Using the User Management API for SAP NetWeaver '04
Course Objectives

After completing this session, you will be able to use the User Management API to:

- Obtain information about users
- Search for users
- Create new users
- Delete users
- Search for groups and/or roles
- Add members to groups and/or roles
User Management Engine: API

Applications
Accessing User Management

User Management
Core Layer

J2EE Application
SAP Enterprise Portal
Web Dynpro Application

User API
User Account API
Group API
Role API
Persistence Manager
Replication Manager

Persistence Manager

Database
LDAP Directory
SAP System
External System

User Management Persistence Adapters
User Management Engine API

A common API provides uniform access to various objects:

**Package** `com.sap.security.api`
User Management and Authorization
- `UMFactory`
- `IUser`
- `IUserAccount`
- `IGroup`
- `IRole`

**Package** `com.sap.security.api.acl`
Access Control List Management
- `IAcl`
- `IAclManager`

Objects are obtained from their corresponding factory (e.g. Users are obtained from the User Factory)
The User Management API must be made available to portal components which wish to use UM functions. This is automatically provided by NW04 (based on WebAS 6.40).

References to the jar file `com.sap.security.api.jar` should be made in your development project. This is automatically provided by the Portal plugin.
For EP6.0 Releases prior to NW04 you need to add a SharingReference to "usermanagement" in the deployment descriptor (portalapp.xml)

```xml
<application>
    <application-config>
        <property name="SharingReference" value="com.sap.portal.htmlb, usermanagement"/>
        <property name="releasable" value="true"/>
    </application-config>
    <components>
        ...
    </components>
</application>
```
## UME Objects

**IPrincipal**  
The primary interface for most objects that you will need to work with. Many UME objects extend the IPrincipal interface: IGroup, IPrincipalMaint, IPrincipalSet, IRole, IUUser, IUUserAccount, IUUserMaint.  
*This interface can often be used when you need a standard method to deal with users, groups, or roles.*

**IUUser**  
This interface provides read-access to the user's attributes, and offers basic support for authorization checking.

**IGroup**  
This interface provides read and write access to principals groups.

**IRole**  
This interface provides read and write access to attributes and properties of instances of roles.
Applications access all the functionality of UME through the class `com.sap.security.api.UMFactory`

The `UMFactory` is the central object from which all UME object factories are obtained.

You use the `UMFactory` object to obtain the appropriate factory for user, group, or role objects via the methods:
- `getUserFactory()`
- `getGroupFactory()`
- `getRoleFactory()`

```java
String uid = "demouser01";
String groupID = "GRUP.CORP_LDAP.cn=developers,ou=groups,dc=sap,dc=corp"

IUser user = UMFactory.getUserFactory().getUserByLogonID(uid);
IGroup grp = UMFactory.getGroupFactory().getGroup(groupID);
```
Unique IDs in the UME World

- The User Management Engine can aggregate users from many different data sources (e.g. several directory servers, R/3 system, and a database).

- UME objects are identified by a **Unique ID** which concatenates the type, data source and the key to locate the object.

Examples:

- USER.CORP_LDAP.uid=i802895,ou=people,dc=sap,dc=corp
- GRUP.CORP_LDAP.cn=developers,ou=groups,dc=sap,dc=corp
- GRUP.SUPER_GROUPS_DATASOURCE.EVERYONE
- ROLE.PCD_ROLE_PERSISTENCE.VvlvkEGjiW9zPFaxR/4pd2/bX5Q=
- USER.PRIVATE_DATASOURCE.un:admin
The User factory **IUserFactory** provides means to:

- Instantiate user objects
- Create new users (possibly by copying the data of an existing one)
- Delete users
- Search for users based on different criteria
- Perform mass commit/rollback operations on a set of users

Access to the User Factory can be obtained by calling:

```java
import com.sap.security.api.*;

IUserFactory userFact = UMFactory.getUserFactory();
```
The **IUserFactory** is the factory used to obtain **IUser** objects.

