Currencies and Currency Conversions in BAPI Programming

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 CA925 and CA926 classes. Prior to founding ARAsoft in 1993, he worked with SAP AG and SAP America for seven years.

(complete bio appears on page 96)

Almost all BAPI-based applications have to deal with currencies. Handling the currency fields in a BAPI correctly requires an in-depth understanding of how currency amounts are represented in SAP. This article discusses the following topics:

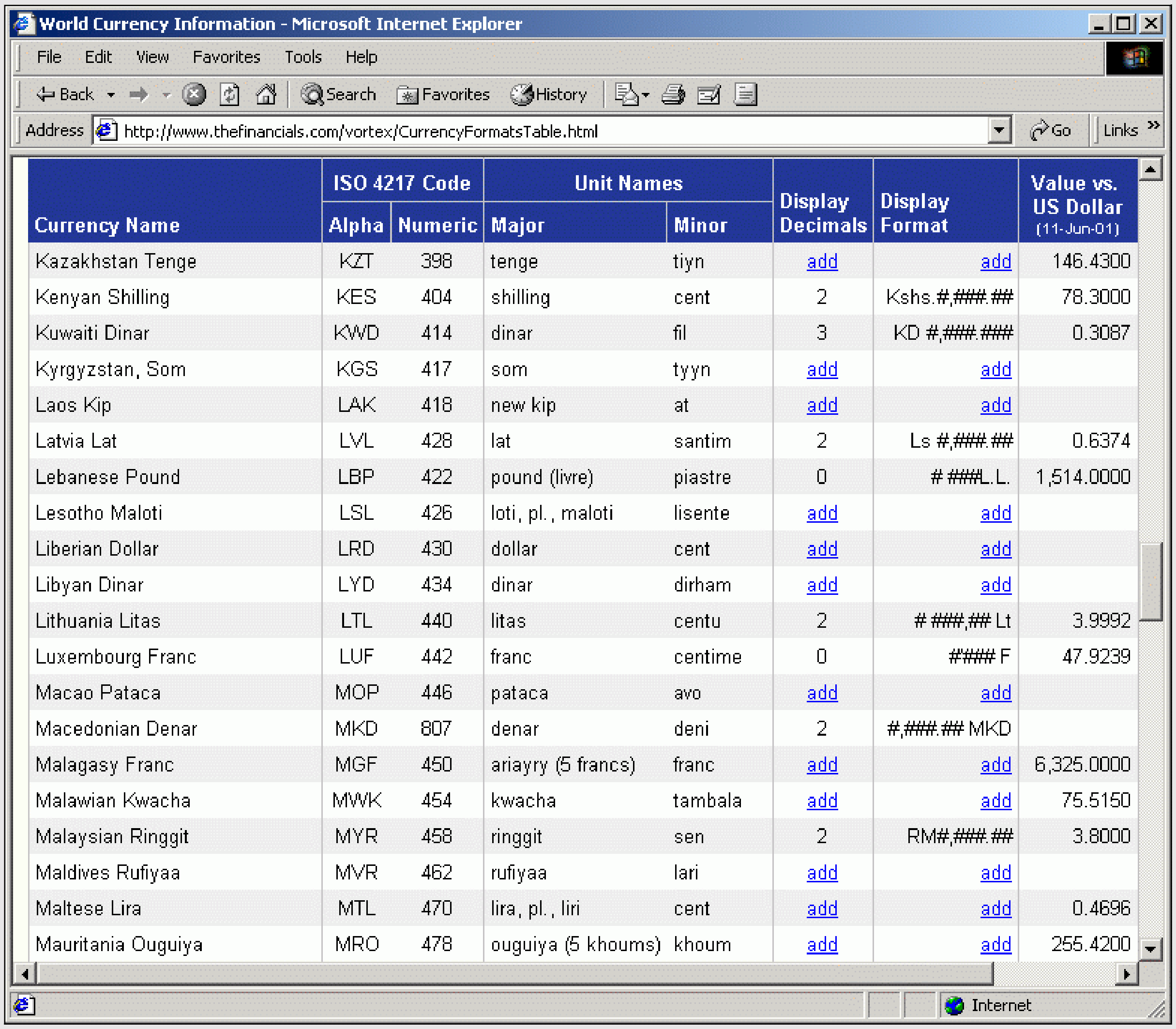
- How should the BAPIs treat currency amounts?
- How do you handle those BAPIs that violate the rules and return "incorrect" amounts?
- How can you use the BAPIs of the `Currency` object type in your applications?
- How can you use the BAPIs of the `ExchangeRate` object type to assist you in converting currency amounts into other currencies?

Anybody involved in BAPI programming, be it as a developer of BAPIs in ABAP or of client applications using BAPIs, should know about these issues.

Currency Fields in BAPI Parameters

Many BAPIs deal with currency amounts. To communicate a currency amount, you need two fields, one for the amount itself and one for the unit, the currency code. Let us deal with the amount field first. Different currencies use different numbers of decimal places. While most use two decimals, there are exceptions: BEF (Belgian Franc),

Figure 1 *Attributes of Some Currencies*


World Currency Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://www.thefinancials.com/vortex/CurrencyFormatsTable.html
Go Links
Internet

	ISO 4217 Code		Unit Names		Display Decimals	Display Format	Value vs. US Dollar (11-Jun-01)
Currency Name	Alpha	Numeric	Major	Minor			
Kazakhstan Tenge	KZT	398	tenge	tiyn	add	add	146.4300
Kenyan Shilling	KES	404	shilling	cent	2	Kshs.#,###.##	78.3000
Kuwaiti Dinar	KWD	414	dinar	fil	3	KD #,###.###	0.3087
Kyrgyzstan, Som	KGS	417	som	tyyn	add	add	
Laos Kip	LAK	418	new kip	at	add	add	
Latvia Lat	LVL	428	lat	santim	2	Ls #,###.##	0.6374
Lebanese Pound	LBP	422	pound (livre)	piastre	0	# ###L.L.	1,514.0000
Lesotho Maloti	LSL	426	loti, pl., maloti	lisente	add	add	
Liberian Dollar	LRD	430	dollar	cent	add	add	
Libyan Dinar	LYD	434	dinar	dirham	add	add	
Lithuania Litas	LTL	440	litas	centu	2	# ###,## Lt	3.9992
Luxembourg Franc	LUF	442	franc	centime	0	#### F	47.9239
Macao Pataca	MOP	446	pataca	avo	add	add	
Macedonian Denar	MKD	807	denar	deni	2	#,###.## MKD	
Malagasy Franc	MGF	450	ariayry (5 francs)	franc	add	add	6,325.0000
Malawian Kwacha	MWK	454	kwacha	tambala	add	add	75.5150
Malaysian Ringgit	MYR	458	ringgit	sen	2	RM#,###.##	3.8000
Maldives Rufiyaa	MVR	462	rufiyaa	lari	add	add	
Maltese Lira	MTL	470	lira, pl., liri	cent	add	add	0.4696
Mauritania Ouguiya	MRO	478	ouguiya (5 khoums)	khoum	add	add	255.4200

ESP (Spanish Peseta), ITL (Italian Lira), and JPY (Japanese Yen), for instance, use no decimals, while KWD (Kuwaiti Dinar) uses three. **Figure 1** is a screenshot of a web site that offers useful information about the world's currencies, including the number of decimals and the preferred display format.

