance of a component coping with these details.

The `Values4Field` table (**Figure 16**) contains

Figure 17 All Region Data



all codes. But here, this information is totally useless since we do not know which country a given region code belongs to. Also note that the table starts with a totally superfluous blank row. Since different countries can have regions with the same codes, the values in this table are not unique either.

Figure 17 contains the long string with all relevant data, to be interpreted according to the metadata shown in Figure 15. Note that not all countries have regions, but all regions belong to a country. And as in table `Values4Field`, there is an empty row at the beginning that must be ignored.

A Helpvalues Component

Before we finally move on to the Match Code mechanism, I want to share some ideas about the componentization of access to the Helpvalues mechanism. Since I have built three versions of such a component for Java and Visual Basic developers, I am confident that you can benefit from these ideas.

The most important issue is to have such a component in the first place and use it in all your applications. This provides several benefits:

- ✔ Only very few people will have to study all details of the Helpvalues mechanism.
- ✔ Whenever a new release of SAP comes out, you will only have to test one component, instead of all your applications.
- ✔ Buffering (caching) the retrieved data for each entity will improve performance since we will only access SAP once for each entity.
- ✔ Building an application will go much faster since all developers can use an easy-to-use (hopefully) component.

Now for some details:

- ✔ The only information you should hard-code in your component are the field names that are returned for each entity. This allows you to use the same code unchanged in different releases.
- ✔ In case new "columns" appear in new releases or old ones disappear, the component needs to know about this and check for the release of SAP it is currently talking to in order to take appropriate action.
- ✔ Hierarchies need to be supported properly. That means that the component understands the existing hierarchies and offers the required properties. The *Country* class of your component should have a *Regions* property, for example, that allows access to a collection of all *Region* objects. The region information should only be retrieved from SAP when the client program is accessing it for the first time (known as "just-in-time data retrieval" or "lazy initialization").
- ✔ Also, the component should be aware of the data conversion issues that I briefly mentioned in conjunction with the magical language code translation in SAPGUI. It would be best if all entity classes like *Country*, *Language*, and so on, had an *InternalCode* and an *ExternalCode* property so that the client program can easily access the appropriate information. The component would call the conversion BAPIs of object type BapiService in order to do the necessary conversions (again just-in-time and buffered).

BAPI Match Code Processing

After this short prelude (the article so far), we are now ready to tackle the Match Code support that was added in 4.6. The first issue will be security. Earlier, we discussed the reasons why up to 4.5 there was support only for customizing check tables. How did SAP make sure that only authorized users can access critical information like customer, employee, and product data? Well, they added a new table (BAPIF4T) that defines which authorization checking module should be called for which entity.