Methods are provided to instantiate user objects given a logon ID or Unique ID

```
IUserFactory userFact = UMFactory.getUserFactory();

getUserFactory().getUser(String uniqueID)
  Gets the user object with the given unique ID

getUserFactory().getUserByLogonID(String logonID)
  Gets the user object with the given logon ID

getUserFactory().getUser(String uniqueID, AttributeList attrList)
  Gets the user object with the given unique ID and pre-populates the specified attributes (e.g. department, etc.)
```
IUser Object

- The **IUser** object contains most the information that you probably need regarding a user.

- Information about the name of the user, their unique ID, LDAP attributes, display name, role membership, etc are available from the **IUser** object.

- If you have a user object (either from the authenticated user or any other user), you can use that object to query the user’s profile using the respective get() - methods.

- It is also possible to edit the corresponding profile data with the interface **IUserMaint**.
IUser Methods

This interface provides read-access to the user's attributes, and offers basic support for authorization checking.

getFirstName()  
Gets the user's first name.

gGetCellPhone()  
Gets the user's default cellphone number.

isMemberOfGroup(Group, recursive)  
Checks if the principal belongs to the passed uniqueIdOfGroup

isMemberOfRole(Role, recursive)  
Checks if the principal belongs to the passed uniqueIdOfRole

getAttribute(Namespace, AttributeName)  
Returns the specified attribute for the user.

hasPermission(permission)  
Returns true if the user has the given permission.
Obtaining Information about the Current User

The user associated with the current portal request can be obtained by calling the `getUser()` method in the `IPortalComponentRequest` object.

Information about the user can be obtained via the `getXXX()` methods.

```java
IPortalComponentRequest request = ...;

IUser user = request.getUser();
String userName = user.getDisplayName();
String depName = user.getDepartment();
String phone = user.getTelephone();
String email = user.getEmail();
...
```
For Web Dynpro applications, use `getSAPUser()` from the `IWDClientUser` interface:

```java
try {
    IUser user = WDClientUser.getCurrentUser().getSAPUser();
    wdContext.currentContextElement().setUserDisplayName
        (user.getDisplayName());
} catch (WDUMException ex) {
}
...
```
Obtaining Information about another User

- Access the user by calling `getUserByLogonID()` with the logon ID of the desired user.

- An exception occurs if the user does not exist.

```java
String uid = "demouser";
try {
    IUser user = UMFactory.getUserFactory().getUserByLogonID(uid);
    String userName = user.getDisplayName();
    String email = user.getEmail();
    . . .
} catch (UMException ex) {
    String exMsg = null;
    if (ex.getNestedException() != null)
        exMsg = ex.getNestedException().getLocalizedMessage();
    else
        exMsg = ex.getLocalizedMessage();
    response.write("UM Exception occurred: " + exMsg);
}
```
Obtaining Information about another User (2)

- You can also obtain the user if you know the **Unique ID** of the user by using `getUser()` from the **UserFactory**.
- An exception occurs if the user does not exist.

Unique IDs can be obtained from the search interfaces. Do **not** try to manually construct the Unique ID (as shown below):

```java
String uniqID="USER.CORP_LDAP.uid=i802895,ou=people,dc=sap,dc=corp";
try {
    IUser user = UMFactory.getUserFactory().getUser(uniqID);
    String userName = user.getDisplayName();
    String email = user.getEmail();
    ...
} catch (UMException ex) {
    String exMsg = null;
    if (ex.getNestedException()!= null)
        exMsg = ex.getNestedException().getLocalizedMessage();
    else
        exMsg = ex.getLocalizedMessage();
    response.write("UM Exception occurred: " + exMsg);
}
```
 LDAP Attributes accessible via UME

- The standard UME configuration ships with default attributes for LDAP data sources
- Logical Attribute names are mapped to Physical Attribute names in the dataSourceConfiguration_XXX.xml config file