Internally, SAP uses data type `CURR` to represent currency amounts. `CURR` is a packed number (or Binary Coded Decimal, if you prefer) with two decimals. How are currencies that use a different number of decimals stored? In order to be able to store large amounts in currency fields, SAP shifts the amount so that the last digit of the value is stored in the second decimal position. Two Japanese Yen, for example, are stored as 0.02, which is wrong by a factor of 100. SAP internally knows how to handle this and converts the amounts as required before displaying them. In order to avoid that extra effort for BAPI client programmers, SAP decided not to use data type `CURR` in BAPIs.

The BAPI Programming Guide (4.6B) clearly states:

Currency amount fields

In an R/3 System a currency amount field can only be used when accompanied by a currency code. Only then can the decimal point be set correctly. A currency code field must be assigned to each currency amount field. For example, two yen are stored as 0.02 in the field of data type CURR in the database. Non-SAP systems will not understand this syntax.

All R/3 currency data types have two digits after the decimal point, even though currencies do exist with three digits after the decimal point.

For these reasons, the data type CURR cannot be used in the BAPI interface. Adhere to the following guidelines when using currency amount fields in BAPIs:

- *You must not use parameters and fields of data type CURR in the interface*
- *All parameters and fields for currency amounts must use the domain BAPICURR with the data element BAPICURR_D or BAPICUREXT with the data element BAPICUREXT.*
- *The position of the decimal point in currency amount fields must be converted correctly*

You can use two function modules for this conversion. The function module BAPI_CURRENCY_CONV_TO_EXTERNAL converts currency amounts from R/3 internal data formats into external data formats. The function module BAPI_CURRENCY_CONV_TO_INTERNAL converts currency amounts from external data formats into internal data formats.

Following rules is something that not everybody is equally good at. When I checked in the BAPI Explorer to ensure that all BAPIs are indeed following the rule (i.e., do not use data type `CURR`), I soon discovered some where that was not the case. For a more systematic analysis, I wrote the program shown in **Figure 2**.

Figure 2 *Program to Find All Usages of Data Type CURR*

```
import com.sap.mw.jco.*;
import de.arasoft.sap.jco.objectfactory.*;
public class GetCURR {
  JCO.Client mConnection;
  public GetCURR() {
    try {
      // Change the logon information for your own system/user
      mConnection =
        JCO.createClient(
        "001",            // SAP client
        "arasoft",        // userid
        "***",            // password
        null,             // language
        "host01",         // host name
        "00");            // system number
      mConnection.connect();
      ObjectFactory bof = new ObjectFactory(mConnection);
```

(continued on next page)

Figure 2 (continued)

```
        for (int i = 0; i < bof.getBOTypes().getSize(); i++) {
          BOType type = bof.getBOTypes().getBOType(i);
          for (int j = 0; j < type.getBOMethods().getSize(); j++) {
            BOMethod method = type.getBOMethods().getBOMethod(j);
            for (int k = 0; k < method.getBOParameters().getSize();
                 k++) {
              BOParameter param =
                method.getBOParameters().getBOParameter(k);
              if (param.isScalar()) {
                if (param.getScalarDataTypeDD().equals("CURR")) {
                  System.out.println(type.getObjectName() + "\t" +
                    type.getObjectType() + "\t" +
                    method.getName() + "\t" +
                    param.getName()
                    );
                }
              } else if (param.isStructure() || param.isTable()) {
                Structure struct = param.getStructure();
                for (int l = 0; l < struct.getFields().getSize(); l++) {
                  Field field = struct.getField(l);
                  if (field.getDataTypeDD().equals("CURR")) {
                    System.out.println(type.getObjectName() + "\t" +
                      type.getObjectType() + "\t" +
                      method.getName() + "\t" +
                      param.getName() + "\t" +
                      field.getName()
                      );
                  }
                }
              }
            }
          }
        }
      }
      catch (Exception ex) {
        ex.printStackTrace();
        System.exit(1);
      }
      mConnection.disconnect();
    }
    public static void main(String[] args) {
      GetCURR getCURR1 = new GetCURR();
    }
  }
```

Figure 3 contains the results for 4.6B. As you can see, there are quite a few exceptions to the rule!

If the field name column is blank, then the parameter is a simple field (a scalar parameter) as opposed to a table or structure parameter.

(Text continues on page 87.)

Figure 3 *All Usages of Data Type CURR in 4.6B*

Object Name	Object Type	BAPI Name	Parameter Name	Field Name
ApprovingOfficer	HRPSSG_AO	GetList	InternalControl	CARVALUE
Attendee	PDOTYPE_PT	GetBookList	AttendeeBookList	KKOST
Attendee	PDOTYPE_PT	GetCompanyBookList	CompanyBookList	KKOST
Attendee	PDOTYPE_PT	GetCompanyPrebookList	CompanyPrebookList	KKOST
Attendee	PDOTYPE_PT	GetPrebookList	AttendeePrebookList	KKOST
BBPIncomingInvoice	BBPBUS2081	BbpInvoiceCreate	IV_Header	AMOUNT
BBPIncomingInvoice	BBPBUS2081	BbpInvoiceCreate	IV_Item	AMOUNT
BBPIncomingInvoice	BBPBUS2081	BbpInvoiceCreate	IV_Shp	AMOUNT
BBPIncomingInvoice	BBPBUS2081	BbpInvoiceCreate	IV_Tax	AMOUNT
BusinessEvent	PDOTYPE_E	GetInfo	EventPrice	IKOST
BusinessEvent	PDOTYPE_E	GetInfo	EventPrice	EKOST
BusinessEventtype	PDOTYPE_D	GetInfo	EventtypePrice	IKOST
BusinessEventtype	PDOTYPE_D	GetInfo	EventtypePrice	EKOST
Customer	KNA1	CheckExistence	CustomerData	UMSAT
Customer	KNA1	CheckExistence	CustomerData	UMSA1
Customer	KNA1	CheckPassword	CustomerData	UMSAT
Customer	KNA1	CheckPassword	CustomerData	UMSA1
CustomerInquiry	BUS2030	CreateFromData	BillingParty	CRED_LIMIT
CustomerInquiry	BUS2030	CreateFromData	BillingParty	ORDER_VALS
CustomerInquiry	BUS2030	CreateFromData	BillingParty	RCVBL_VALS
CustomerInquiry	BUS2030	CreateFromData	BillingParty	CRED_LIAB
CustomerInquiry	BUS2030	CreateFromData	BillingParty	VAL_LIMIT
CustomerInquiry	BUS2030	CreateFromData	OrderItemsIn	COND_VALUE
CustomerQuotation	BUS2031	CreateFromData	BillingParty	CRED_LIMIT
CustomerQuotation	BUS2031	CreateFromData	BillingParty	ORDER_VALS
CustomerQuotation	BUS2031	CreateFromData	BillingParty	RCVBL_VALS
CustomerQuotation	BUS2031	CreateFromData	BillingParty	CRED_LIAB
CustomerQuotation	BUS2031	CreateFromData	BillingParty	VAL_LIMIT
CustomerQuotation	BUS2031	CreateFromData	OrderItemsIn	COND_VALUE
Employee	EMPLOYEET	GetList	InternalControl	CARVALUE
EmployeeAbstract	BUS1065	GetList	InternalControl	CARVALUE
EmployeeBankDetail	BANKDETAIL	Change	Standardvalue	