Figure 18 *Structure of Table BAPIF4T*

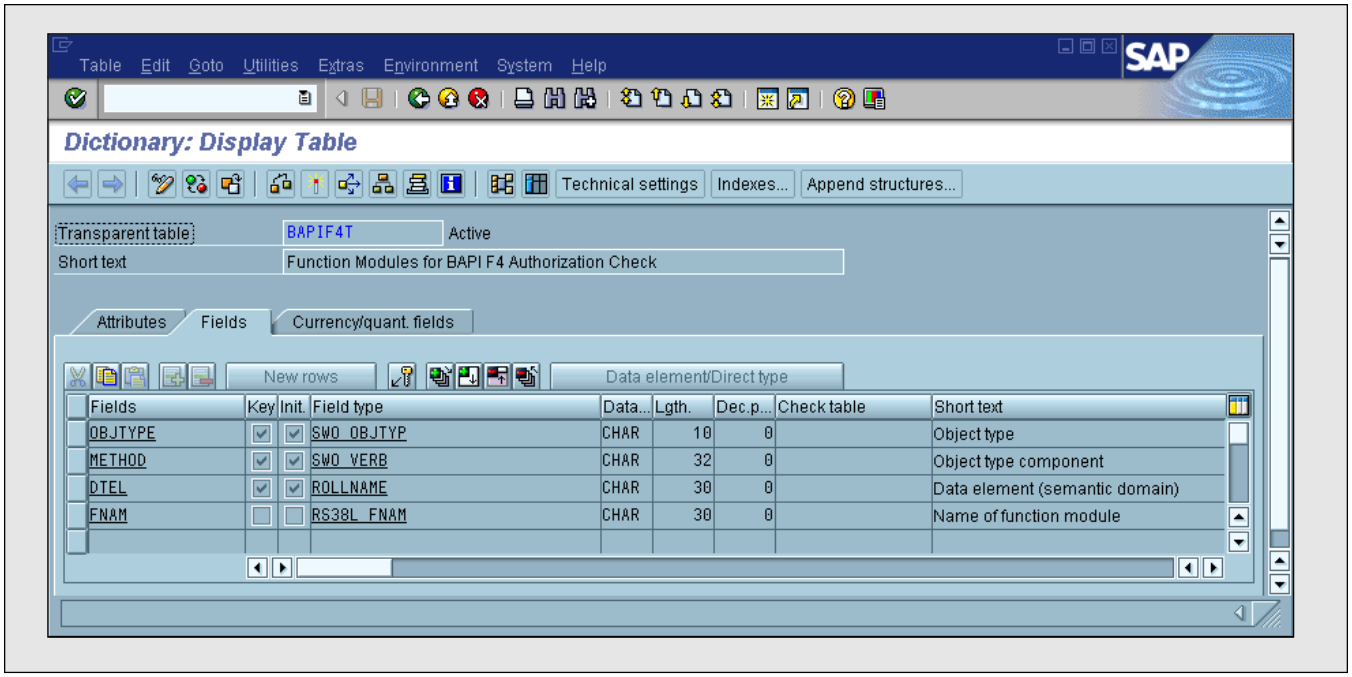


Figure 19 *Contents of Table BAPIF4T*

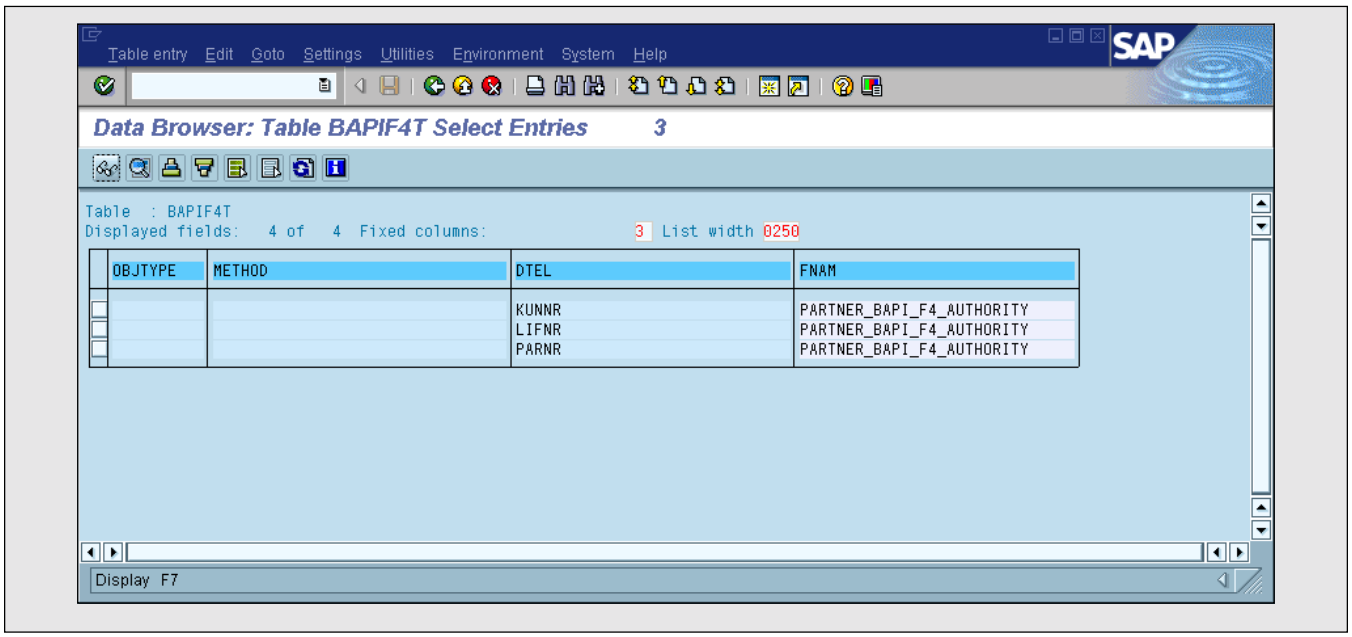


Figure 18 shows the columns of this table, and **Figure 19** the contents as delivered by SAP.

As you can see, only three entities (customers, suppliers, and partners) are supported in the

Figure 20 Helpvalues.GetSearchhelp Parameters

Parameter Name	Data Type	Import/ Export	Mandatory	Description
Objtype	Character 10	Import		Internal name of object type (e.g., "BUS2032")
Objname	Character 32	Import		Official name of object type (e.g., "SalesOrder")
Method	Character 32	Import	Yes	Name of the BAPI (e.g., "CreateFromDat1")
Parameter	Character 32	Import	Yes	Name of the parameter (e.g., "OrderPartners")
Field	Character 30	Import		Field name in structure or table parameter (e.g., "PARTN_NUMB")
Return	Structure BAPIRETURN	Export		Return code information
ShlpForHelpvaluesGet	Table BAPISHLP	Imp/Exp	Yes	List of all Search Helps

standard system. The authorization check for all three happens in a function module called PARTNER_BAPI_F4_AUTHORITY, which contains the appropriate ABAP code to ensure that only users with proper authorization can use the BAPI Match Code support for these entities.

You can add support for new entities (like products) by executing these steps:

