Password Management for Extranet Applications

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 68)

An extranet application is defined as one in which you allow users outside your firewall limited, controlled access to certain functionality within the firewall. If this functionality involves SAP, the question arises whether you want to assign each user his own SAP userid or use a generic one. Why do we need an SAP userid at all? Before being able to call any BAPI (or other RFC-enabled Function Module), you have to connect (log on) to SAP using a valid userid/password. If you are willing to give each extranet user his own SAP userid, then you do not need this article. Normally, though, buying one SAP userid for each of your business partners (or, worse, one SAP userid for each of their employees who need to access your application) is not high on your list of ways to spend money.

You want to use a generic SAP userid instead. This obviously raises a new question: How do we control access to our extranet applications so that only selected business partners can use them? You can solve this issue completely independently of SAP, but SAP has foreseen this scenario and offers capabilities to manage special extranet userids for customers, vendors, etc.

This article will discuss three topics:

- The online SAPGUI transaction for maintaining extranet userids. This part is interesting for everybody involved in building extranet solutions, but specifically for security administrators.
- The password BAPIs available for object types like `Customer`, `Vendor`, etc. This part is for developers and architects involved in

building extranet applications. We will discuss the functionality required for a normal extranet application (primarily the ability to check a user's password) as well as a maintenance application for administrators (to be used instead of the SAPGUI transaction mentioned previously).

- A component written in Java that encapsulates access to the password BAPIs. This part will be most beneficial for Java developers, but if you use a different programming language, you should be able to translate the concepts into your own language. This component can be used in normal extranet applications as well as in administration tools.

Before we really get going, I would like to extend the term "extranet" for the purpose of this article. As indicated above, it usually refers to controlled access to an application from without the firewall. This applies mainly to business partners and employees who happen to be outside the firewall (e.g., because they are on a business trip). I would like to also include employees who do not have their own SAP userid, but reside within the firewall, in the definition.

So an extranet application would be one that requires a special userid/password (but not an SAP userid), regardless of whether the application is accessed from within or without the firewall. While this definition is not accurate outside the scope of this article, it allows us to use concise language to discuss our topics in the article.

Maintaining Extranet Userids in SAP

SAP since 3.1 offers special web transactions, initially called Internet Application Components (IACs), and recently renamed to Easy Web Transactions (EWTs). Some of them are true Internet applications in the sense that anybody on the Internet can use them without requiring a logon. Most EWTs, though, are either for employees or selected business partners, in other words they are extranet applications according to the definition valid for the scope of this article. For those EWTs, SAP needed a way to manage extranet userids and passwords without requiring an SAP userid for each user.

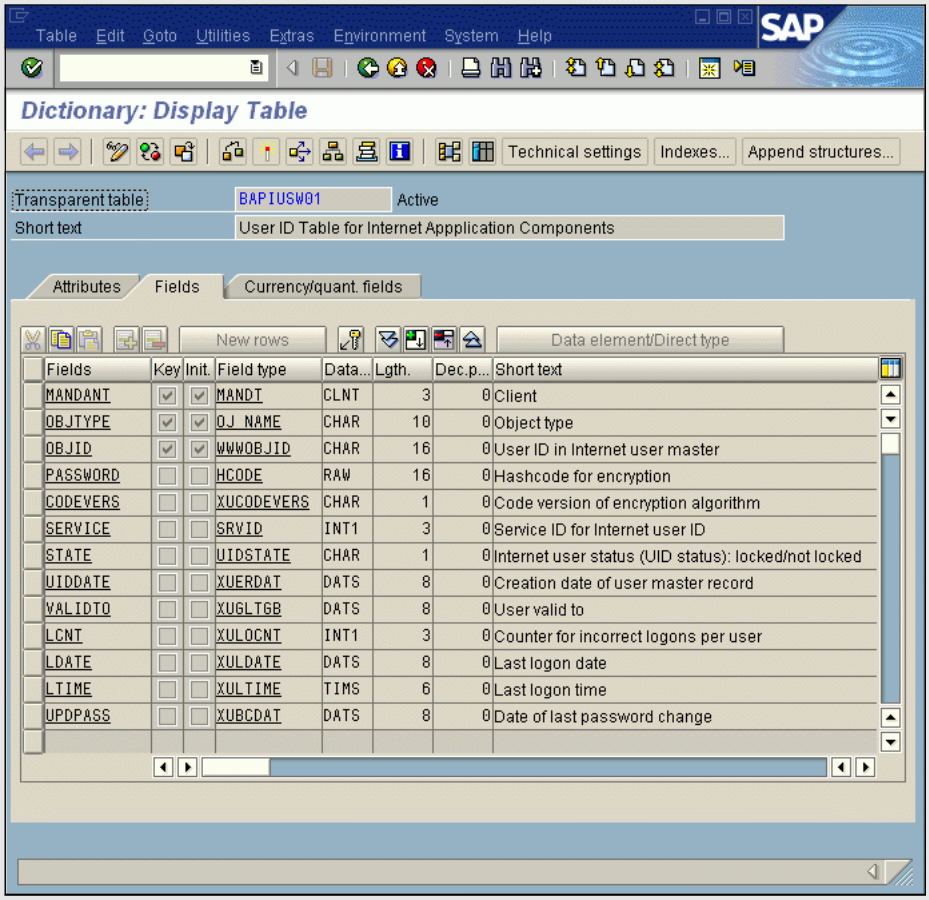
SAP added a new table to store the extranet userids, called `BAPIUSW01` (see **Figure 1**). This table has two key fields (in addition to the normal client key field): object type (`OBJTYPE`) and object id (`OBJID`). Object type identifies the type of userid, e.g., `KNA1` denotes a customer, `BUS1065` an employee. This allows all the extranet userids to be kept in one table. Object id is the object-type-specific key, for a customer this would be the customer number.

`BAPIUSW01` contains all sorts of useful information that we will discuss later when we learn about the `GetPassword` BAPI, but there is one field I would like to mention now, the one called `PASSWORD`. This, contrary to what the name seems to imply, does not contain the password, not even the password in an encrypted form that could be decrypted again. Instead, SAP calculates a hash code based on the password and stores that hash code in the table. The hashing algorithm is irreversible, so there is no way to find out the password for a user (unless you are willing to try out every possible password, which is rendered impossible by the fact that SAP locks the extranet userid after 12 consecutive incorrect logon attempts). But SAP can easily check an entered password by calculating the hash code for the entered password and comparing this hash code to the one stored in the table.