(continued on next page)

Figure 3 (continued)

Object Name	Object Type	BAPI Name	Parameter Name	Field Name
EmployeeBankDetail	BANKDETAIL	Create	Standardvalue	
EmployeeBankDetail	BANKDETAIL	Createsuccessor	Standardvalue	
EmployeeBankDetail	BANKDETAIL	Getdetail	Standardvalue	
EmployeeBankDetail	BANKDETAIL	Getdetailedlist	Bankdetail	STANDARDVALUE
EmployeeBankDetail	BANKDETAIL	Request	Standardvalue	
EmployeeBankDetail	BANKDETAIL	Simulatecreation	Standardvalue	
EmployeeBankDetailJP	BANKJP	Change	Standardvalue	
EmployeeBankDetailJP	BANKJP	Create	Standardvalue	
EmployeeBankDetailJP	BANKJP	Createsuccessor	Standardvalue	
EmployeeBankDetailJP	BANKJP	Getdetail	Standardvalue	
EmployeeBankDetailJP	BANKJP	Getdetailedlist	BANKDETAIL	STANDARDVALUE
EmployeeBankDetailJP	BANKJP	Request	Standardvalue	
EmployeeBankDetailJP	BANKJP	Simulatecreation	Standardvalue	
EmployeeCH	EMPLOYEECH	GetList	InternalControl	CARVALUE
EmployeeE	EMPLOYEEE	GetList	InternalControl	CARVALUE
EmployeeGB	EMPLOYEEGB	GetList	InternalControl	CARVALUE
EmployeeJP	EMPLOYEEJP	GetList	InternalControl	CARVALUE
EmployeeUS	EMPLOYEEUS	GetList	InternalControl	CARVALUE
InternalOrder	BUS2075	Create	E_Master_Data	ESTIMATED_COSTS
InternalOrder	BUS2075	GetDetail	MasterData	ESTIMATED_COSTS
ManagerExtPayroll	BUS7023	InsertLegacy	Wagetypes	RATE
ManagerExtPayroll	BUS7023	InsertLegacy	Wagetypes	AMOUNT
ManagerExtPayroll	BUS7023	InsertOutsourcer	Arrears	AMOUNT
ManagerExtPayroll	BUS7023	InsertOutsourcer	Wagetypes	RATE
ManagerExtPayroll	BUS7023	InsertOutsourcer	Wagetypes	AMOUNT
Network	BUS2002	Getdetail	EActivity	PRICE
Network	BUS2002	Getdetail	EActivity	ACTIVITY_COSTS
Network	BUS2002	Getdetail	EActivity	USER_FIELD_CURR1
Network	BUS2002	Getdetail	EActivity	USER_FIELD_CURR2
Network	BUS2002	Getdetail	EActivityElement	PRICE
Network	BUS2002	Getdetail	EActivityElement	ACTIVITY_COSTS
Network	BUS2002	Getdetail	EActivityElement	USER_FIELD_CURR1
Network	BUS2002	Getdetail	EActivityElement	USER_FIELD_CURR2

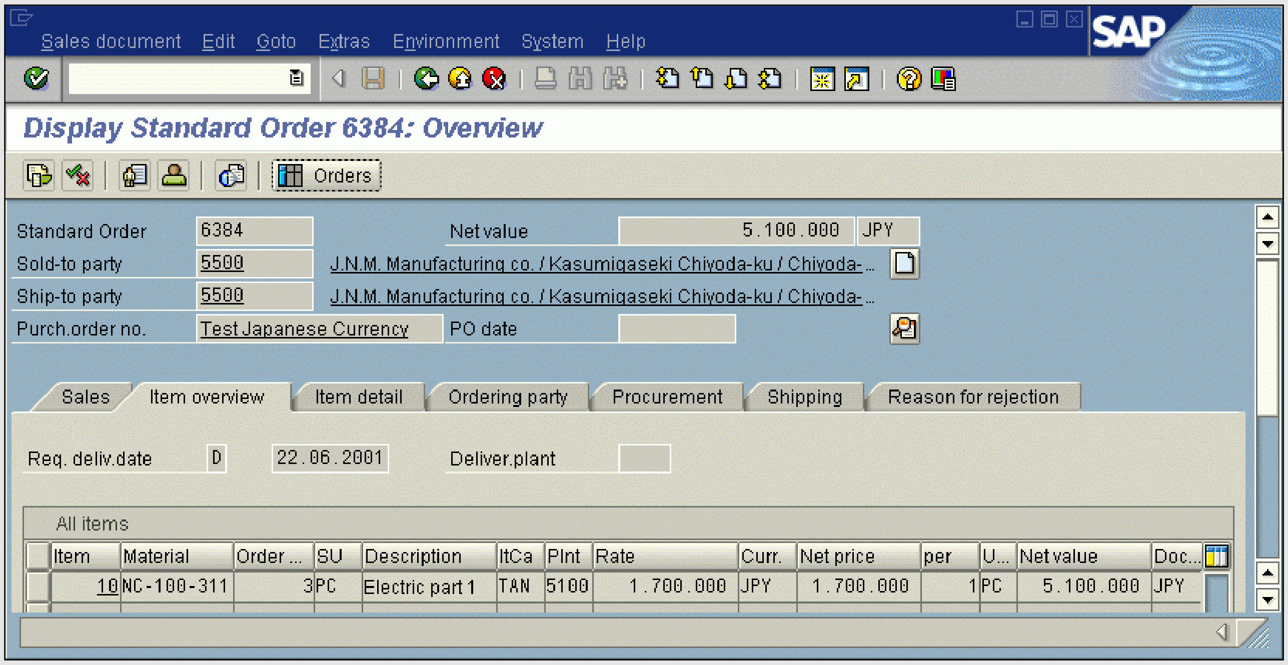
Object Name	Object Type	BAPI Name	Parameter Name	Field Name
Network	BUS2002	Getdetail	EComponent	PRICE
Network	BUS2002	Getinfo	EActivity	PRICE
Network	BUS2002	Getinfo	EActivity	ACTIVITY_COSTS
Network	BUS2002	Getinfo	EActivity	USER_FIELD_CURR1
Network	BUS2002	Getinfo	EActivity	USER_FIELD_CURR2
Network	BUS2002	Getinfo	EActivityElement	PRICE
Network	BUS2002	Getinfo	EActivityElement	ACTIVITY_COSTS
Network	BUS2002	Getinfo	EActivityElement	USER_FIELD_CURR1
Network	BUS2002	Getinfo	EActivityElement	USER_FIELD_CURR2
Network	BUS2002	Getinfo	EComponent	PRICE
Network	BUS2002	Maintain	IActivity	PRICE
Network	BUS2002	Maintain	IActivity	ACTIVITY_COSTS
Network	BUS2002	Maintain	IActivity	USER_FIELD_CURR1
Network	BUS2002	Maintain	IActivity	USER_FIELD_CURR2
Network	BUS2002	Maintain	IActivityElement	PRICE
Network	BUS2002	Maintain	IActivityElement	ACTIVITY_COSTS
Network	BUS2002	Maintain	IActivityElement	USER_FIELD_CURR1
Network	BUS2002	Maintain	IActivityElement	USER_FIELD_CURR2
PTMgrExtPExpenses	BUS7015	Insert	ExternalWagetypes	AMOUNT_EXT_WAGETYPE
PublicSectorSG	PSSGOBTYPE	GetList	InternalControl	CARVALUE
RecOfficerSG	HRPSSG_RO	GetList	InternalControl	CARVALUE
SalesOrder	BUS2032	CreateFromDat1	BillingParty	CRED_LIMIT
SalesOrder	BUS2032	CreateFromDat1	BillingParty	ORDER_VALS
SalesOrder	BUS2032	CreateFromDat1	BillingParty	RCVBL_VALS
SalesOrder	BUS2032	CreateFromDat1	BillingParty	CRED_LIAB
SalesOrder	BUS2032	CreateFromDat1	BillingParty	VAL_LIMIT
SalesOrder	BUS2032	CreateFromDat1	OrderItemsIn	COND_VALUE
SalesOrder	BUS2032	CreateFromData	BillingParty	CRED_LIMIT
SalesOrder	BUS2032	CreateFromData	BillingParty	ORDER_VALS
SalesOrder	BUS2032	CreateFromData	BillingParty	RCVBL_VALS
SalesOrder	BUS2032	CreateFromData	BillingParty	CRED_LIAB
SalesOrder	BUS2032	CreateFromData	BillingParty	VAL_LIMIT