- Add a new row to BAPIF4T specifying the data element name for the new entity (MATNR in the product example). The data element name can be found in the Data Dictionary. Specify the name of a new function module that will contain the authorization check code.

- Add a function module with the name used in the previous step. The best way to accomplish this is to copy PARTNER_BAPI_F4_AUTHORITY and assign it a new name. That ensures that the function module already has the proper interface.

- In the new function module, execute the authority checks for the new entity. You will need to know ABAP and also the authorization object(s) that should be used for the new entity. Ask an ABAP application developer for help unless you are an ABAP wizard yourself.

The Helpvalues.GetSearchhelp BAPI

One entity can have multiple Match Codes, therefore the first step of using the Match Code support is to retrieve a list of the Match Codes for the entity in which we are interested. The Helpvalues.GetSearchhelp BAPI allows us to do this. Its parameters are shown in **Figure 20**.

Figure 21 The ShlpForHelpvaluesGet Parameter

Field Name	Data Type	Description
SHLPNAME	Character 30	Name of Search Help
SHLPTYPE	Character 2	Type of Search Help
TITLE	Character 60	Descriptive text
REPTEXT	Character 55	Report header text

Figure 22 Retrieving the Match Codes for the Customer Entity

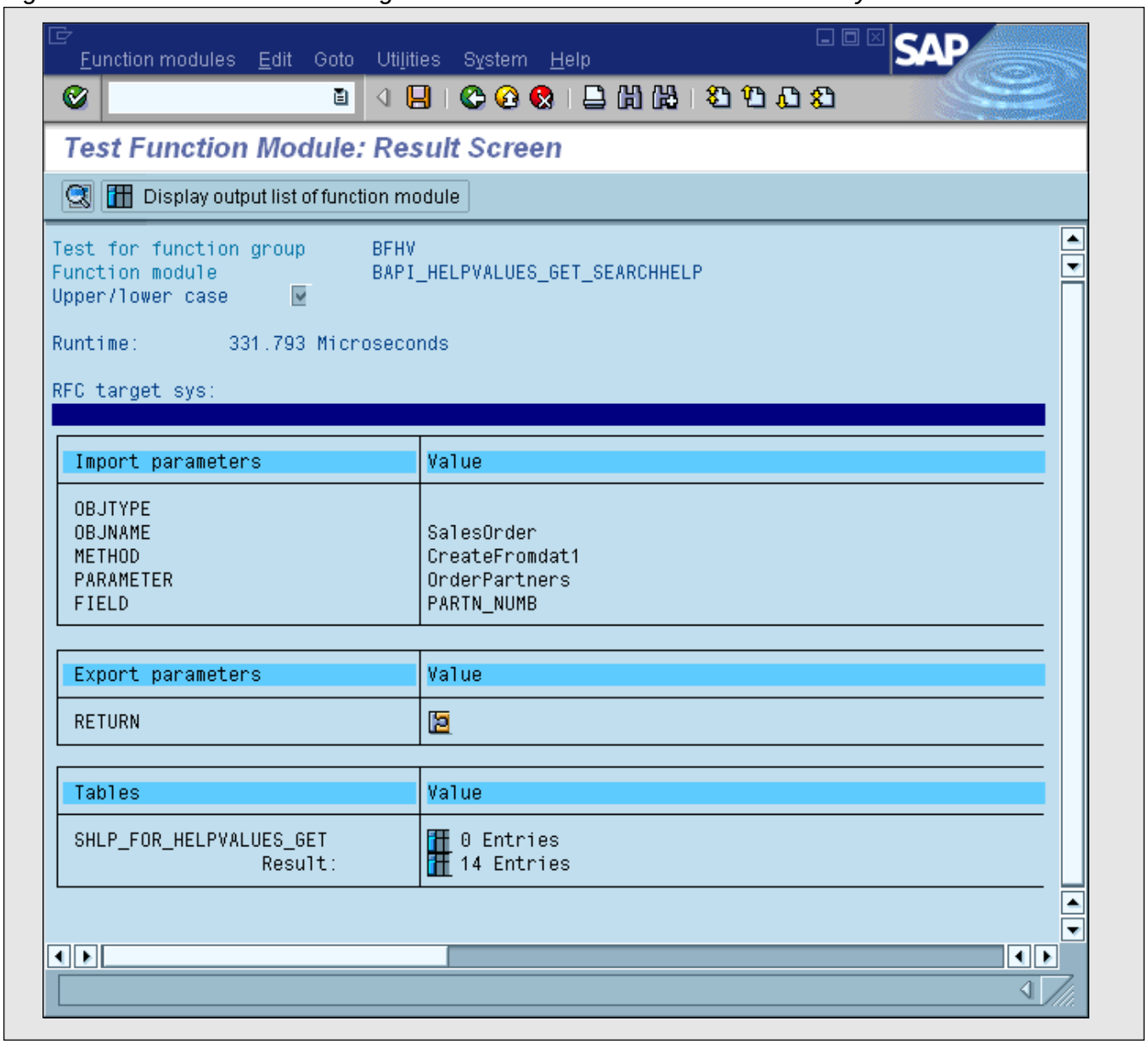
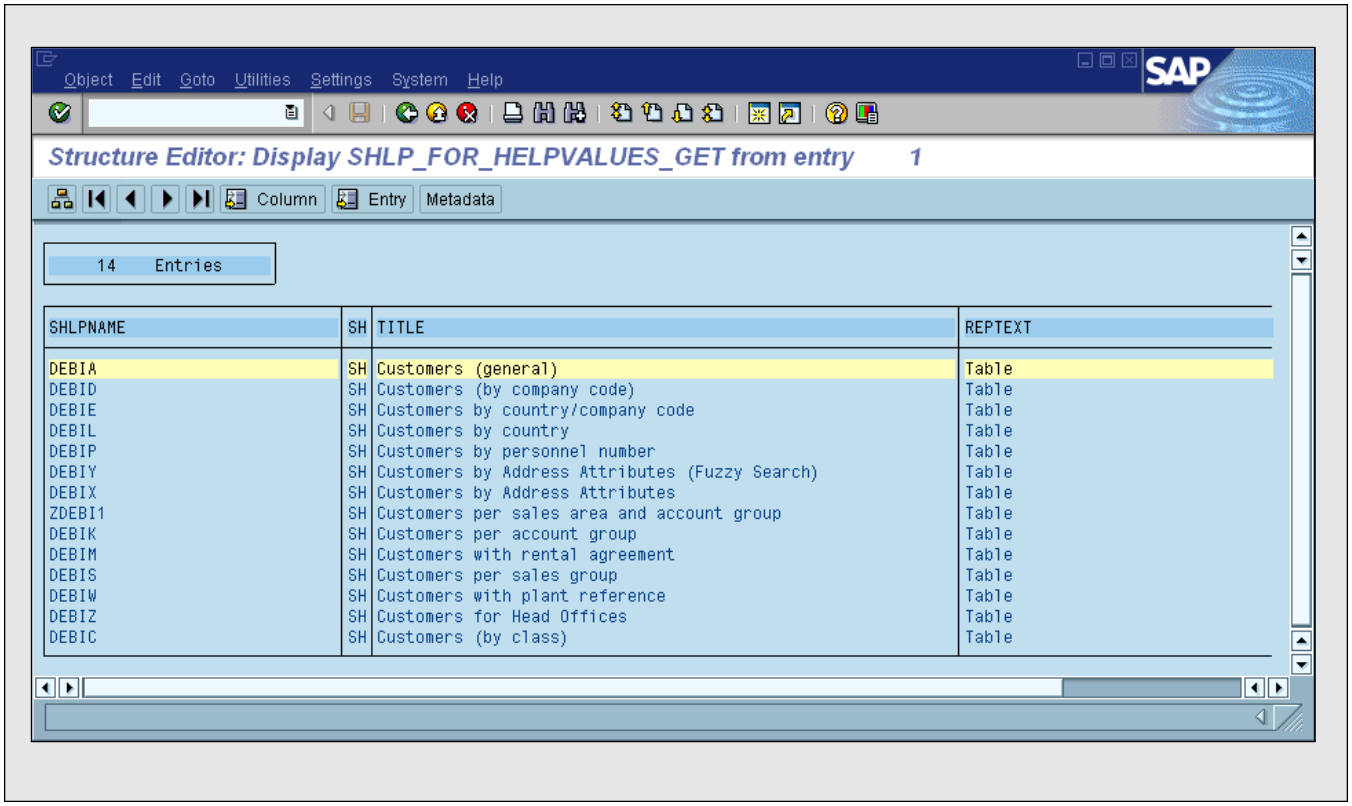