<table>
<thead>
<tr>
<th>Logical Attribute</th>
<th>Physical Attribute</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstname</td>
<td>givenname</td>
<td>getFirstName()</td>
</tr>
<tr>
<td>displayname</td>
<td>displayname</td>
<td>getDisplayName()</td>
</tr>
<tr>
<td>lastname</td>
<td>sn</td>
<td>getLastName()</td>
</tr>
<tr>
<td>fax</td>
<td>facsimiletelephonenumber</td>
<td>getFax()</td>
</tr>
<tr>
<td>uniquename</td>
<td>uid</td>
<td>getUniqueName()</td>
</tr>
<tr>
<td>email</td>
<td>mail</td>
<td>getEmail()</td>
</tr>
<tr>
<td>mobile</td>
<td>mobile</td>
<td>getCellPhone()</td>
</tr>
<tr>
<td>telephone</td>
<td>telephonenumber</td>
<td>getTelephone()</td>
</tr>
<tr>
<td>department</td>
<td>ou</td>
<td>getDepartment()</td>
</tr>
<tr>
<td>description</td>
<td>description</td>
<td>getDescription()</td>
</tr>
<tr>
<td>streetaddress</td>
<td>postaladdress</td>
<td>getStreet()</td>
</tr>
</tbody>
</table>
Namespaces

Different applications or iViews which want to store application-specific data for the same principal object might cause inconsistencies by overwriting or deleting the data written by another application or iView. To prevent inconsistencies of this type, UME configuration files support namespaces, where each application can store its application-specific data in a separate namespace. In this way, custom attributes that have the same name are stored in a different namespace and do not clash with each other.

Example:

A user, USER.CORP_LDAP.uid=testuser,ou=research,o=sap, has three email addresses, one in it’s master data (in the UME default namespace com.sap.security.core.usermanagement), one for application1 and another for application2.
Obtaining Nonstandard LDAP Attributes

- Additional **Custom Attributes** can be added by editing the `dataSourceConfiguration_xxx` file.
- Details on configuration of the data sources can be found in the SAP NetWeaver '04 Help:
  Identity Management -> User Management Engine -> UME Configuration -> Customizing a UME Data Source Configuration
- A mapping (logical to physical) needs to be performed for each attribute
- Use `getAttribute()` to retrieve the value of the required custom LDAP attribute
Attribute Mapping Configuration

Details at help.sap.com:
Identity Management -> User Management Engine
-> UME Configuration -> Customizing a UME Data Source Configuration
Adding a new LDAP Attribute in Custom Namespace

```xml
<dataSource id="CORP_LDAP" />

<responsibleFor>
  <principal type="user">
    <nameSpaces>
      <nameSpace name="com.customer.usermanagement">
        <attributes>
          <attribute name="hr_org"/>
        </attributes>
      </nameSpace>
    </nameSpaces>
  </principal>
</responsibleFor>

<attributeMapping>
  <principals>
    <principal type="user">
      <nameSpaces>
        <nameSpace name="com.customer.usermanagement">
          <attributes>
            <attribute name="hr_org" physicalAttributeName="hrorganization"/>
          </attributes>
        </nameSpace>
      </nameSpaces>
    </principal>
  </principals>
</attributeMapping>
```

Add new "hr_org" logical attribute

Add physical attribute mapping to LDAP attribute "hrorganization"
Reading Custom LDAP Attributes

Access the configured custom LDAP attributes by calling `getAttribute()` with the namespace and logical attribute name.

`getAttribute()` returns a String array of attribute values for the specified logical attribute – even if the attribute is not defined to be multi-valued.

```java
static final String CUSTOMER_NAMESPACE = "com.customer.usermanagement";

IUser user = ...;

String org[] = user.getAttribute(CUSTOMER_NAMESPACE,"hr_org");

if (org != null) {
    for (int i=0; i< org.length(); i++) {
        response.write("Org: " + i + " = " + org[i]);
    }
} else {
    response.write("No orgs found for this user.");
}
```
Searching for Users

Searching for UME objects consists of:

1. Creating a search filter from the UserFactory
2. Setting the Search Attributes for the search
3. Invoking the search
4. Iterating through the search result

The result of the search is of type ISearchResult and returns an iterator containing the UniqueIds of the principals returned.