Passwords for extranet users are case-sensitive. They are a minimum of three and a maximum of 16 characters long and must obey the following rules:

- A password cannot be "sap", "SAP", "pass", or "PASS"; "SaP", on the other hand, is a legal password.
- The first three characters cannot be identical,

Figure 1 *Table BAPIUSW01*



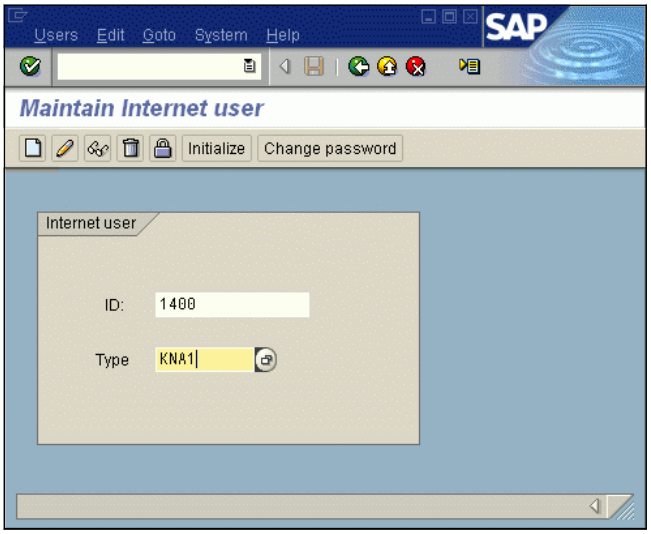
Fields	Key	Init.	Field type	Data...	Lgth.	Dec.p...	Short text
MANDANT	☑	☑	MANDT	CLNT	3	0	Client
OBJTYPE	☑	☑	OJ_NAME	CHAR	10	0	Object type
OBJID	☑	☑	WWWOBJID	CHAR	16	0	User ID in Internet user master
PASSWORD	☐	☐	HCODE	RAW	16	0	Hashcode for encryption
CODEVERS	☐	☐	XUCODEVERS	CHAR	1	0	Code version of encryption algorithm
SERVICE	☐	☐	SRVID	INT1	3	0	Service ID for Internet user ID
STATE	☐	☐	UIDSTATE	CHAR	1	0	Internet user status (UID status): locked/not locked
UIDDATE	☐	☐	XUERDAT	DATS	8	0	Creation date of user master record
VALIDTO	☐	☐	XUGLTGB	DATS	8	0	User valid to
LCNT	☐	☐	XULOCNT	INT1	3	0	Counter for incorrect logons per user
LDATE	☐	☐	XULDATE	DATS	8	0	Last logon date
LTIME	☐	☐	XULTIME	TIMS	6	0	Last logon time
UPDPASS	☐	☐	XUBCDAT	DATS	8	0	Date of last password change

e.g., “aaardvark” is not a legal password, but “Aaardvark” is.

- A password may not contain the “<” or space characters.
- A password cannot start with the “?” character.
- The first three characters of a password may not be substrings of the extranet userid. For instance, if the customer number is “0000001400”, a password of “014tgs” would be illegal, whereas “410tgs” would be okay.

The SAP transaction code used for the maintenance of extranet userids is `SU05` (see **Figure 2**).

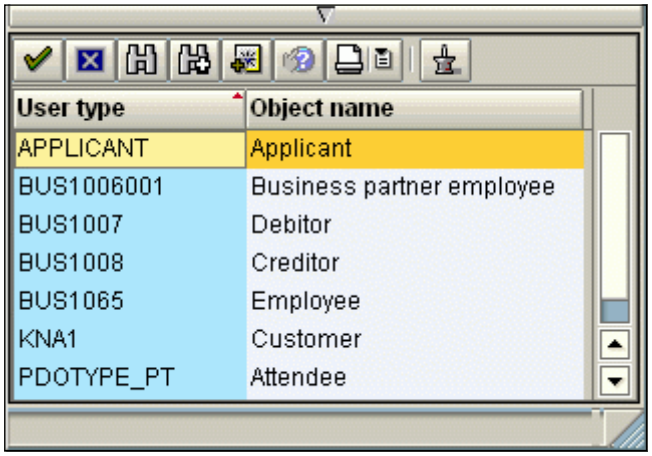
Figure 2 *Transaction Code SU05*



As you can see, SAP calls what I call an extranet userid an "Internet user". Both terms are slightly inaccurate, but I think mine is a little closer to the truth.

Which object types can have extranet userids in SAP? At least in 4.6B, there are two answers to that question. If you select input help on the "Type" field in Figure 2, you will see the list in **Figure 3**.

Figure 3 Supported Object Types in SU05



If, on the other hand, you create a list of all object types that have BAPIs to manipulate extranet userids, you end up (in 4.6B) with **Figure 4**.

If you compare the two lists, you will note that extranet userids for vendors are not supported by `SU05`, but they can be created by the appropriate BAPIs.

Let us now look at the functionality offered by `SU05`.

To **create** an extranet userid, you enter the object type and the object id (which will be the extranet userid) and select the "Create" menu item or icon. A dialog pops up (see **Figure 5**) that allows you to limit the validity period of the userid.

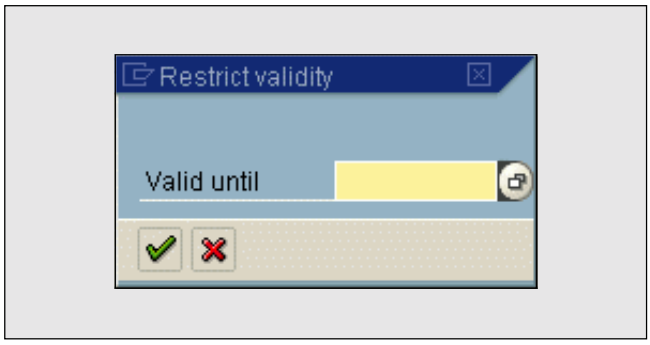
If you do not specify any date, the new userid will be valid forever.[1] The next pop-up (cf. **Figure 6**) shows the initial password assigned to the new user. You can either give this password to the user

[1] Or, more accurately, the userid will be valid until 9999-12-31. You already knew that SAP's software is not Y10K-compliant, didn't you?

Figure 4 Object Types With Password BAPIs

Object Type Key	Description	Object Type Name
APPLICANT	Applicant	Applicant
BUS1006001	Business partner employee	BusPartnerEmployee
BUS1007	Customer	Debtor
BUS1008	Vendor	Creditor
BUS1065	Employee	EmployeeAbstract
KNA1	Customer	Customer
LFA1	Vendor	Vendor
PDOTYPE_PT	Attendee	Attendee

Figure 5 Restricting the Validity Period of a Userid



(especially if you have reason to believe that he will still talk to you after having to enter the 16 characters shown in Figure 6) or immediately change the password to something else (see next page).

When creating a new extranet userid, `SU05` does not check whether the object id is valid, allowing you to create an extranet userid for a non-existing customer, for instance. The BAPIs are less forgiving. **Figure 7** contains information about whether the `CreatePassword` BAPI works for a non-existing object id in 4.6B.