Figure 23 All Match Codes for the Customer Entity



Parameters `Objtype`, `Objname`, `Method`, `Parameter`, `Field`, and `Return` are used the same way as in `Helpvalues.GetList`. (The capitalization of `Objtype` and `Objname` is slightly different but they serve the same function as their cousins.)

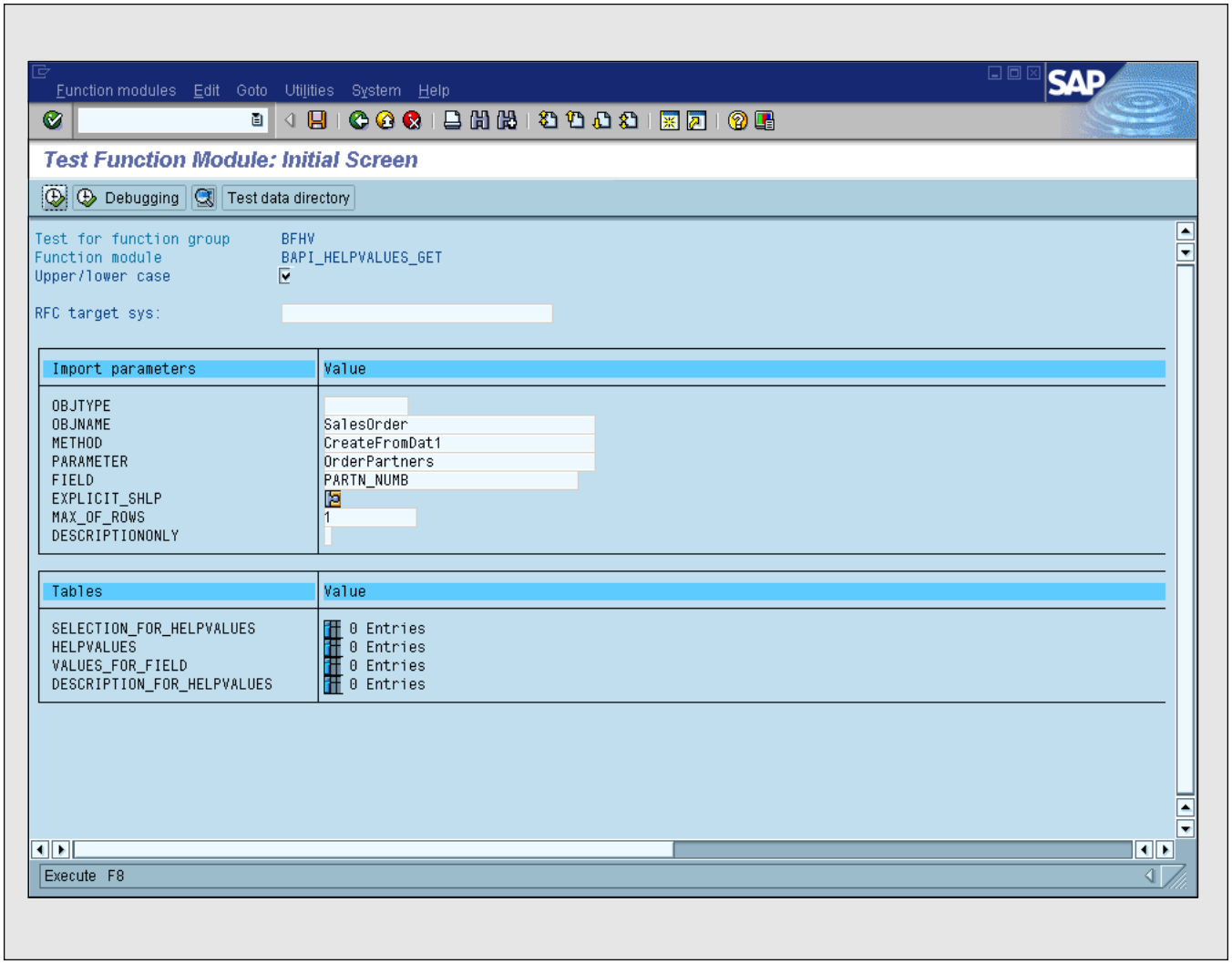
Parameter `ShlpForHelpvaluesGet` contains a list of all Search Helps for the selected entity. **Figure 21** shows the structure of this parameter.