(continued on next page)

Figure 3 (continued)

Object Name	Object Type	BAPI Name	Parameter Name	Field Name
SalesOrder	BUS2032	CreateFromData	OrderItemsIn	COND_VALUE
SalesOrder	BUS2032	CreateWithDia	SalesConditionsIn	ROUNDOFFDI
SalesOrder	BUS2032	GetList	SalesOrders	NET_PRICE
SalesOrder	BUS2032	GetList	SalesOrders	NET_VAL_HD
SalesOrder	BUS2032	GetList	SalesOrders	NET_VALUE
SalesOrder	BUS2032	GetStatus	Statusinfo	NET_VALUE
SalesOrder	BUS2032	GetStatus	Statusinfo	NET_PRICE
SalesOrder	BUS2032	Simulate	BillingParty	CRED_LIMIT
SalesOrder	BUS2032	Simulate	BillingParty	ORDER_VALS
SalesOrder	BUS2032	Simulate	BillingParty	RCVBL_VALS
SalesOrder	BUS2032	Simulate	BillingParty	CRED_LIAB
SalesOrder	BUS2032	Simulate	BillingParty	VAL_LIMIT
SalesOrder	BUS2032	Simulate	OrderItemsIn	COND_VALUE
SiteLayoutModule	BUS1083	GetItem	Items	MOVEMENT
SiteLayoutModule	BUS1083	GetItems	Items	MOVEMENT
WorkBreakdownStruct	BUS2054	Getinfo	EActivityTable	PRICE
WorkBreakdownStruct	BUS2054	Getinfo	EActivityTable	ACTIVITY_COSTS
WorkBreakdownStruct	BUS2054	Getinfo	EActivityTable	USER_FIELD_CURR1
WorkBreakdownStruct	BUS2054	Getinfo	EActivityTable	USER_FIELD_CURR2
WorkBreakdownStruct	BUS2054	Getinfo	EWbsElementTable	USER_FIELD_CURR1
WorkBreakdownStruct	BUS2054	Getinfo	EWbsElementTable	USER_FIELD_CURR2
WorkBreakdownStruct	BUS2054	Maintain	IActivity	PRICE
WorkBreakdownStruct	BUS2054	Maintain	IActivity	ACTIVITY_COSTS
WorkBreakdownStruct	BUS2054	Maintain	IActivity	USER_FIELD_CURR1
WorkBreakdownStruct	BUS2054	Maintain	IActivity	USER_FIELD_CURR2
WorkBreakdownStruct	BUS2054	Maintain	IActivityElement	PRICE
WorkBreakdownStruct	BUS2054	Maintain	IActivityElement	ACTIVITY_COSTS
WorkBreakdownStruct	BUS2054	Maintain	IActivityElement	USER_FIELD_CURR1
WorkBreakdownStruct	BUS2054	Maintain	IActivityElement	USER_FIELD_CURR2
WorkBreakdownStruct	BUS2054	Maintain	IWbsElementTable	USER_FIELD_CURR1
WorkBreakdownStruct	BUS2054	Maintain	IWbsElementTable	USER_FIELD_CURR2
WorkBreakdownStruct	BUS2054	SaveReplica	WbsElement	USER_FIELD_CURR1
WorkBreakdownStruct	BUS2054	SaveReplica	WbsElement	USER_FIELD_CURR2

Figure 4 *Displaying a Sales Order Using Japanese Currency*



Display Standard Order 6384: Overview

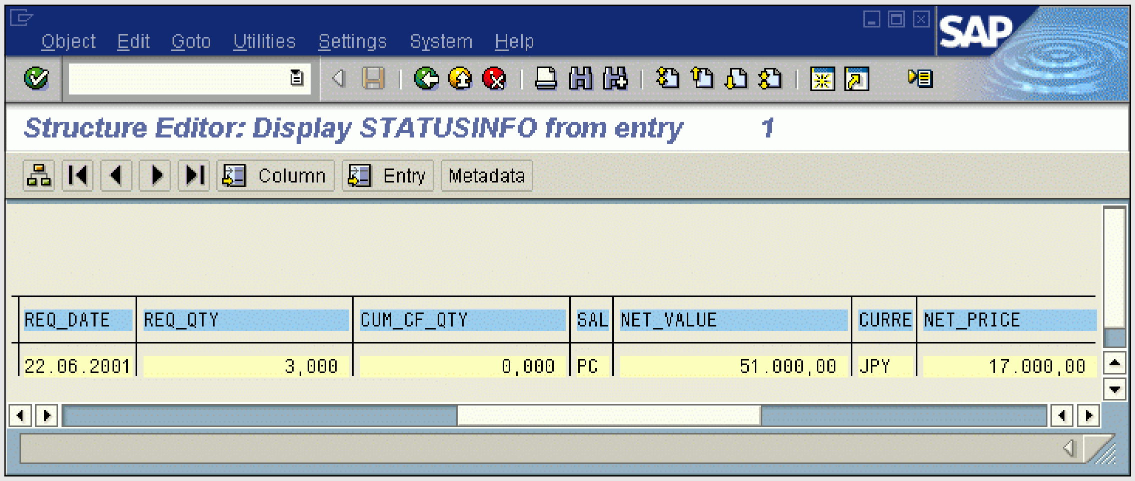
Standard Order	6384	Net value	5.100.000 JPY
Sold-to party	5500	J.N.M. Manufacturing co. / Kasumigaseki Chiyoda-ku / Chiyoda-...	
Ship-to party	5500	J.N.M. Manufacturing co. / Kasumigaseki Chiyoda-ku / Chiyoda-...	
Purch.order no.	Test Japanese Currency	PO date	

Req. deliv.date D 22.06.2001 Deliver.plant

All items

Item	Material	Order ...	SU	Description	ItCa	Plnt	Rate	Curr.	Net price	per	U...	Net value	Doc...
10	NC-100-311	3	PC	Electric part 1	TAN	5100	1.700.000	JPY	1.700.000	1	PC	5.100.000	JPY