Figure 6 The Initial Password

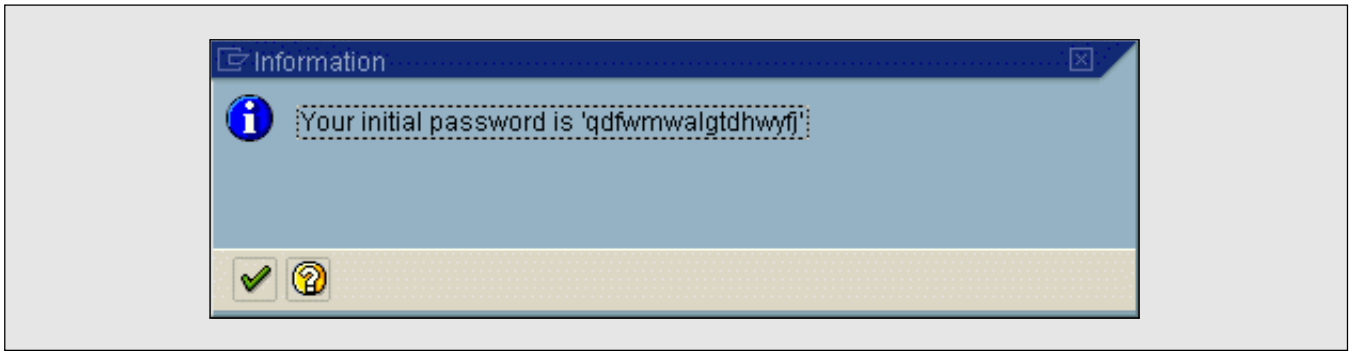


Figure 7 Are Object Ids Checked by the CreatePassword BAPI?

Object Type Key	Object Type Name	Checks Object Id
APPLICANT	Applicant	Yes
BUS1006001	BusPartnerEmployee	Yes
BUS1007	Debtor	Yes
BUS1008	Creditor	Yes
BUS1065	EmployeeAbstract	Yes
KNA1	Customer	No
LFA1	Vendor	Yes
PDOTYPE_PT	Attendee	N/A[2]

[2] The Attendee object type (PDOTYPE_PT) does not have a CreatePassword BAPI.

To **change** an extranet **password**, click the "Change password" button in Figure 2. The pop-up shown in **Figure 8** requires you to enter the new password twice to prevent typos. There are no additional rules for new passwords beyond the password rules discussed previously, specifically there is no requirement that the new password must differ from the old one. If you want to enforce additional rules, you have to implement them yourself.

If the userid is locked (see below), the password cannot be changed.

To **re-initialize** an extranet password, click the "Initialize" button in Figure 2. SAP will ask you to confirm this in the pop-up displayed in **Figure 9**.

If you click the "Yes" button, a new initial password will be generated and displayed in a pop-up as seen before in Figure 6.

If the userid is locked (see below), the password cannot be re-initialized.

To **change** the validity period of an extranet userid (or correct the userid itself), select the "Change" menu item or icon in Figure 2. The dialog shown in **Figure 10** allows you to make the necessary changes. The check box labeled "without restrict." can be used to remove a previously set valid-to date, as an alternative to entering the end-of-the-world date of 9999-12-31.

To **display** information about an extranet userid without making any changes, select the "Display" menu item or icon in Figure 2. As you can see in **Figure 11**, there is no validity period specified for this userid, but the userid is locked.

I have artificially created this situation by attempting to log on 12 times in a row with an invalid password.

To **lock** or **unlock** an extranet userid, select the "Lock/unlock" menu item or icon in Figure 2.