Let us look at a practical example to see what kind of information we will receive in this table. Assume that we need Match Code support for the customer entity in our application. Then we will call `Helpvalues.GetSearchhelp` with a set of parameters identifying the customer entity, for example those shown in **Figure 22**.

Parameter `ShlpForHelpvaluesGet` now contains the data displayed in **Figure 23**. The first two columns (Search Help name and type) will be used in subsequent calls to the `Helpvalues.GetList` BAPI. The contents of the `TITLE` column will be useful to display a list of all available Match Codes so that the user can select one. `REPTEXT` does not seem particularly useful.

Once the user has decided which Match Code he wants to use, we need to find out the columns of this Match Code. This is where we return to `Helpvalues.GetList` and finally make use of its new `ExplicitShlp` parameter. This parameter contains the same fields as parameter `ShlpForHelpvaluesGet` (shown in Figure 20), but the `ExplicitShlp` parameter is a structure,

Figure 24 Retrieving the Structure of a Match Code



not a table. **Figure 24** shows that the parameters to identify the entity are the same as in our call to `Helpvalues.GetSearchhelp`. This should not come as a surprise since we are still working with the same entity (customer). The reason that we set `MaxOfRows` to "`1`" is that we are not ready for the actual data retrieval yet. We just want to find out the structure of the relevant Match Code, not retrieve all customers in the system.

Before actually calling the BAPI, we have to fill in the `ExplicitShlp`, as shown in **Figure 25**. In this example, I assume that the user has selected the `DEBIS` Match Code, otherwise the name obviously would have to be different.

Figure 26 is a screen shot right after the execution of the BAPI. As you can see, the `Description4HV` table contains 10 rows. The number could have been different for other Match Codes.

Figure 25 Specifying the Selected Match Code

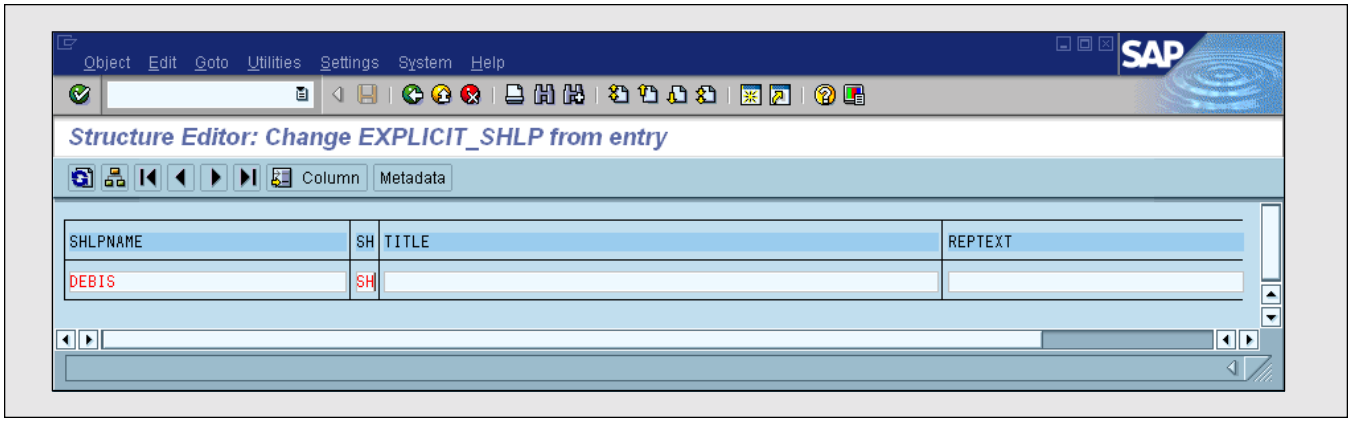


Figure 26 After the Structure Retrieval

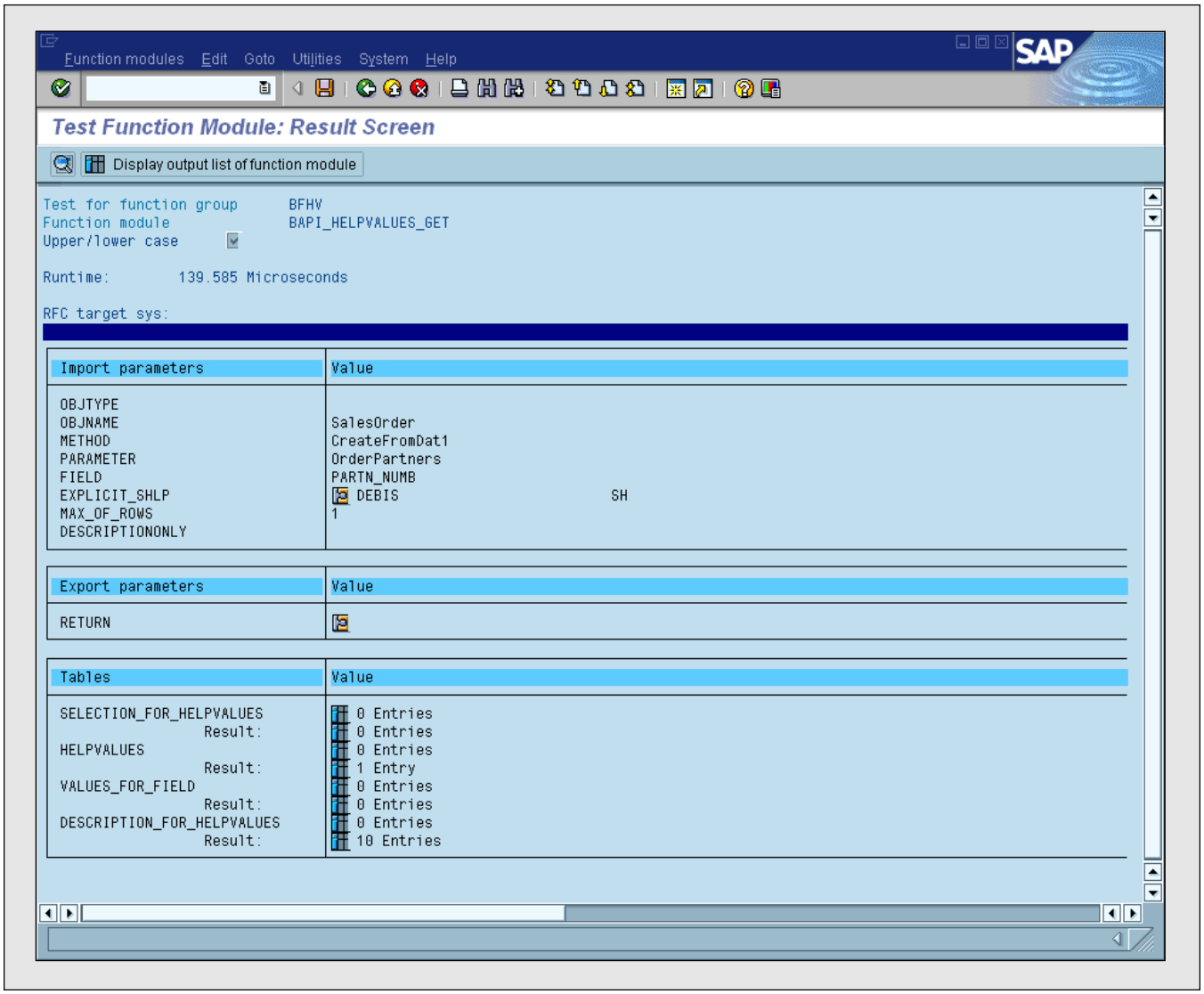


Figure 27 The Columns of the DEBIS Match Code (Part 1)