Figure 5 *Using a BAPI to Display the Sales Order*



Structure Editor: Display STATUSINFO from entry 1

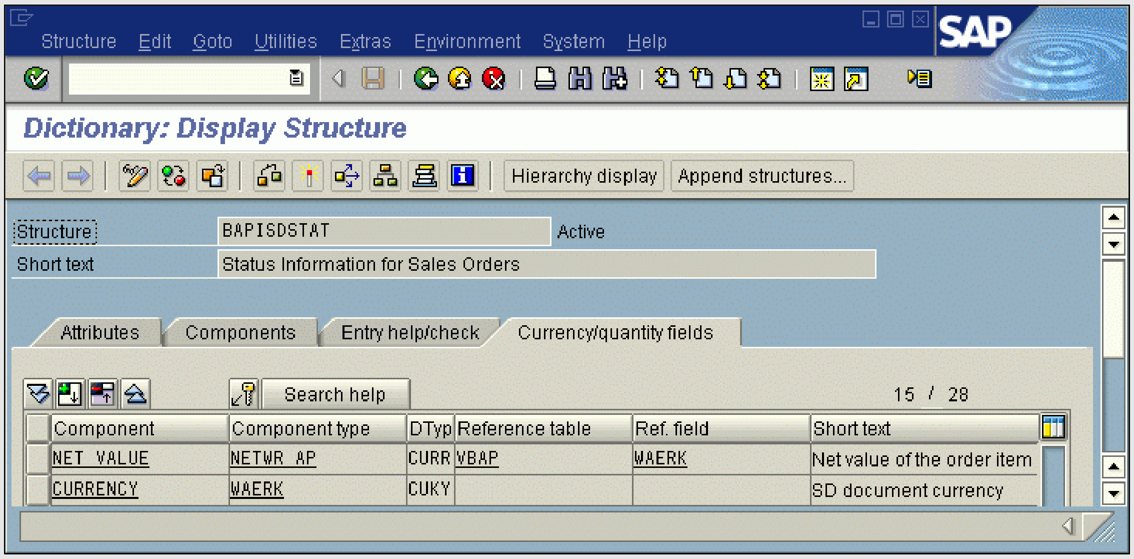
REQ_DATE	REQ_QTY	CUM_CF_QTY	SAL	NET_VALUE	CURRE	NET_PRICE
22.06.2001	3,000	0,000	PC	51.000,00	JPY	17.000,00

(Text continued from page 82.)

What exactly are the consequences if you want to use one of the BAPIs in Figure 3? To demonstrate that, I created a sales order for a Japanese customer, displayed in **Figure 4**.

As you can see, the complete value of the order is 5.100.000 Yen. Let us now check the results of calling the BAPI `SalesOrder.GetStatus` for this order. **Figure 5** is a screenshot of the `Statusinfo` parameter returned by the BAPI.

Figure 6 *Displaying Metadata in the Dictionary*



Structure BAPISDSTAT Active
Short text Status Information for Sales Orders

15 / 28

Component	Component type	DTyp	Reference table	Ref. field	Short text
NET_VALUE	NETWR_AP	CURR	VBAP	WAERK	Net value of the order item
CURRENCY	WAERK	CUKY			SD document currency

Field NET_VALUE contains 51.000 instead of 5.100.000. This is exactly why SAP created the rule about not using data type CURR in BAPIs!

Since in many applications we will need to use some of the BAPIs in Figure 3, we need a way to deal with this situation. First, it is important that whenever you use a BAPI with currency amounts, you verify whether the BAPI breaks the rule. You can do this in the BAPI Explorer or by checking Figure 3. If CURR is indeed used, then we will have to adjust the amount. To be able to do this we need to find the number of decimals for the associated currency code.

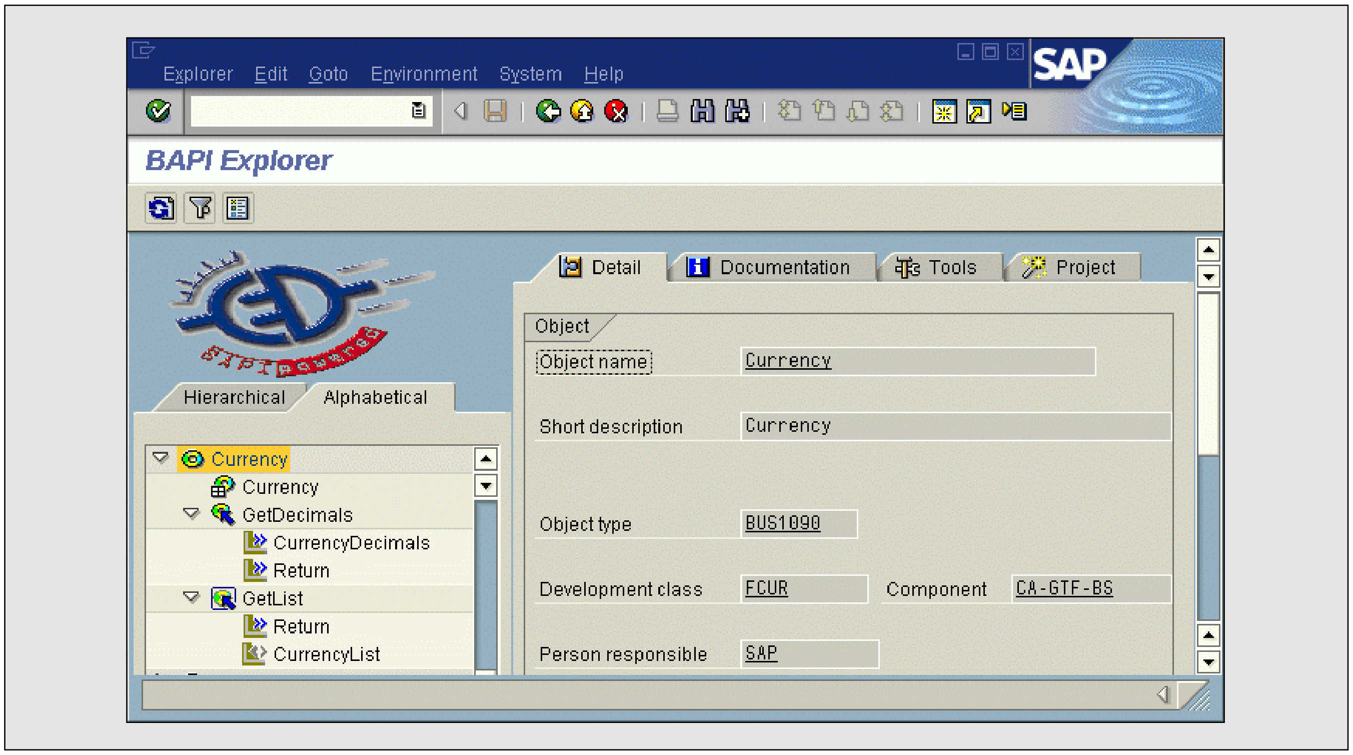
Currency Codes in BAPI Parameters

For each currency amount, there must be an associated currency code. In Figure 5, it is returned in the field after NET_VALUE, called CURRENCY (only the first five letters are visible in Figure 5 since the field is only five characters wide). In the next section we will learn how to find out the number of decimals for a currency code by invoking a BAPI.