Figure 8 Changing a Password

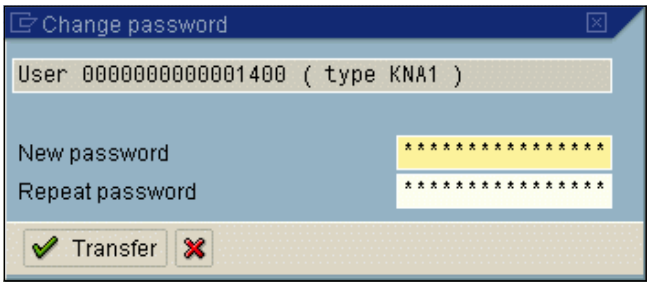


Figure 9 Confirming Password Initialization

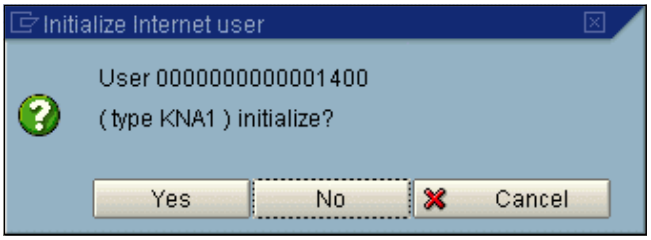


Figure 10 Changing an Extranet Userid

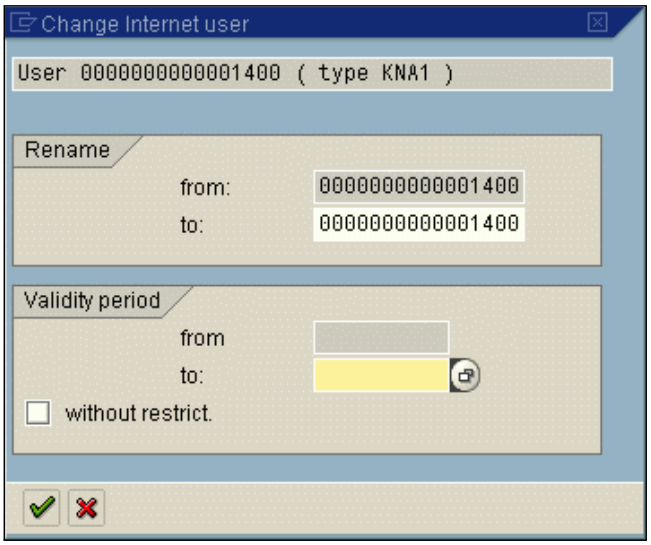


Figure 11 Displaying an Extranet Userid

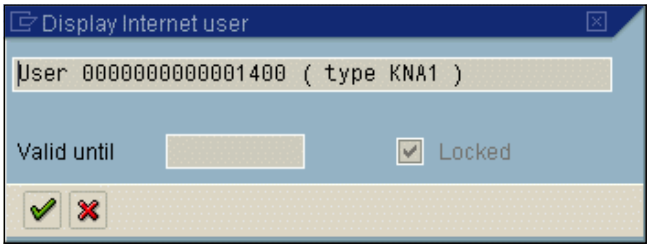


Figure 12 Unlocking a Userid

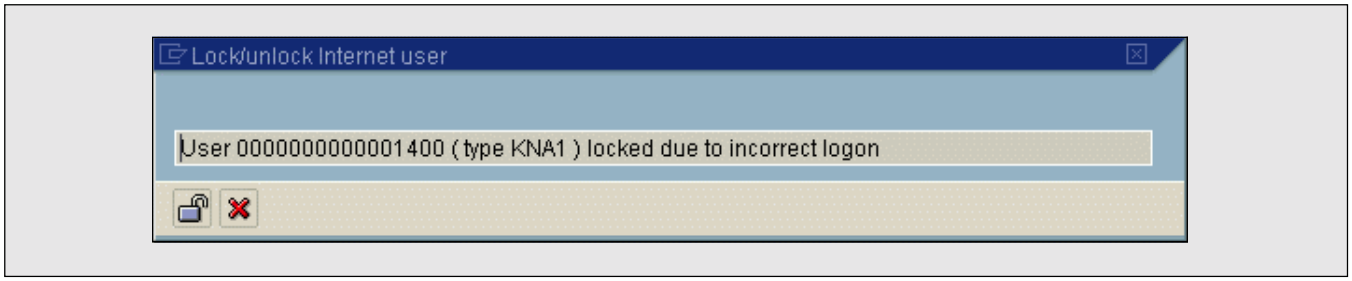
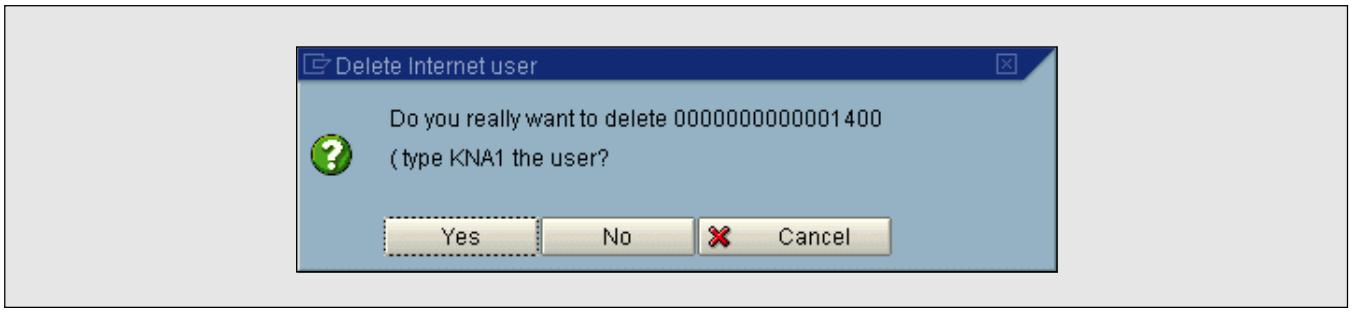


Figure 13 Confirming the Deletion of a Userid



The message in **Figure 12** tells you that the userid was actually locked due to too many invalid logon attempts. Click the unlock icon to unlock the userid again. In real life, you would now probably also assign a new password and contact the user to figure out whether he had forgotten his password. Otherwise, we should alert our network security people so that they can try to find out who has been trying to hack into our system.

Finally, it is also possible to **delete** an extranet userid in `SU05`. Selecting the "Delete" menu item or icon in Figure 2 will cause a window like the one in **Figure 13** to show up.

Let me summarize what we have learned about `SU05`: This transaction code allows us to create, display, lock, unlock, and delete extranet userids. We can change the validity period of a userid, as well as change or re-initialize the associated password. At least in 4.6B, the `Vendor` object type is not supported in `SU05`, and — as opposed to most of the equivalent BAPIs — object ids are not validated.

Using the Password BAPIs

When developing an extranet application that wants to take advantage of SAP extranet userid capabilities, we obviously need at least a BAPI to check a password entered by the user. SAP has been nice enough to give us more than just that: There is a whole group of BAPIs that can be used to deal with extranet userids and passwords. All these BAPIs contain the string "Password" so I will call them the password BAPIs hereinafter. The functionality offered by these BAPIs overlaps with the functionality available in `SU05`. There are things you can do with the BAPIs that cannot be done in `SU05` and vice versa. One example was discussed above: To create an extranet userid for a vendor, `SU05` does not work, but you can use BAPIs instead. On the other hand, there is no BAPI to lock a userid (unless you resort to the trick of calling `CheckPassword` 12 times in a row with incorrect passwords).

Using the password BAPIs for the first time is not trivial. SAP's naming conventions are such that you