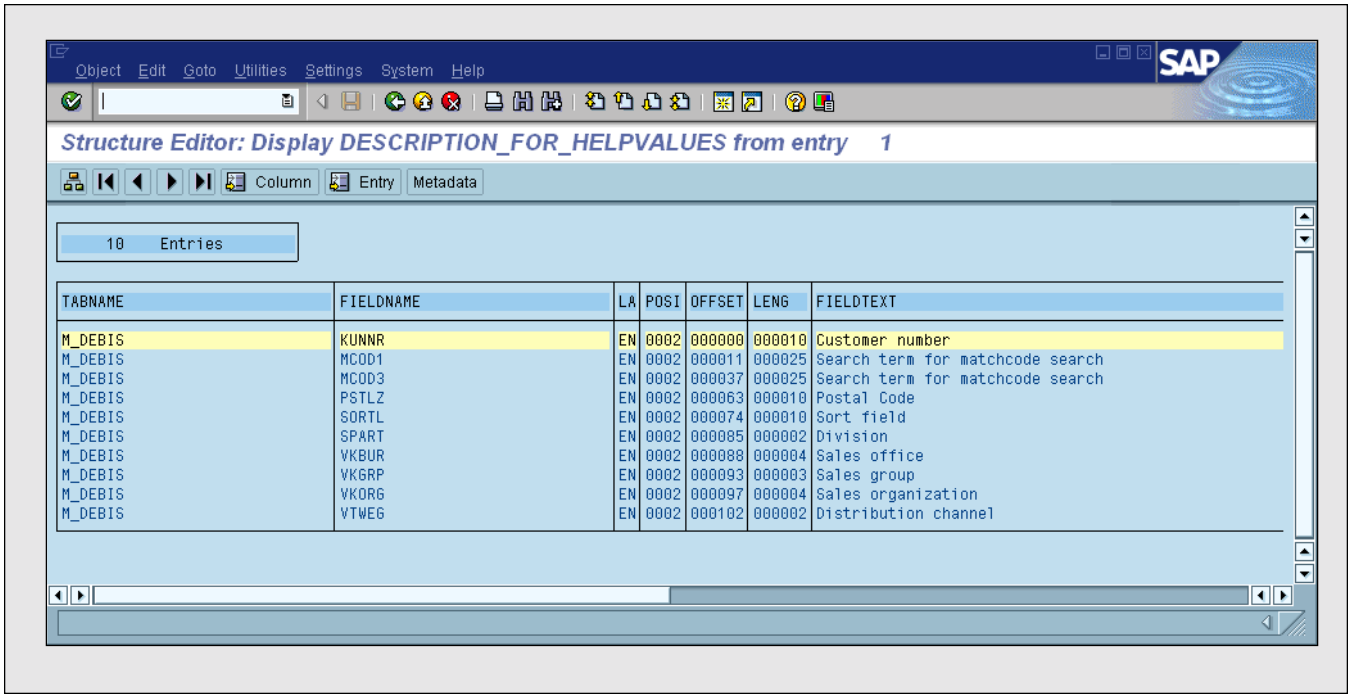
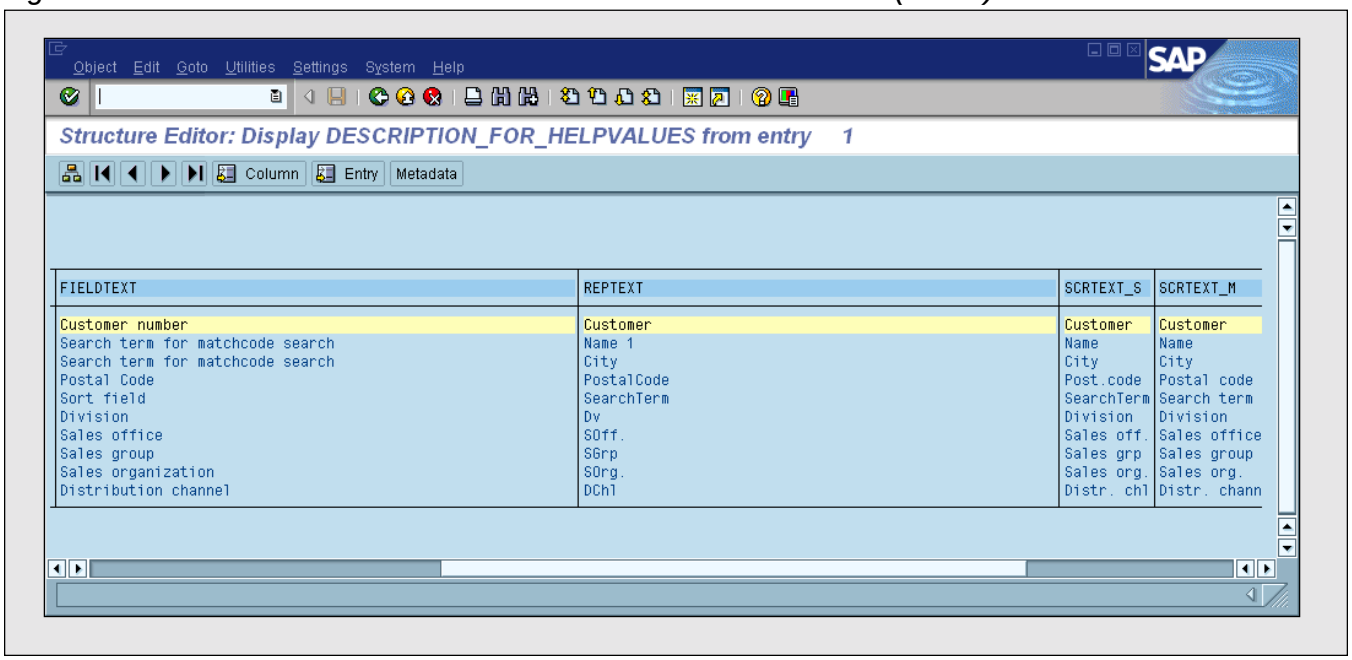
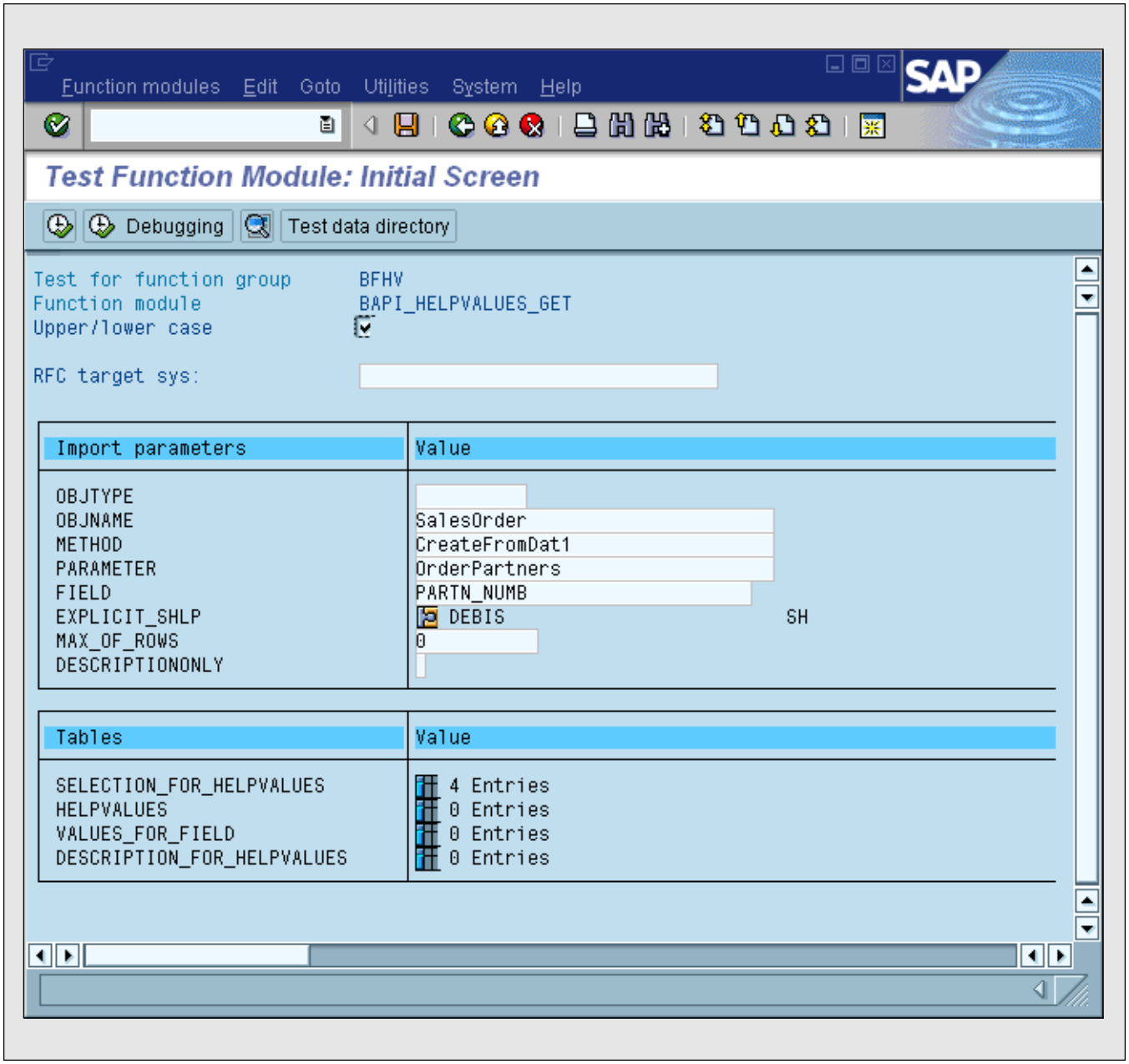