But first I would like to discuss another issue: How do we know which currency code is associated with a specific amount field? Unfortunately, the answer is that there is no algorithm to do this, we have to use human instead of computer intelligence. Some of you might now say, "Wait, there is metadata available in the dictionary that enables you to programmatically determine the associated currency code." **Figure 6** displays the two fields NET_VALUE and CURRENCY in the dictionary.

As you can see, there is a reference for field NET_VALUE (VBAP-WAERK) that should refer to the associated currency code. But as you can also see, it does not refer to the CURRENCY field in our structure. In other words, this does not help us to determine the associated currency code automatically. For normal client applications, this is not a big deal. You can check the dictionary and find the required code field, usually directly before or after the amount. For tools and very flexible applications, this is bad news, and as I said, there is no solution. (Unless you want an algorithm that guesses and sometimes guesses wrong.)

Figure 7 *BAPIs of Object Type Currency*



Object

Object name	Currency		
Short description	Currency		
Object type	BUS1090		
Development class	FCUR	Component	CA-GTF-BS
Person responsible	SAP		

Figure 8 *Fields of Parameter CurrencyDecimals*

Field Name	Data Type	Description
CURRENCY	Character 5	SAP currency code
CURDECIMALS	1-byte Integer	Number of decimals
CURRENCY_ISO	Character 5	ISO currency code

Using the BAPIs of Object Type Currency

Let us return to our quest for the number of decimals for a given currency code. **Figure 7** displays the two BAPIs of the `Currency` object type.

The `GetDecimals` BAPI is what we need. Its `CurrencyDecimals` parameter has the fields shown in **Figure 8**.

Field **`CURDECIMALS`** contains the required value. Using this allows us to make the necessary adjustment to an amount in a parameter of a BAPI that uses data type `CURR`. In our specific case, the Japanese Yen uses zero decimals, hence we need to multiply the amount by a factor of 100. The general formula for the factor is

```
10**(2 – Number_of_decimals)
```

Couldn't we save some work and use the functions `BAPI_CURRENCY_CONV_TO_EXTERNAL` and `BAPI_CURRENCY_CONV_TO_INTERNAL`

Figure 9 Fields of Parameter CurrencyList

Field Name	Data Type	Description
CURRENCY	Character 5	SAP currency code
CURRENCY_ISO	Character 5	ISO currency code
ALT_CURR	Character 3	Alternative currency code
VALID_TO	Date	Date until which the currency code is valid
LONG_TEXT	Character 40	Description

that SAP recommends for this purpose in their BAPI Programming Guide (cf. earlier quote)? Unfortunately, these functions (despite their names) are not really BAPIs. They are not RFC-enabled and can only be invoked from within ABAP.

The number of decimals is also required to properly format amount fields that follow the rules. The `BAPICURR` and `BAPICUREXT` domains that any rule-abiding BAPI should use have a fixed number of decimals (four and nine, respectively). While the amounts in fields based on these domains are correct, we will still have to format them for display in any user interface. Figure 4 shows you that SAP takes into account the number of decimals for a given currency code when displaying amounts, and our applications should do the same.

Field **`CURRENCY_ISO`** contains the standard ISO currency code. This is often the same as the SAP currency code, but not always. This field allows you to display the ISO code instead of the SAP code in your GUI if you find that more appropriate.

The other BAPI of object type `Currency`, `GetList`, returns a list of all currency codes defined in the system. The fields of the `CurrencyList` parameter are listed in **Figure 9**.

The `ALT_CURR` field is, according to the SAP documentation, currently only used in Belgium and Spain, and I cannot tell you anything more about it. All the other fields should be easy to understand.

`Currency.GetList` is used mainly for two purposes:

- To provide a drop-down list for a user to select a currency code from.
- To get the description for a currency code, to be displayed in addition to — or instead of — the currency code.

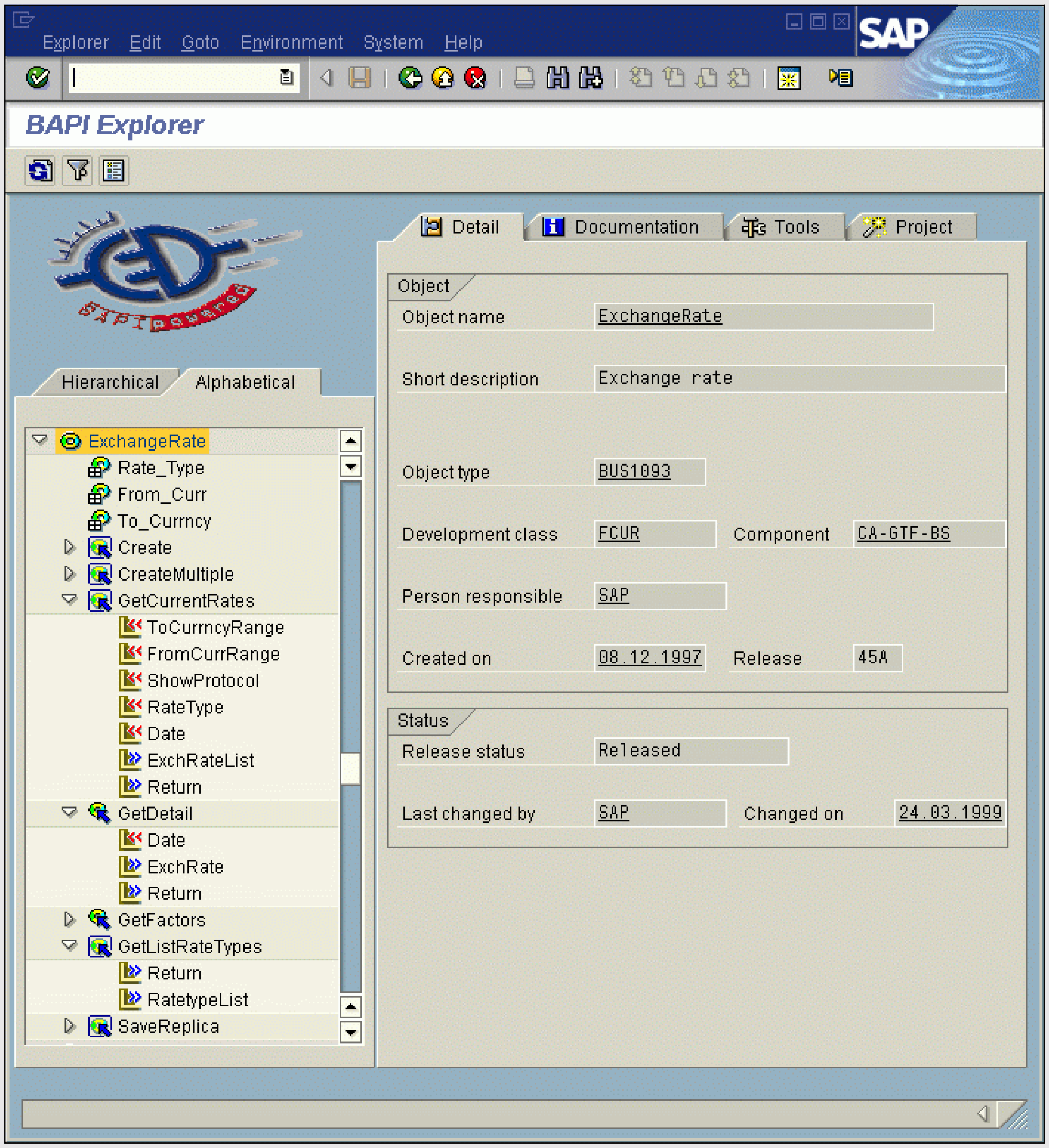
Let us summarize our discussion so far:

Whenever you use BAPIs that deal with currency amounts you need to check whether the data type `CURR` is utilized for the amount. If so, the value needs to be adjusted. To enable you to do this, and for some other purposes, the `Currency` object type offers two very useful BAPIs.