The search result also contains the state of the search:

SEARCH_RESULT_INCOMPLETE
SEARCH_RESULT_OK
SEARCH_RESULT_UNDEFINED
SEARCH_RESULT_EXCEEDED
TIME_LIMIT_EXCEEDED
Searching for Users – Code sample

This example illustrates how to search for users with last name "Sm?th*

```java
IUserFactory userFact = UMFactory.getUserFactory();

IUserSearchFilter userFilt = userFact.getUserSearchFilter();

userFilt.setLastName("Sm?th*", ISearchAttribute.LIKE_OPERATOR, false);

ISearchResult result = userFact.searchUsers(userFilt);
if (result.getState() == ISearchResult.SEARCH_RESULT_OK) {
    while (result.hasNext()) {
        String uniqId = (String) result.next();
        IUser thisUser = userFact.getUser(uniqId);
        String name = thisUser.getDisplayName();
        ...
    }
} else {
    // print error or warning
}
```

Returns UniqueID of user

Use the UniqueID to obtain an instance of the User
This example illustrates how to search for a custom attribute "hr_org"

```java
String organization = "IT_DEVELOPMENT";

IUserFactory userFact = UMFactory.getUserFactory();
IUserSearchFilter grpFilt = userFact.getUserSearchFilter();
userFilt.setSearchAttribute("com.customer.usermanagement", 
    "hr_org", organization, 
    ISearchAttribute.EQUALS_OPERATOR,false);

ISearchResult result = userFact.searchUsers(userFilt);
if (result.getState() == ISearchResult.SEARCH_RESULT_OK) {
    while (result.hasNext()) {
        String uniqId = (String) result.next();
        IUser thisUser = userFact.getUser(uniqId);
        String name = thisUser.getDisplayName();
        ...
    }
} else {
    // print error or warning
}
```
Changing Information about the Current User

- User information can be modified using the `IUserMaint` interface and the associated set() methods.
- Use `commit()` to update the user after making all changes.

```java
IUserFactory userFact = UMFactory.getUserFactory();
String userId = "testuser01";

IUserMaint user = userFact.getMutableUser(
    userFact.getUserByLogonID(userId).getUniqueID());

user.setFirstName("Demo");
user.setLastName("User 01");
user.setCity("Dallas");
user.setState("TX");
user.setEmail("testuser01@sap.com");
// save the user data
user.commit();
```
IUserAccount

The IUserAccount interface is used to get and set user account data (passwords, group membership or users role).

checkPassword(String password)
  compares the stored password with the input password

getSuccessfulLogonCounts()
  get number of successful logon attempts

isPasswordChangeRequired()
  indicates the need to force change password on next logon

newUserAccount(String userID, String uniqueID)
  Creates a new user account with userID and associates it with the user identified by the unique ID.
Creating a New User

- Users can be created with the UserFactory `newUser()` method. `IUserMaint` represents a modifiable user object. You must issue a `commit()` in order to actually create the user.

- An associated account needs to be created for the User with `newUserAccount()` using the UniqueID of the new User.

- Take care to delete the user if an error occurs (catch the appropriate exception and handle it).
// Create User
String uid = "testuser01";
IUserMaint newUser = UMFactory.getUserFactory().newUser(uid);
newUser.setFirstName("Test");
newUser.setLastName("User");
newUser.setEmail("testuser01@sap.com");
newUser.save();
newUser.commit();
try {
   // Create User Account
   IUserAccount uacc = UMFactory.getUserAccountFactory().
       newUserAccount(uid, newUser.getUniqueID());
   uacc.setPassword("secret");
   uacc.save();
   uacc.commit();
   // NOTE: productive passwords cannot be set if the
   // account is created in a SAP System
   // continue process by trying to set password flag...
Creating a New User (3)