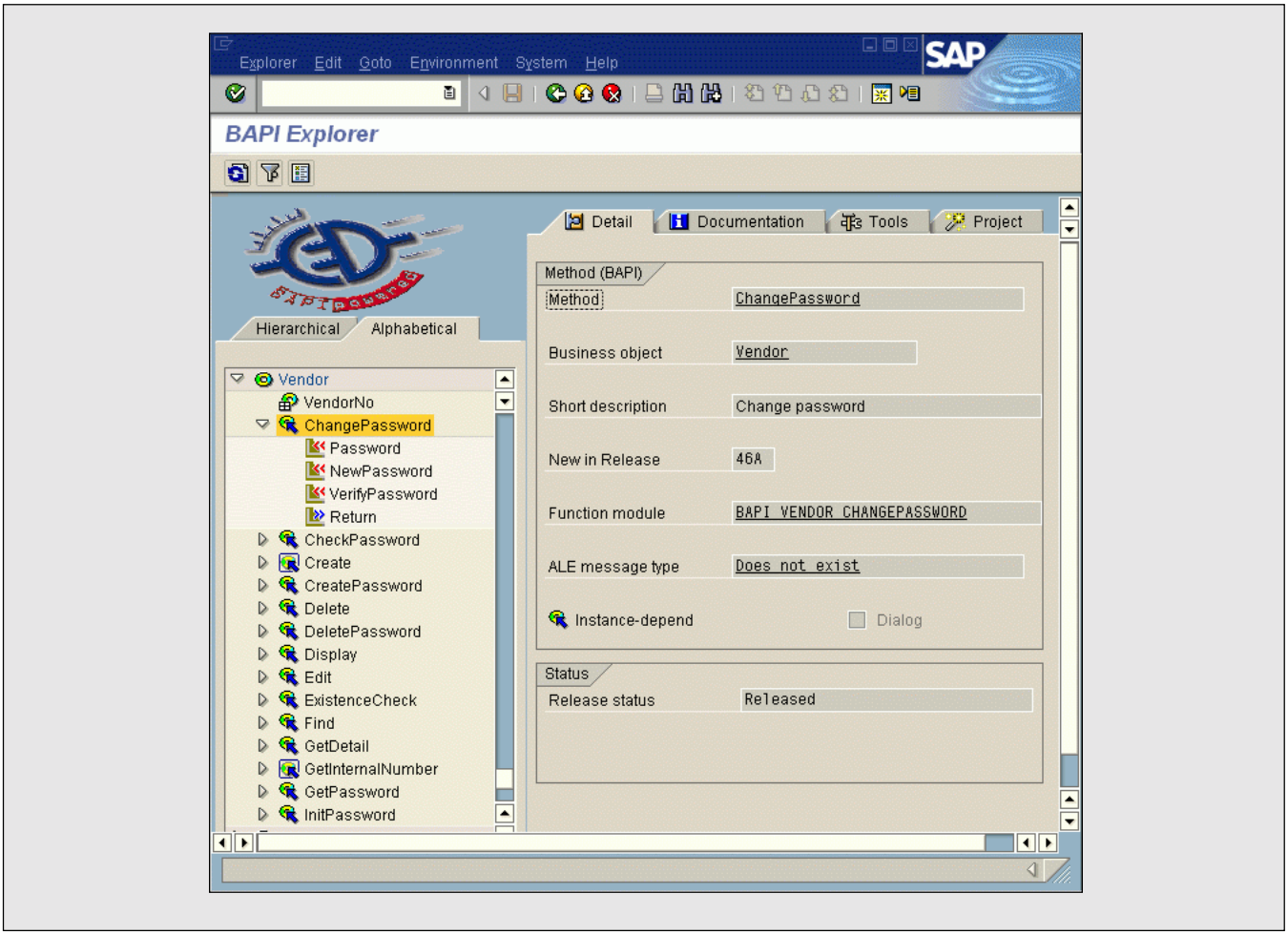
Figure 14 The Standard Password BAPIs

BAPI Name	Description
ChangePassword	Changes a password.
CheckPassword	Checks whether a password is correct.
CreatePassword	Creates an extranet userid in the database, but does not create the actual password.
DeletePassword	Deletes an extranet userid from the database.
GetPassword	Returns status information, but not the password itself.
InitPassword	Creates an initial password; requires an extranet userid created via CreatePassword.

Figure 15 Object Types with Password BAPIs

Object Type Name	Object Type Key	Description	Standard Password BAPIs	Remarks
Applicant	APPLICANT	Applicant	Yes	
Attendee	PDOTYPE_PT	Attendee	No	Supports only ChangePassword and CheckPassword.
BusPartnerEmployee	BUS1006001	Business partner employee	Yes	
Creditor	BUS1008	Vendor	No	Obsolete since 4.6A; uses non-standard names for CreatePassword, DeletePassword, and GetPassword.
Customer	KNA1	Customer	Yes	New version of the CheckPassword BAPI since 4.6A.
Debtor	BUS1007	Customer	No	Obsolete since 4.6A; uses non-standard names for CreatePassword, DeletePassword, and GetPassword.
EmployeeAbstract	BUS1065	Employee	Yes	Additionally, all its subclasses (e.g., Employee, EmployeeCH).
Vendor	LFA1	Vendor	Yes	New since 4.6A.

Figure 16 The Vendor Object Type in the BAPI Explorer



are easily confused as to the purpose of a specific BAPI. **Figure 14** contains the standard BAPIs supported by most object types that have password BAPIs at all.

And when I tell you that `CreatePassword` does lots of things, but does not create a password, and that `GetPassword` returns very useful information, but never the password, you may appreciate that this requires a little more study.

So, let us get busy…

First, I want to give you an extended version of Figure 4 (see **Figure 15**).

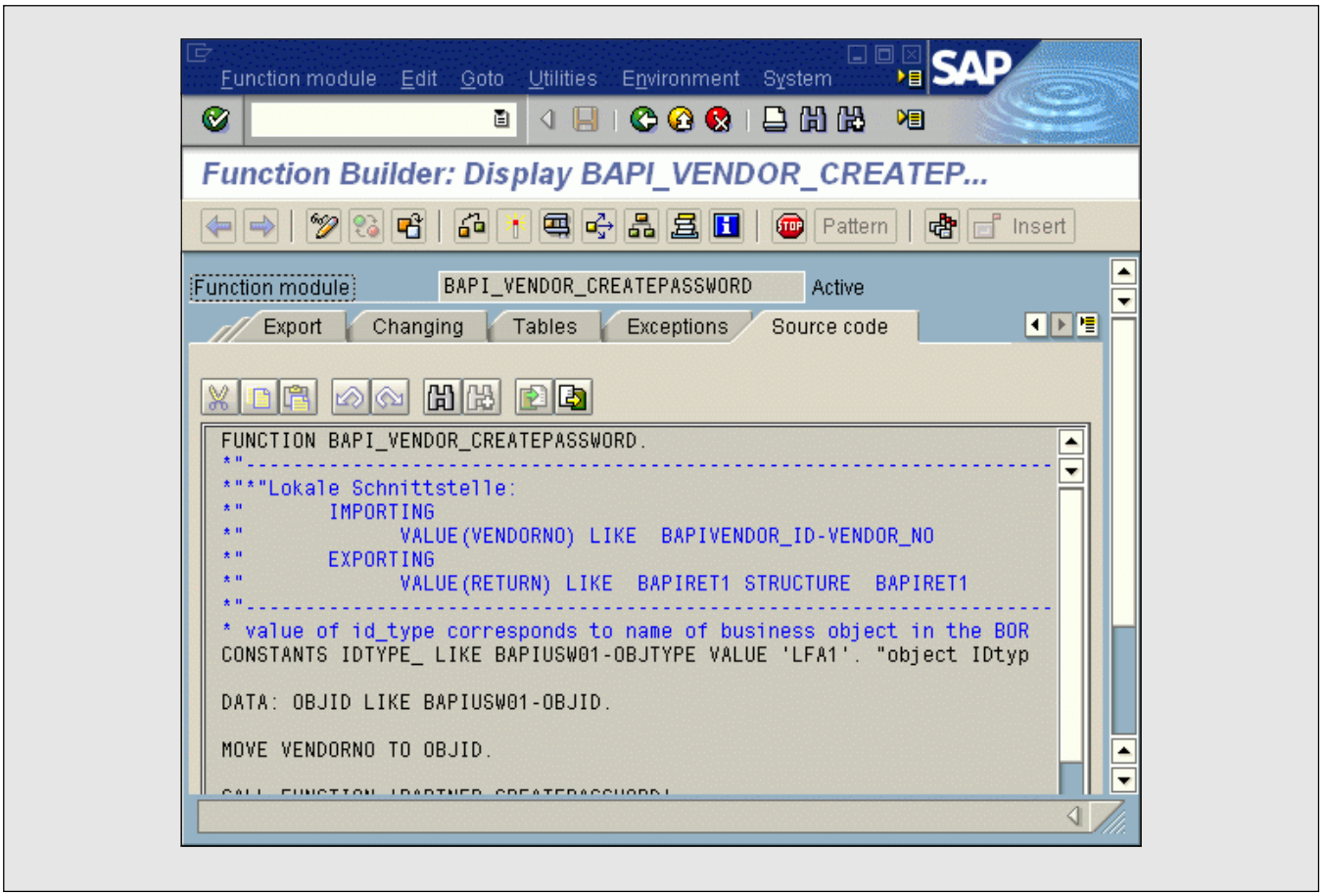
`Applicant`, `BusPartnerEmployee`, `Customer`, `EmployeeAbstract` (and its subclasses), and `Vendor` have complete support for the standard password BAPIs. See the remarks column in Figure 15 to find out how the other object types differ.

Figure 16 is a screenshot that lists all BAPIs for the `Vendor` object type in the BAPI Explorer. You can see that all standard password BAPIs are supported. The BAPI Explorer (transaction code `BAPI`) is the best starting point if you want to study and try out any BAPI, so you may want to use it while reading the remaining paragraphs of this section.