Figure 28 The Columns of the DEBIS Match Code (Part 2)



Next we need to utilize the structure of the Match Code from parameter `Description4HV`, as shown in **Figure 27** and **Figure 28** (unfortunately there is more information than can be shown in one

Figure 29 *Ready to Search for Customers*



screen shot, Figure 28 contains what you see when you scroll right in Figure 27). The metadata in this parameter now allows us to assemble a nice selection screen for the user that allows him to specify the criteria based on which he wants to search for customers (screen shots of an actual client program will follow later).

Once the user has entered the selection criteria, we get ready to call `Helpvalues.GetList` one more time (**Figure 29**). We have changed MaxOfRows back to zero because now we want all customers matching the user's selection criteria. As you can see, we have added four rows to parameter Selection4Helpvalues.

Figure 30 The Selection Criteria

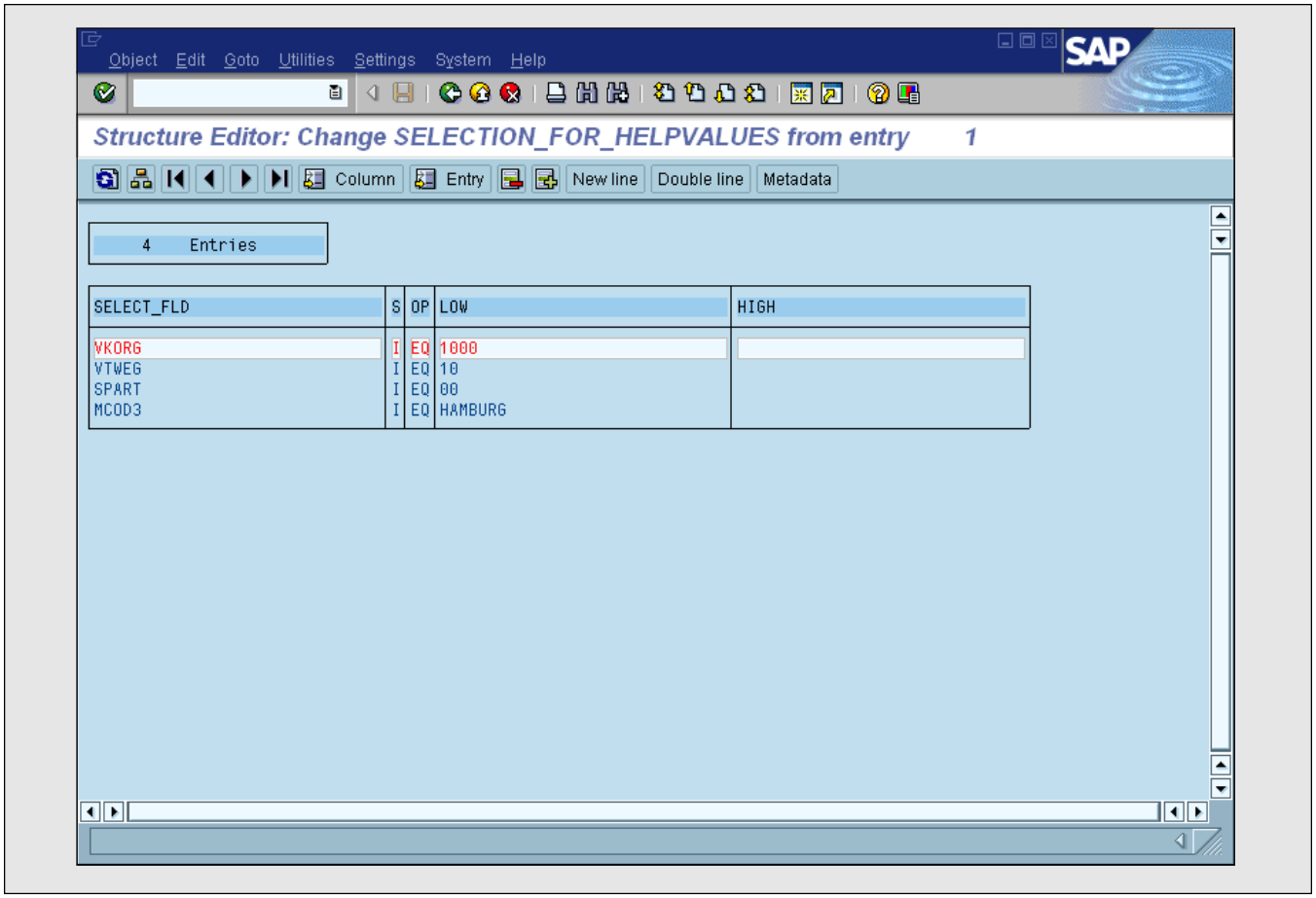


Figure 30 shows the criteria we entered. Obviously, our user was only interested in customers from Hamburg defined for a specific division of a specific distribution channel in a specific sales organization (users can be very picky!).