// Now try to modify password change flag for the new user
try {
    IUserAccount uacc = UMFactory.getUserAccountFactory().
        getMutableUserAccount(uacc.getUniqueID());
    uacc.setPasswordChangeRequired(false);
    uacc.save();
    uacc.commit();
} catch (UMException ex) {
    //trace warning: “Password is not productive and has to
    //               changed during the first logon”.
}

} catch (UMException umex) {
    UMFactory.getUserFactory().deleteUser(newUser.getUniqueID());
    //trace error: “User cannot be created because of an
    //            error which occurred during the creation of the
    //            corresponding account object”
}
Deleting an Existing User

- Users can be deleted with the UserFactory `deleteUser()` method.
- All associated accounts, role assignments, etc. are deleted.

```java
// Delete a User
String uid="testuser01";

try {
    IUserFactory userFact = UMFactory.getUserFactory();
    IUser user = userFact.getUserByLogonID(uid);
    String uniqueId = user.getUniqueID();
    userFact.deleteUser(uniqueId);
    response.write("Delete of user: " + uid + " appears to have succeeded.");
} catch (UMException ex) {
    ...
}
```
IGroupFactory - Overview

The Group factory IGroupFactory provides means to:

- Instantiate group objects
- Create new groups (possibly by copying the data of an existing one)
- Delete groups
- Search for groups based on different criteria
- Perform mass commit/rollback operations on a set of users
- Establish listeners for Group operations

Access to the Group Factory can be obtained by calling:

```java
import com.sap.security.api.*;

IGroupFactory grpFact = UMFactory.getGroupFactory();
```
Some of the Group maintenance functions:

**newGroup** (String uniqueName)
Creates a new group with the Unique Name (name MUST be unique)

**deleteGroup** (String uniqueID)
Delete a group from the data store

**getGroup** (String uniqueID)
Gets the group object with the given unique ID

**addUserToGroup** (String uniqueIdOfUser, String uniqueIdOfGroup)
assign user with uniqueIdOfUser to group with uniqueIdOfGroup.

**getMutableGroup** (String uniqueName)
Gets an updateable group object with the given uniqueName
The **IGroup** object represents logical groups of users or other principals.

Groups are may be derived from LDAP OUs in a deep hierarchy, or by individual user assignments in a flat hierarchy.

Groups often have many user members (of type IUserInfo).

Groups may also have roles assigned to them (of type IRole).

Groups may contain other groups (nested groups).
Sample IGroup Methods

get_Description ()
  Returns the description of this group.

is_GroupMember (String uniqueIdOfGroup, boolean checkParents)
  Returns true if the passed principal of type groups is a member of this group.

g_GroupMembers (boolean getChildMembers)
  Returns principals of type group belonging to this group.

g_UserMembers (boolean getChildMembers)
  Returns principals of type user belonging to this group.

g_Roles (boolean getChildMembers)
  Returns all roles belonging to this group.

add_UserMember (String uniqueIdOfUser)
  Adds the user identified by Unique ID to the group.

remove_UserMember (String uniqueIdOfUser)
  Removes the user identified by Unique ID from the group.
Obtaining a Group

Use one of the following methods from the Group Factory to obtain an instance of a Group:

- `getGroup(String uniqueID)`
- `getGroupByUniqueName(String uniqueName)`
- `getGroups(String[] arrayOfUniqueIDs)`
- `getMutableGroup(String uniqueID)`

```java
// Get a Group
String groupid1 = "<unique-id of group1>";
IGroup grp = UMFactory.getGroupFactory().getGroup(grpid1);

String grpDesc = grp.getDescription();
IUser user = request.getUser();
if (grp.isUserMember(user.getUniqueID())) {
    response.write(user.getFirstName() +
                   " is a member of " + grpDesc);
}
```
Searching for Groups