Figure 17 CreatePassword Standard Parameters

Parameter	Data Type	In/Out	Description
Return	Structure	Out	The standard BAPI return parameter

Figure 18 The RFM for Vendor.CreatePassword



Next, we want to discuss the standard password BAPIs one by one, starting with **`CreatePassword`**. Contrary to its name, this BAPI does not create a password, but it creates an extranet userid. **Figure 17** contains all the parameters, in this case exactly one, the standard BAPI return parameter that informs us about success or failure of our BAPI call. Where do we specify the object id (e.g., the vendor number)? Remember (or if you do not remember, find out now) that the BAPI Explorer does not show key field parameters for instance[3] BAPIs, instead it lists all key fields for the object type directly underneath the object name (cf. Figure 16, key field "VendorNo" right above the highlighted "ChangePassword" BAPI).

[3] An instance BAPI is one that requires key fields to access a particular record in a database table.

Figure 19 InitPassword Standard Parameters

Parameter	Data Type	In/Out	Description
Password	Character 16	Out	Generated password
Return	Structure	Out	The standard BAPI return parameter

Figure 20 ChangePassword Standard Parameters

Parameter	Data Type	In/Out	Description
Password	Character 16	In	Old password
NewPassword	Character 16	In	New password
VerifyPassword	Character 16	In	New password again
Return	Structure	Out	The standard BAPI return parameter

Figure 18 is a screenshot of the RFM (RFC-enabled Function Module) that implements the `CreatePassword` BAPI for the `Vendor` object type. And there you can see that it has indeed the expected import parameter for the vendor number (which will become the userid of the extranet user).

If the return code from the `CreatePassword` BAPI (in parameter `Return`) indicates success, we have created a new extranet userid, but not yet assigned a password (whereas in `SU05`, creating a userid automatically initializes the password).

In order to (re-)initialize the password, we use the **`InitPassword`** BAPI, the parameters of which are listed in **Figure 19**.

The `Return` parameter tells us whether the call succeeded, `Password` contains the freshly generated password. In addition to generating a (new) initial password, `InitPassword` also unlocks the extranet userid if it was locked before. This is different from the behavior in `SU05`, where we cannot re-initialize the password of a locked userid without unlocking it first.

The initial password is not something a user will want to remember, so a normal application will include a dialog for the user to change the password. This is accomplished using the **`ChangePassword`** BAPI. Its parameters are enumerated in **Figure 20**. `ChangePassword` will fail for a locked userid.

The BAPI obviously assumes that you require the user to type the new password twice, which is a good way of preventing typos from creating a different new password than the user expected. Note that the old password must be specified here, whereas `InitPassword` did not require that. The reason is that `ChangePassword` provides functionality that is used in normal applications, whereas resetting a password is usually restricted to administrators. Also, since there is no way to figure out somebody's password, we would not be able to assign somebody a new password once he had forgotten the current one, if that required us to know the old password.

The most commonly used password-related function in an application is to check a password entered by a user. **`CheckPassword`** allows us to do

Figure 21 CheckPassword Standard Parameters

Parameter	Data Type	In/Out	Description
Password	Character 16	In	Password to be checked
Return	Structure	Out	The standard BAPI return parameter

Figure 22 GetPassword Standard Parameters

Parameter	Data Type	In/Out	Description
StatusInfo	Structure	Out	Status information (see Figure 23)
Return	Structure	Out	The standard BAPI return parameter

Figure 23 StatusInfo Fields

Field Name	Data Type	Description
OBJTYPE	Character 10	Object type
OBJID	Character 16	User ID in Internet user master
SERVICE	1-byte Integer	Service ID for Internet user ID
STATE	Character 1	Internet user status (UID status): locked/not locked
UIDDATE	Date	Creation date of user master record
VALIDTO	Date	User valid to
LCNT	1-byte Integer	Counter for incorrect logons per user
LDATE	Date	Last logon date
LTIME	Time	Last logon time
UPDPASS	Date	Date of last password change

that (see **Figure 21** for the parameters). The password to be checked is passed in parameter `Password`, `Return` tells us whether the entered password was correct.

To find out about the status of an extranet userid, including whether it is locked and when the password was changed last, we use the **`GetPassword`** BAPI. **Figure 22** shows its parameters. All the interesting

Figure 24 DeletePassword Standard Parameters

Parameter	Data Type	In/Out	Description
Return	Structure	Out	The standard BAPI return parameter

information is returned in table parameter `StatusInfo`, the structure of which is given in **Figure 23**.

`OBJTYPE` and `OBJID` should be obvious by now. What `SERVICE` is used for, I cannot tell you, probably it has some meaning for ITS applications.

`STATE` is blank to indicate an unlocked userid, non-blank for a locked userid. `UIDDATE` contains the date when the extranet userid was created. `VALIDTO` is either empty (indicating unlimited validity) or contains the end of the validity period. `LCNT` contains the number of consecutive invalid logon attempts. This field is reset after a successful logon and after the password is (re-)initialized. `LDATE` and `LTIME` contain the date and time, respectively, of the last logon. `UPDPASS` tells us when the password was changed last. This could be used in an application that wants to force a user to change his password after a certain interval in order to provide more security.[4]

Finally, there is a BAPI to delete an extranet userid, **`DeletePassword`**. Its parameters are contained in **Figure 24**.

Let us summarize this part of the article: Once you have recovered from the shock that some of the password BAPIs have inappropriate names, it is not so difficult to use them. `CheckPassword` will be used in all your extranet applications, `ChangePassword` whenever you want to allow the users to change their passwords. `GetPassword` returns information relevant for advanced functionality like forcing users to change their passwords periodically.

`CreatePassword`, `InitPassword`, and `DeletePassword` will normally only be used in administrative tools.

Building a Java Component for the Password BAPIs

Whenever something takes a few hours or more to figure out, and you assume that the functionality might be useful for other projects, you should try to build a reusable component that hides the complexity. For this article I have chosen Java as the programming language, because it facilitates the development of reusable components.

I had the following goals in mind when designing this component:

- All SAP object types that implement all standard password BAPIs should be supported.
- Adding additional object types in the future should be very easy.
- The idiosyncrasies of the SAP password BAPIs should be hidden.
- The component should be extremely easy to use and require as little SAP knowledge as possible.

In order to make it simple to add new specific userid classes, everything that can be done generically is done in class `UserId`. But this base class cannot know the names of the RFMs implementing the

[4] In reality, sometimes the opposite happens: A user forced to change passwords periodically is more likely to write the passwords on a little piece of paper hidden under the mouse pad than somebody who is allowed to select a password once and keep it. Security is a difficult business!

various password BAPIs for the different SAP object types. Also, in some cases the RFM parameter names used are different for different object types. Each subclass needs to provide the necessary information by implementing the abstract methods of class `UserId` shown in **Figure 25**.

Figure 26 is a listing of the complete source code for one of the subclasses, `CustomerUserId`. The code for `ApplicantUserId`, `BusPartnerEmployeeUserId`, `EmployeeUserId`, and `VendorUserId` looks very similar.