After the call, we expect table `Helpvalues` to contain the proper customer information (see **Figure 31**). Our last task will be to split the data in this table apart using the metadata for this Match Code (see Figure 27) and to display the result set so that the user can select one of the customers.

Believe it or not, we are done. We have managed the steps required to support Match Codes in our BAPI-enabled applications. Obviously, all those steps need to be encapsulated, so that we have a nice, reusable, high-performance component that can handle all non-visual aspects of the Match Code processing. In addition, we might want to build some visual controls that facilitate the development of a GUI with Match Code support.

A Little Sample Program

All that is left to do for me is to show you a real-world sample GUI on behalf of which we could have executed the steps just discussed. In **Figure 32**, the user can press the binocular icon to access the Match Code support offered by the sample program.

In response, the pop-up window depicted in

Figure 31 The Customers Matching the Criteria

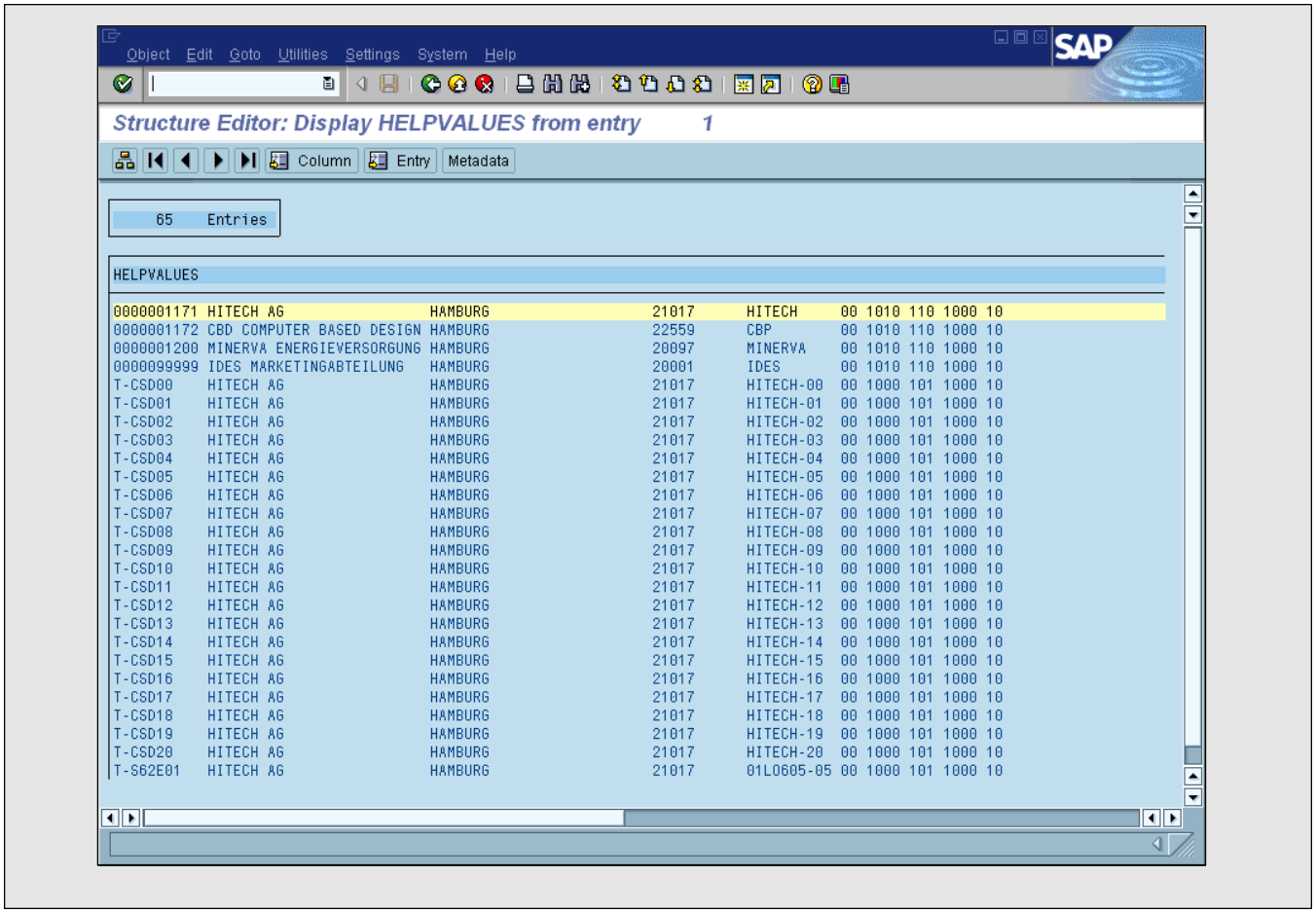


Figure 32 The Sample Program

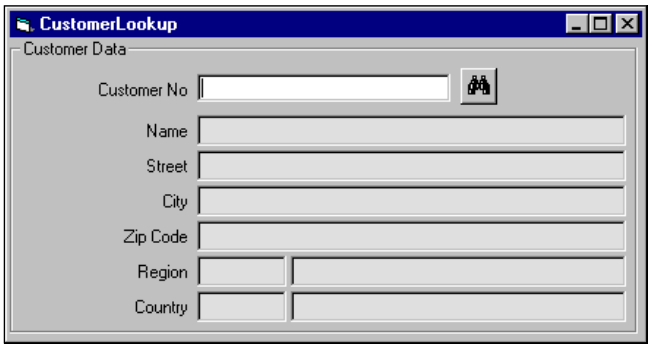


Figure 33 Entering Selection Criteria

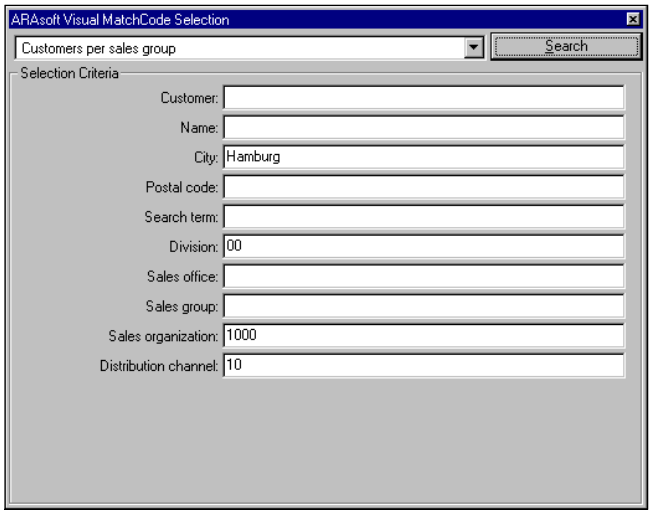
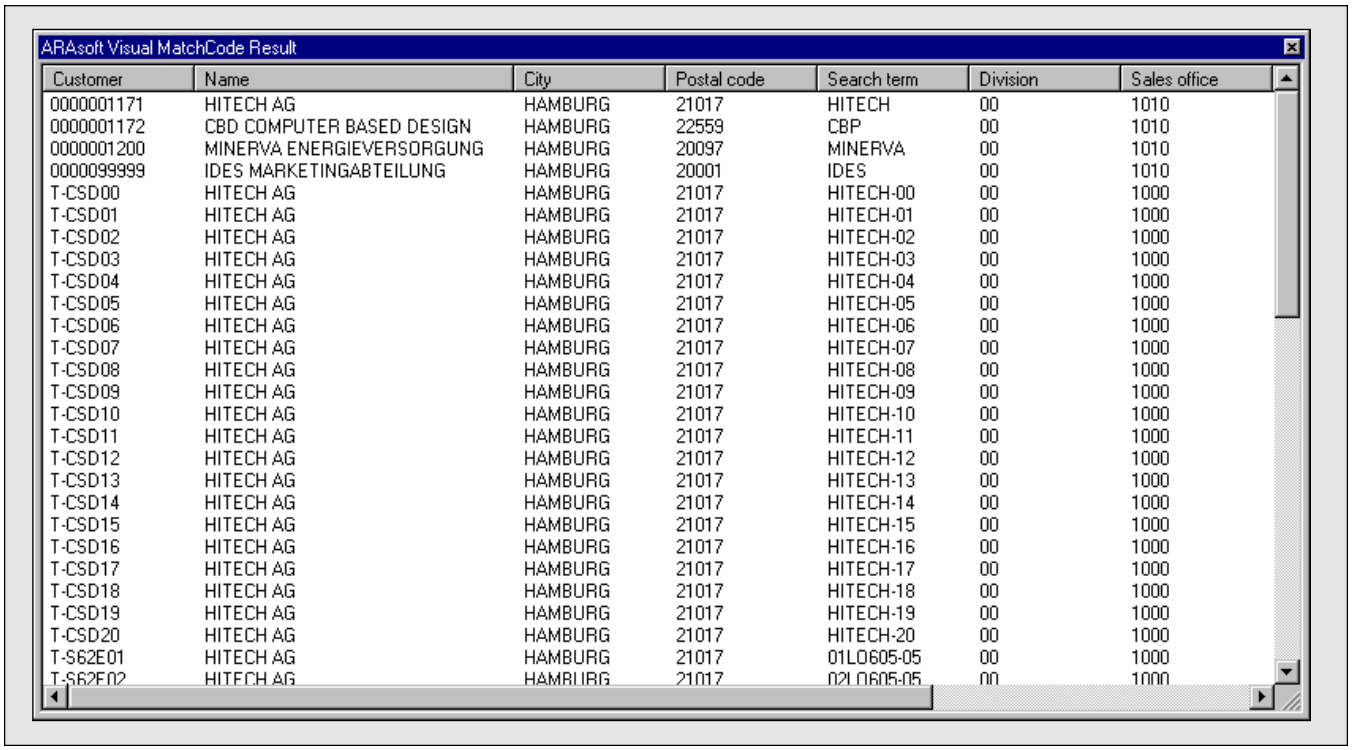


Figure 33 is displayed by the application. There is a dropdown combo box that contains the descriptions of all the Match Codes for the customer entity and a frame with the selection criteria possible for a specific Match Code. This second part is populated

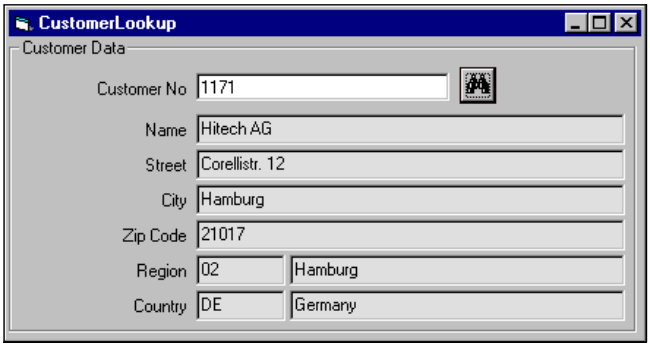
Figure 34 The Customers Matching the Criteria



Customer	Name	City	Postal code	Search term	Division	Sales office
0000001171	HITECH AG	HAMBURG	21017	HITECH	00	1010
0000001172	CBD COMPUTER BASED DESIGN	HAMBURG	22559	CBP	00	1010
0000001200	MINERVA ENERGIEVERSORGUNG	HAMBURG	20097	MINERVA	00	1010
0000099999	IDES MARKETINGABTEILUNG	HAMBURG	20001	IDES	00	1010
T-CSD00	HITECH AG	HAMBURG	21017	HITECH-00	00	1000
T-CSD01	HITECH AG	HAMBURG	21017	HITECH-01	00	1000
T-CSD02	HITECH AG	HAMBURG	21017	HITECH-02	00	1000
T-CSD03	HITECH AG	HAMBURG	21017	HITECH-03	00	1000
T-CSD04	HITECH AG	HAMBURG	21017	HITECH-04	00	1000
T-CSD05	HITECH AG	HAMBURG	21017	HITECH-05	00	1000
T-CSD06	HITECH AG	HAMBURG	21017	HITECH-06	00	1000
T-CSD07	HITECH AG	HAMBURG	21017	HITECH-07	00	1000
T-CSD08	HITECH AG	HAMBURG	21017	HITECH-08	00	1000
T-CSD09	HITECH AG	HAMBURG	21017	HITECH-09	00	1000
T-CSD10	HITECH AG	HAMBURG	21017	HITECH-10	00	1000
T-CSD11	HITECH AG	HAMBURG	21017	HITECH-11	00	1000
T-CSD12	HITECH AG	HAMBURG	21017	HITECH-12	00	1000
T-CSD13	HITECH AG	HAMBURG	21017	HITECH-13	00	1000
T-CSD14	HITECH AG	HAMBURG	21017	HITECH-14	00	1000
T-CSD15	HITECH AG	HAMBURG	21017	HITECH-15	00	1000
T-CSD16	HITECH AG	HAMBURG	21017	HITECH-16	00	1000
T-CSD17	HITECH AG	HAMBURG	21017	HITECH-17	00	1000
T-CSD18	HITECH AG	HAMBURG	21017	HITECH-18	00	1000
T-CSD19	HITECH AG	HAMBURG	21017	HITECH-19	00	1000
T-CSD20	HITECH AG	HAMBURG	21017	HITECH-20	00	1000
T-S62E01	HITECH AG	HAMBURG	21017	01LO605-05	00	1000
T-S62E02	HITECH AG	HAMBURG	21017	02LO605-05	00	1000

dynamically based on which Match Code the user has selected in the combo box. After entering the relevant selection criteria, the user presses the *Search* button in order to display the result set shown in **Figure 34**. Now the customer can select any of the displayed customers by double-clicking or cancel the search by hitting the *Esc* key. **Figure 35** shows the main screen of the sample program again.

Figure 35 Customer Detail Information



The user has picked a customer and the application has used the customer number in order to retrieve customer information using the `Customer.GetDetail` BAPI.

Conclusion

While utilizing the new Match Code support for BAPIs in 4.6 is not the easiest endeavor imaginable, I hope that you will agree that the new capabilities allow us to add exciting new features to our applications. And once you have built the appropriate component(s) you can fortunately forget all the details.

As always, have fun!