1. Searching for a group or role consists of:
   1. Creating a search filter from the Group or Role Factory
   2. Setting the Search Attributes for the search
   3. Invoking the search
   4. Iterating through the search result

2. The result of the search is of type ISearchResult and returns an iterator containing the Uniquelds of the principals returned

3. The search result also contains the state of the search:

   SEARCH_RESULT_INCOMPLETE
   SEARCH_RESULT_OK
   SEARCH_RESULT_UNDEFINED
   SEARCH_RESULT_EXCEEDED
   TIME_LIMIT_EXCEEDED
Searching for Groups – Code sample

This example illustrates how to search for groups with description "Finance"

```java
IGroupFactory grpFact = UMFactory.getGroupFactory();

IGroupSearchFilter grpFilt = grpFact.getGroupSearchFilter();

grpFilt.setDescription("Finance",
    ISearchAttribute.EQUALS_OPERATOR,false);

ISearchResult result = grpFact.searchGroups(grpFilt);
if (result.getState() == ISearchResult.SEARCH_RESULT_OK) {
    while (result.hasNext()) {
        String uniqId = (String) result.next();
        IGroup thisGroup = grpFact.getGroup(uniqId);
        String name = thisGroup.getDisplayName();

        // Do something with the group...
    }
}
else {
    // print error or warning
}
```

Returns UniqueID of group
Use the UniqueID to obtain an instance of the Group
Group Assignment of Single Users

- Use the Group Factory's `addUserToGroup` or `removeUserFromGroup` methods to assign or remove a group to a single user
- An automatic `commit()` is performed after calling the method
- Do not use this method of assignment for assigning or removing large numbers of users

```java
// assign individual user to a Group
String userid = "<unique-id of user>";
String groupid = "<unique-id of group>";
IGroupFactory grpFact = UMFactory.getGroupFactory();

grpFact.addUserToGroup(userid, groupid);

grpFact.removeUserFromGroup(userid, groupid);
```
Group Assignment of Multiple Users

- Every commit on a UME object leads to a cluster-wide invalidation of the object in the UME caches. Refer to SAP Note 746682 for details.

- Too many unnecessary calls of `commit()` are a performance problem because of cluster notifications and additional read operations afterwards.

```java
// assign users to Groups
String groupid1 = "<unique-id of group1>";
String groupid2 = "<unique-id of group2>";
IGroup grp1 = null;
grp1 = UMFactory.getGroupFactory().getMutableGroup(grpid1);
grp1.addUserMember("<uniq-id1>");
grp1.addUserMember("<uniq-id2>");
grp1.addUserMember("<uniq-id3>");
grp1.save();
grp1.commit(); // update this group before going to next group
IGroup grp2 = UMFactory.getGroupFactory().getMutableGroup(grpid2);
grp2.addUserMember("<uniq-id4>");
grp2.addUserMember("<uniq-id5>");
grp2.save();
grp2.commit();
```
IRoleFactory - Overview

The Role factory IRoleFactory provides means to:

- Instantiate role objects (from a User Management perspective)
- Create new roles (from a User Management perspective)
- Delete roles
- Search for roles based on different criteria
- Add and delete users or groups to a role
- Establish listeners for Role Operations

Access to the Role Factory can be obtained by calling:

```java
import com.sap.security.api.*;

IRoleFactory roleFact = UMFactory.getRoleFactory();
```
IRoleFactory – Maintenance Methods

This interface provides methods to access, create, search and delete IRole objects. Here are some of the methods:

getRole(String uniqueID)

Gets the role object with the given uniqueID

getRoleByUniqueName (String uniqueName)

Gets the role object with the given uniqueName

addGroupToRole (String uniqueIdOfGroup, String uniqueIdOfRole)

Adds the specified group to the specified role

addUserToRole (String uniqueIdOfUser, String uniqueIdOfRole)

Adds the specified user to the specified role

getMutableRole (String uniqueID)

Gets an updateable role object with the given uniqueID

registerListener (RoleListener listener, int modifier)