As you can see the code is pretty straightforward, each method just returns the required string. The only logic takes place in method `getCheckPasswordMethodName()` since there are two versions of the appropriate BAPI, one for systems below 4.6 and a new one available since 4.6. The code simply checks the release of the connected SAP system and returns the correct name.

As stated before, all real work takes place in class `UserId`. This class defines four private fields (**Figure 27**).

The first three of these will be set when the constructor of the class is called, the last one will be discussed later in the context of the `getUserIdInformation()` method.

The constructor for class `UserId` is shown in **Figure 28**.

Figure 25 Abstract Methods of Class UserId

```
protected abstract String getCheckPasswordMethodName();
protected abstract String getChangePasswordMethodName();
protected abstract String getCreatePasswordMethodName();
protected abstract String getDeletePasswordMethodName();
protected abstract String getGetPasswordMethodName();
protected abstract String getInitPasswordMethodName();
protected abstract String getNewPasswordParameterName();
protected abstract String getVerifyPasswordParameterName();
```

Figure 26 Source Code for Class CustomerUserId

```
package de.arasoft.sap.jco.password;
import com.sap.mw.jco.*;
public class CustomerUserId extends UserId {
  public CustomerUserId (String objectKey,
                         JCO.Client connection,
                         IRepository repository)
         throws UserIdException {
    super(objectKey, connection, repository);
  }
  protected String getObjectKeyName() {
    return "CUSTOMERNO";
  }
  protected String getChangePasswordMethodName() {
    return "BAPI_CUSTOMER_CHANGEPASSWORD";
```


Figure 26 (continued)

```
  }
  protected String getCheckPasswordMethodName() {
    if (getConnection().getAttributes().
          getPartnerRelease().compareTo("46A") < 0)
      return "BAPI_CUSTOMER_CHECKPASSWORD";
    else
      return "BAPI_CUSTOMER_CHECKPASSWORD1";
  }
  protected String getCreatePasswordMethodName() {
    return "BAPI_CUSTOMER_CREATEPWREG";
  }
  protected String getDeletePasswordMethodName() {
    return "BAPI_CUSTOMER_DELETEPWREG";
  }
  protected String getGetPasswordMethodName() {
    return "BAPI_CUSTOMER_GETPWREG";
  }
  protected String getInitPasswordMethodName() {
    return "BAPI_CUSTOMER_INITPASSWORD";
  }
  protected String getNewPasswordParameterName() {
    return "NEW_PASSWORD";
  }
  protected String getVerifyPasswordParameterName() {
    return "VERIFY_PASSWORD";
  }
}
```

Figure 27 Private Fields for Class UserId

```
private String objectKey = null;
private JCO.Client connection = null;
private IRepository repository = null;
private UserIdInformation pi = null;
```

Figure 28 Constructor for Class UserId

```
protected UserId (String objectKey,
                  JCO.Client connection,
                  IRepository repository) throws UserIdException {
  this.objectKey = objectKey;
  this.connection = connection;
  this.repository = repository;
}
```

Figure 29 The create() method of Class UserId

```
synchronized public String create () throws UserIdException {
  JCO.Function function =
    createFunction(this.getCreatePasswordMethodName());
  if (function == null)
    throw new UserIdException("Required RFM " +
                              this.getCreatePasswordMethodName() +
                              " not found.");
  try {
    function.getImportParameterList().
      setValue(objectKey, getObjectKeyName());
    getConnection().execute(function);
    JCO.Structure bapiReturn =
      function.getExportParameterList().getStructure("RETURN");
    if (bapiReturn.getName().equals("BAPIRETURN")) {
      if ( ! ( bapiReturn.getString("TYPE").equals("") ||
               bapiReturn.getString("TYPE").equals("S") ||
               bapiReturn.getString("CODE").equals("S>501")
             ) )
        throw new
          UserIdException(getBapiReturnErrorMessage(bapiReturn));
    } else {
      if ( ! ( bapiReturn.getString("TYPE").equals("") ||
               bapiReturn.getString("TYPE").equals("S") ||
               bapiReturn.getString("NUMBER").equals("501")
             ) )
        throw new
          UserIdException(getBapiReturnErrorMessage(bapiReturn));
    }
  }
  catch (UserIdException ex) {
    throw ex;
  }
  catch (Exception ex) {
    throw new UserIdException(ex.getMessage(), ex);
  }
  function = createFunction(this.getInitPasswordMethodName());
  if (function == null)
    throw new UserIdException("Required RFM " +
                              this.getInitPasswordMethodName() +
                              " not found.");
  try {
    function.getImportParameterList().
      setValue(objectKey, getObjectKeyName());
    getConnection().execute(function);
    JCO.Structure bapiReturn =
      function.getExportParameterList().getStructure("RETURN");
    if ( ! ( bapiReturn.getString("TYPE").equals("") ||
             bapiReturn.getString("TYPE").equals("S")
           ) )
```


Figure 29 (continued)

```
        throw new UserIdException(getBapiReturnErrorMessage(bapiReturn));
      return function.getExportParameterList().getString("PASSWORD");
    }
    catch (UserIdException ex) {
      throw ex;
    }
    catch (Exception ex) {
      throw new UserIdException(ex.getMessage(), ex);
    }
  }
```

The `objectKey` parameter is the key identifying the entity, for a customer this would be the customer number. This key is the extranet userid. The formatting of the key must be appropriate for the specific subclass. When calling the constructor of subclass `CustomerUserId`, for example, an all-numeric customer number must be left-padded with zeroes.

The `connection` parameter is used to pass an active connection to SAP, required so that the object can call BAPIs in SAP.

The `repository` parameter must contain a reference to a JCo[5] repository object connected to the same SAP system. The repository dynamically retrieves the required metadata for the BAPIs.

The creation of a new extranet userid and the initialization of its password is done via a call to `create()` (see **Figure 29**) in our component. This method makes two BAPI calls, one for the creation and one for the password initialization. We have combined the two BAPIs into one method since it is not very useful to create an extranet userid without giving it a password. The generated password is returned by the `create()` method.

What would happen if we called `create()` for a userid that already exists? In our implementation, we simply continue and (re-)initialize the password. This can be useful if a user has forgotten his password and an administrator, using a GUI built on top of our component, needs to assign a new one.

The next method we need to implement is `changePassword()`. The initial password generated by `create()` is impossible to remember for most users. We can either change the initial password immediately after the creation and give the changed password to the user or give the user the initial password and an option to change it when he logs on for the first time. Anyway, we need `changePassword()`, as shown in **Figure 30**.

[5] The SAP Java Connector (or JCo) is the standard middleware for Java connectivity to SAP. You can download it from **http://service.sap.com/connectors**.