Converting Currency Amounts to Different Currencies

Often, we need to convert a currency amount into another currency, e.g., to show a customer in a foreign country the price of a product in the local currency. SAP offers the `ExchangeRate` object type to handle currency conversion issues. **Figure 10** is a screenshot of this object type in the BAPI Explorer.

Figure 10 *The ExchangeRate Object Type*



Object

Object name	ExchangeRate		
Short description	Exchange rate		
Object type	BUS1093		
Development class	FCUR	Component	CA-GTF-BS
Person responsible	SAP		
Created on	08.12.1997	Release	45A

Status

Release status	Released		
Last changed by	SAP	Changed on	24.03.1999

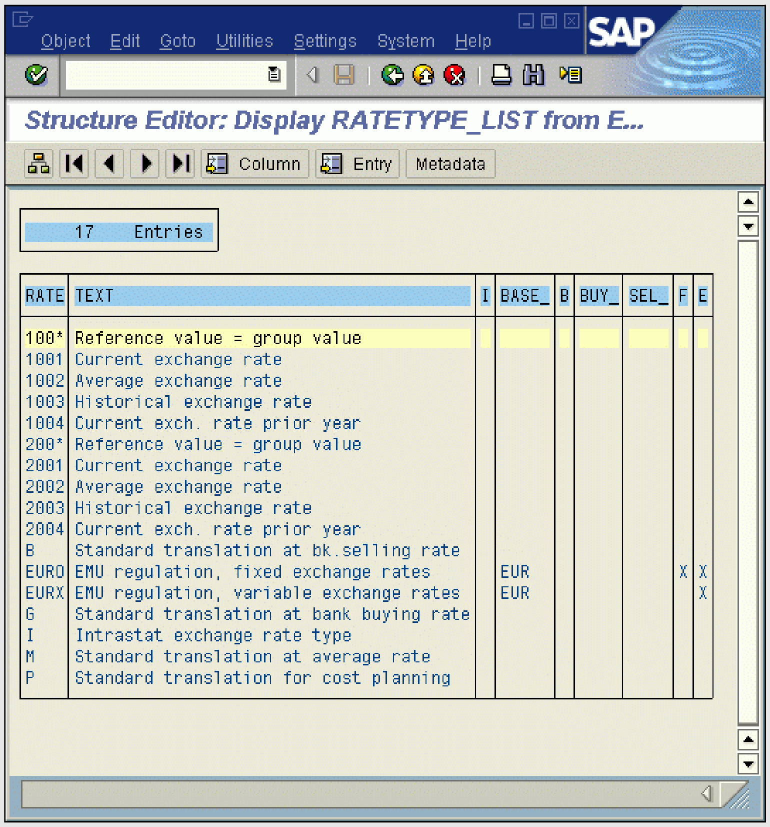
There are three key fields (`Rate_Type`, `From_Curr`, and `To_Currncy`) that we will have to set to be able to call instance-dependent BAPIs like `GetDetail`. On the level of the underlying RFM (RFC-enabled Function Module) those key fields will simply appear as import parameters.

What is a rate type? As you probably know, there are different exchange rates depending on whether you are buying or selling currencies. And these are just the two most important rate types. The SAP system comes with several pre-configured rate types and customers can add their own. To get a list of all rate types defined in a system, we use the

Figure 11 *Fields of Parameter RatetypeList*

Field Name	Data Type	Description
RATE_TYPE	Character 4	Exchange rate type
TEXT	Character 40	Description
INVR_ALLOW	Character 1	Indicator: Calculation allowed with inverted exchange rate?
BASE_CURR	Character 5	Reference currency for currency translation
BCURR_FROM	Character 1	Ind.: Base curr. is "From" curr. in the exchange rate table
BUY_RT_FOR	Character 4	Exch. rate type of av. rate used to determine buying rate
SEL_RT_FOR	Character 4	Exch. rate type of av. rate used to determine selling rate
FIXED_RATE	Character 1	Indicator: Exchange rate type uses fixed exchange rates
EMU_CONV	Character 1	Indicator: Exchange rate type uses special translation model

Figure 12 *Exchange Rate Types*



Structure Editor: Display RATETYPE_LIST from E...

17 Entries

RATE	TEXT	I	BASE_	B	BUY_	SEL_	F	E
100*	Reference value = group value							
1001	Current exchange rate							
1002	Average exchange rate							
1003	Historical exchange rate							
1004	Current exch. rate prior year							
200*	Reference value = group value							
2001	Current exchange rate							
2002	Average exchange rate							
2003	Historical exchange rate							
2004	Current exch. rate prior year							
B	Standard translation at bk.selling rate							
EURO	EMU regulation, fixed exchange rates		EUR				X	X
EURX	EMU regulation, variable exchange rates		EUR					X
G	Standard translation at bank buying rate							
I	Intrastat exchange rate type							
M	Standard translation at average rate							
P	Standard translation for cost planning							

`GetListRateTypes` BAPI. Its `RatetypeList` table parameter contains the fields listed in **Figure 11**.

For normal requirements, fields `RATE_TYPE` and `TEXT` will suffice, for everything else you should

Figure 13 *ExchangeRate.GetDetail Parameters*

Parameter	Data Type	In/Out	Description
RATE_TYPE	Character 4	In	Exchange rate type
FROM_CURR	Character 5	In	“From” currency
TO_CURRNCY	Character 5	In	“To” currency
DATE	Date	In	Value date
EXCH_RATE	Structure	Out	Exchange rate information
RETURN	Structure	Out	The standard BAPI return parameter

Figure 14 *Fields of Structure BAPI1093_0*

Field Name	Data Type	Description
RATE_TYPE	Character 4	Exchange rate type
FROM_CURR	Character 5	“From” currency
TO_CURRNCY	Character 5	“To” currency
VALID_FROM	Date	Date from which the entry is valid
EXCH_RATE	Packed number (format: 1234.12345)	Direct quoted exchange rate
FROM_FACTOR	Packed number (format: 123456789)	Ratio for the “From” currency units
TO_FACTOR	Packed number (format: 123456789)	Ratio for the “To” currency units
EXCH_RATE_V	Packed number (format: 1234.12345)	Indirect quoted exchange rate
FROM_FACTOR_V	Packed number (format: 123456789)	Ratio for the “From” currency units
TO_FACTOR_V	Packed number (format: 123456789)	Ratio for the “To” currency units

study the SAP documentation. **Figure 12** is an example of the data you might get when calling the BAPI. In your system, you might have even more exchange rate types.