Registers a listener to be called when certain actions on a Role occur
IRole Object

- The IRole object represents logical roles from a user management perspective.
- The IRole object does NOT contain the content associated with a PCD role.
- Roles are often associated with Group objects (of type IGroup).
- Roles may have many User members (of type IUser).
IRole – Sample Methods

This interface provides read and write access to attributes and properties Roles

getUserMembers (boolean getChildMembers)
Returns IDs of user principals belonging to this role

getGroupMembers (boolean getChildMembers)
Returns IDs of group principals belonging to this role

isUserMember (String member, boolean checkChildren)
Returns true if the passed principal of type user is assigned to this role.

isGroupMember (String member, boolean checkChildren)
Returns true if the passed principal is a member of this role.
Role Assignment of Single Users

- Use the Role Factory’s `addUserToRole` or `removeUserFromRole` methods to assign or remove a role to a single user.
- An automatic `save()` and `commit()` are performed after calling the method.
- Do not use this method of assignment for assigning or removing large numbers of users.

```java
// assign individual user to a Role
String userid = "<unique-id of user>";
String roleid = "<unique-id of role>";
IRoleFactory roleFact = UMFactory.getRoleFactory();

roleFact.addUserToRole(userid, roleid);

roleFact.removeUserFromRole(userid, roleid);
```
Role Assignment of Multiple Users

- Every commit on a UME object leads to a cluster-wide invalidation of the object in the UME caches.
- Too many unnecessary calls of `commit()` are a performance problem because of cluster notifications and additional read operations afterwards.
- Get an updateable role, and modify the role instead:

```java
// assign users to roles
String roleid1 = "<unique-id of role1>";
IRole role1 = null;
role1 = UMFactory.getRoleFactory().getMutableRole(roleid1);
role1.addUserMember("<uniq-id1>");
role1.addUserMember("<uniq-id2>");
role1.addUserMember("<uniq-id3>");
role1.save();
role1.commit();  // update this role before moving on to next
```
More Information...

SDN Articles:
- Using The New User Management API in EP 6.0
- Configure UME for Multiple LDAP Data Sources
- Developer Tutorial UME 40

Javadocs (PDK or NWDS Online Help):

```java
com.sap.security.api

Interface IUser

All Superinterfaces:
java.security.Principal, java.io.Serializable

All Known Subinterfaces:
IUserMaint

public interface IUser
extends java.security.Principal, IUser

This interface provides read-access to the user's attributes, and offers basic support for and checking. Implementations of this interface must make sure that all get-methods with a retur
You should now be able to use the User Management API to:

- Obtain information about users
- Search for users
- Create new users
- Delete users
- Search for groups and/or roles
- Add members to groups and/or roles
Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Die von SAP AG oder deren Vertriebsfirmen angebotenen Softwareprodukte können Softwarekomponenten auch anderer Softwarehersteller enthalten.

Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® und SQL Server® sind eingetragene Marken der Microsoft Corporation.

IBM®, DB2®, DB2 Universal Database, OS/2®, Parallel Sysplex®, MVS/ESA, AIX®, S/390®, AS/400®, OS/390®, OS/400®, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere®, Netfinity®, Tivoli®, Informix und Informix® Dynamic ServerTM sind Marken der IBM Corporation in den USA und/oder anderen Ländern.

ORACLE® ist eine eingetragene Marke der ORACLE Corporation.

UNIX®, X/Open®, OSF/1® und Motif® sind eingetragene Marken der Open Group.

Citrix®, das Citrix-Logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® und andere hier erwähnte Namen von Citrix-Produkten sind Marken von Citrix Systems, Inc.


JAVA® ist eine eingetragene Marke der Sun Microsystems, Inc.

JAVASCRIPT® ist eine eingetragene Marke der Sun Microsystems, Inc., verwendet unter der Lizenz der von Netscape entwickelten und implementierten Technologie.

MarketSet und Enterprise Buyer sind gemeinsame Marken von SAP AG und Commerce One.