Figure 30 The changePassword() method of Class UserId

```
synchronized public void changePassword (String oldPassword,
                                         String newPassword)
                         throws UserIdException {
  JCO.Function function =
    createFunction(this.getChangePasswordMethodName());
```

(continued on next page)

Figure 30 (continued)

```
    if (function == null)
      throw new UserIdException("Required RFM " +
                                this.getChangePasswordMethodName() +
                                " not found.");
    try {
      function.getImportParameterList().
        setValue(objectKey, getObjectKeyName());
      function.getImportParameterList().
        setValue(oldPassword, "PASSWORD");
      function.getImportParameterList().
        setValue(newPassword, this.getNewPasswordParameterName());
      function.getImportParameterList().
        setValue(newPassword, this.getVerifyPasswordParameterName());
      getConnection().execute(function);
      JCO.Structure bapiReturn =
        function.getExportParameterList().getStructure("RETURN");
      if ( ! ( bapiReturn.getString("TYPE").equals("") ||
               bapiReturn.getString("TYPE").equals("S")
             ) )
        throw new UserIdException(getBapiReturnErrorMessage(bapiReturn));
    }
    catch (UserIdException ex) {
      throw ex;
    }
    catch (Exception ex) {
      throw new UserIdException(ex.getMessage(), ex);
    }
  }
```

The code is not very exciting. Just remember that SAP's change password BAPIs want the same password twice. Our `changePassword()` method assumes that the client has verified that the user has entered the exact same new password twice and therefore requires only one parameter with the new password.

The most important function for a normal application is the ability to check a user's password. This is accomplished in our `isPasswordCorrect()` method (see **Figure 31**).

In order to facilitate access to the information made available by the `GetPassword` BAPI, we have built the `getUserIdInformation()` method (source code listed in **Figure 32**). This method has a parameter that lets the client program

Figure 31 The isPasswordCorrect() method of Class UserId

```
  synchronized public boolean isPasswordCorrect (String passWord)
                             throws UserIdException {
    JCO.Function function =
      createFunction(this.getCheckPasswordMethodName());
    if (function == null)
```


Figure 31 (continued)

```
    throw new UserIdException("Required RFM " +
                              this.getCheckPasswordMethodName() +
                              " not found.");
  try {
    function.getImportParameterList().setValue(passWord, "PASSWORD");
    function.getImportParameterList().
      setValue(objectKey, getObjectKeyName());
    getConnection().execute(function);
    JCO.Structure bapiReturn =
      function.getExportParameterList().getStructure("RETURN");
    return bapiReturn.getString("TYPE").equals("") ||
          bapiReturn.getString("TYPE").equals("S") ? true : false;
  }
  catch (Exception ex) {
    throw new UserIdException(ex.getMessage(), ex);
  }
}
```

Figure 32 The getUserIdInformation() method of Class UserId

```
synchronized public UserIdInformation
            getUserIdInformation(boolean refresh)
            throws UserIdException {
  if ( (! refresh) && (pi != null) ) return pi;
  JCO.Function function =
    createFunction(this.getGetPasswordMethodName());
  if (function == null)
    throw new UserIdException("Required RFM " +
                              this.getGetPasswordMethodName() +
                              " not found.");
  try {
    function.getImportParameterList().
      setValue(objectKey, getObjectKeyName());
    getConnection().execute(function);
    JCO.Structure bapiReturn =
      function.getExportParameterList().getStructure("RETURN");
    if ( ! ( bapiReturn.getString("TYPE").equals("") ||
             bapiReturn.getString("TYPE").equals("S")
           ) )
      throw new UserIdException(getBapiReturnErrorMessage(bapiReturn));
    JCO.Table status =
      function.getTableParameterList().getTable("STATUSINFO");
    if (status.getNumRows() != 1)
      throw new UserIdException("Incorrect status info returned.");
    status.setRow(0);
    Date validToDate = status.getDate("VALIDTO");
```

(continued on next page)

Figure 32 (continued)

```
    if (validToDate == null)
      validToDate = new GregorianCalendar(9999, 11, 31).getTime();
    pi = new UserIdInformation(
              status.getString("OBJTYPE"),
              status.getInt("SERVICE"),
              status.getString("STATE").equals("") ? false : true,
              status.getDate("UIDDATE"),
              validToDate,
              status.getInt("LCNT"),
              UserId.combineDateAndTime(
                status.getDate("LDATE"), status.getDate("LTIME")),
              status.getDate("UPDPASS")
         );
  }
  catch (UserIdException ex) {
    throw ex;
  }
  catch (Exception ex) {
    throw new UserIdException(ex.getMessage(), ex);
  }
  return pi;
}
```

decide whether we should obtain fresh information from SAP, or whether the information obtained before is still sufficient. This allows us to improve the performance by not making unnecessary SAP calls, but also to provide up-to-date information when required.

The information returned by `getUserIdInformation()` is encapsulated in an object of class `UserIdInformation`, as shown in **Figure 33**. The individual properties were discussed earlier in this article.

Figure 33 Class UserIdInformation

```
public class UserIdInformation extends Object {
  private String objectType;
  private int serviceID;
  private boolean isLocked;
  private Date created;
  private Date validTo;
  private int incorrectLogons;
  private Date lastLogon;
  private Date passwordChanged;
  UserIdInformation( String objectType,
                     int serviceID,
                     boolean isLocked,
                     Date created,
                     Date validTo,
                     int incorrectLogons,
```


Figure 33 (continued)

```
                    Date lastLogon,
                    Date passwordChanged ) {
    this.objectType = objectType;
    this.serviceID = serviceID;
    this.isLocked = isLocked;
    this.created = created;
    this.validTo = validTo;
    this.incorrectLogons = incorrectLogons;
    this.lastLogon = lastLogon;
    this.passwordChanged = passwordChanged;
  }
  public String getObjectType() {
    return objectType;
  }
  public int getServiceID() {
    return serviceID;
  }
  public boolean isLocked() {
    return isLocked;
  }
  public Date getCreated() {
    return created;
  }
  public Date getValidTo() {
    return validTo;
  }
  public int getIncorrectLogons() {
    return incorrectLogons;
  }
  public Date getLastLogon() {
    return lastLogon;
  }
  public Date getPasswordChanged() {
    return passwordChanged;
  }
  public String toString() {
    return "ObjectType: \t" + getObjectType() + "\n"
           + "ServiceID: \t" + String.valueOf(getServiceID()) + "\n"
           + "IsLocked: \t" + String.valueOf(isLocked()) + "\n"
           + "Created: \t" + getCreated().toString() + "\n"
           + "ValidTo: \t" + getValidTo().toString() + "\n"
           + "IncorrectLogons: \t"
           + String.valueOf(getIncorrectLogons()) + "\n"
           + "LastLogon: \t" + getLastLogon().toString() + "\n"
           + "PasswordChanged: \t"
           + getPasswordChanged().toString() + "\n"
           ;
  }
}
```

The code for the `delete()` method is trivial to build, refer to our discussion of the `DeletePassword` BAPI earlier.

If you are interested in obtaining an up-to-date version of a jar file with all the classes discussed in this article, please send an e-mail to the author. And, as always, have fun using the information from this article!

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.