“B”, “G”, and “M” are the most commonly used types.

Now that we know how to get a list of rate types, we can proceed to an actual conversion. BAPI `GetDetail` is the one that will be of assistance here. Its parameters (on the RFM level) are shown in **Figure 13**.

All *In* (import) parameters are mandatory. If an exchange rate was found for the specified currency pair for the specified date (check the `RETURN` parameter to verify), all pertinent information is returned in the `EXCH_RATE` parameter, the fields of which are listed in **Figure 14**.

Figure 15 ExchangeRate.GetCurrentRates Parameters

Parameter	Data Type	In/Out	Description
DATE	Date	In	Selection date
DATE_TYPE	Character 1	In	Date type, default "V"
RATE_TYPE	Character 4	In	Exchange rate type, default "M"
SHOW_PROTOCOL	Character 1	In	Indicator: Display log
FROM_CURR_RANGE	Table	In	"From" currency selection criteria
TO_CURRNCY_RANGE	Table	In	"To" currency selection criteria
EXCH_RATE_LIST	Table	Out	List of exchange rates
RETURN	Table	Out	The standard BAPI return parameter

The first four fields require no further explanation. The remaining six can be broken into two groups of three fields. The first group (always populated) contains an exchange rate based on the "From" currency. If I want to convert German Marks (DEM) to US Dollars (USD), for instance, the exchange rate (field `EXCH_RATE`) would give me the equivalent of 1 DEM in USD (roughly 0.44 at the time of writing). `FROM_FACTOR` and `TO_FACTOR` are required so that exchange rates between currencies that have a huge difference in the value of one unit can be expressed without requiring more than four digits before the decimal point. So while both fields contain 1 for the conversion between DEM and USD, the situation is different if we want to convert Italian Lire (ITL) to USD: Here the `FROM_FACTOR` is 1000 and the `TO_FACTOR` is 1. The formula for the conversion of a currency amount is thus

```
amount_in_from_currency * EXCH_RATE /
FROM_FACTOR * TO_FACTOR
```

In some instances (for the Euro, for example) it is required or customary to express the exchange rate the other way around (known as indirect quotation). The exchange rate is based on the "To" currency. The required information (if available) is returned in the second group of three fields: `EXCH_RATE_V`, `FROM_FACTOR_V`, and `TO_FACTOR_V`. `EXCH_RATE_V` must obviously contain the reciprocal value of `EXCH_RATE`, but the result of a conversion may be slightly different depending on whether you use direct or indirect quotation, the reason being that 1 divided by the reciprocal value is not equal to the original value for numbers with a limited number of decimals. The formula for conversions based on an indirect quotation is

```
amount_in_from_currency / EXCH_RATE_V /
FROM_FACTOR_V * TO_FACTOR_V
```

In other words, we divide instead of multiply by the exchange rate.

Remember to check the `RETURN` parameter to make sure that an exchange rate was available. This leads to the question of how to find out for which currency pairs exchange rates are available in a system. The `GetCurrentRates` BAPI can help us with that. Its parameters (on the RFM level) are shown in **Figure 15**.

Figure 16 Fields for FROM_CURR_RANGE and TO_CURRNCY_RANGE

Field Name	Data Type	Description
SIGN	Character 1	Inclusion/exclusion criterion SIGN for range tables
OPTION	Character 2	Selection operator OPTION for range tables
LOW	Character 5	Currency code
HIGH	Character 5	Currency code

The `DATE`, `FROM_CURR_RANGE`, `TO_CURRNCY_RANGE`, `EXCH_RATE_LIST`, and `RETURN` parameters are mandatory.

The meaning of the `DATE` parameter depends on the setting of the `DATE_TYPE` parameter. The default ("V") means that you want all exchange rates valid *on* the specified date. This is almost always what we are interested in, hence the parameter is not even defined in the Business Object Repository (cf. Figure 10). A value of "E" means that we only want exchange rates the validity of which begins exactly on the specified date. This may be useful for a maintenance application, but not for the purpose of converting currencies.

The `SHOW_PROTOCOL` parameter controls whether or not information should be returned in the `RETURN` parameter (which for this BAPI is a table). Ignoring the parameter or passing a blank string means that you are not interested in error messages. Passing an "X" means that you want one row for each currency pair for which an exchange rate was not available for the specified date. While normally the `RETURN` is very important to check, here I recommend not to use it. The `EXCH_RATE_LIST` contains all available exchange rates, hence it is superfluous to get a list of currency pairs for which no exchange rate was available. Leaving `SHOW_PROTOCOL` blank saves a lot of data transmission because the `RETURN` table can easily contain a few thousand rows. Unless you limit the amount of returned information by specifying selection criteria, which is a good idea anyway. If you leave the `FROM_CURR_RANGE` and `TO_CURRNCY_RANGE` table parameters empty, the BAPI will take forever (70 seconds on my system) to execute. Usually, we only want to know which currencies a given currency can be converted from or to. The structure of the `FROM_CURR_RANGE` and `TO_CURRNCY_RANGE` parameters is given in **Figure 16**.

These parameters allow us to specify selection criteria. Both table parameters can contain as many rows as required, although in real life there will hardly ever be more than one.

`SIGN` can contain "I" or "E". "I" means that the BAPI should include information based on the condition, "E" means the opposite.

`OPTION` can contain any valid ABAP comparison operator. Normally "EQ" (equal) is all we need.

`LOW` contains the value for the comparison, the currency code.

`HIGH` is left blank unless you are using the "BT" (between) or "NB" (not between) range operators, which would be unusual for this BAPI.

Let me give you a concrete example. If you are interested to find out which currencies USD can be converted to, you would put one row in the

Figure 17 Sample Data for the FROM_CURR_RANGE Parameter

Field Name	Contents
SIGN	"I"
OPTION	"EQ"
LOW	"USD"
HIGH	blank

FROM_CURR_RANGE parameter with the contents shown in **Figure 17**.

The fields of the EXCH_RATE_LIST table parameter are exactly the same as the ones listed in Figure 14. Both parameters are based on the same dictionary structure, but while the EXCH_RATE parameter of the GetDetail BAPI was a structure, the EXCH_RATE_LIST parameter of the GetCurrentRates BAPI is a table. So not only do we get a list of all currencies from or to which (depending on our selection criteria) we can convert, we also get the associated exchange rates.

This means that if you have called GetCurrentRates already (for instance, to display a drop-down list of all "To" currencies for a given "From" currency) you will not have to call GetDetail at all. If, on the other hand, you are sure that an exchange rate exists for a given currency pair, then the call to GetCurrentRates is not required.

Conclusion

We have started our discussion with an analysis of why SAP has decided not to use currency amount fields based on the CURR data type in BAPIs. We then learned how to use the BAPIs of the Currency object type to deal with amount fields that violate the rule and to find the number of decimals for a given currency code in order to display a currency amount properly.

Then we studied those BAPIs of the ExchangeRate object type that enable us to convert between different currencies.

As always, encapsulating what you have learned in this article into a nice, reusable component is a good idea. Contact me if you are interested in a ready-to-use component, or, if you have a rainy weekend to while away, build one yourself.

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 CA925 "Developing BAPI-enabled Web applications with Visual Basic" and CA926 "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.