Users and Groups 2

Institutional User Data

The UniversalUser object is a wrapper for institutional data. It contains fields that are essential for the application to function and are not institution specific, along with convenience methods that are used in authorization. It also provides access to module-specific user information via its list of `KualiModuleUsers`. The nervous system takes care of setting up this object upon a new session and making available for other code through the GlobalVariables object.

The `UniversalUserService` provides the methods that access the institutional user data. Much work has been done to ensure that you can switch out the provided implementation, which assumes that the institutional user data is in FS_UNIVERSAL_USR_T, with LDAP or other. However, this could lead to performance concerns, particularly in cases like the account and project director lookups which employ database level joins in the default implementation (see `LookupDaoOjb`). If you do choose to override the default implementation, you can do this via a simple spring bean override. The bean id is *universalUserService*. If you do not want to maintain the institutional user data in Kuali but also don't want to impact performance, you can simply feed FS_UNIVERSAL_USR_T from your source for institutional user data. The `maintain.users.locally` configuration property controls whether this section of the user document is editable. And, the `validate.password` property is used to control the editability of that field on the User document, in addition to being used by the default authentication implementation.

To avoid assuming that the institutional user data resides in the KFS database, we do not map `UniversalUser` to other objects via OJB reference-descriptors. Instead, we link other entities to institutional user information via data dictionary relationships.

Account.xml

```
<relationships>
      <relationship objectAttributeName="accountFiscalOfficerUser" >
            <primitiveAttribute sourceName="accountFiscalOfficerSystemIdentifier"
targetName="personUniversalIdentifier" />
      </relationship>

      <relationship objectAttributeName="accountSupervisoryUser" >
            <primitiveAttribute sourceName="accountsSupervisorySystemsIdentifier"
targetName="personUniversalIdentifier" />
      </relationship>

      <relationship objectAttributeName="accountManagerUser" >
            <primitiveAttribute sourceName="accountManagerSystemIdentifier"
targetName="personUniversalIdentifier" />
      </relationship>

      <relationship objectAttributeName="accountGuideline" >
           <primitiveAttribute sourceName="chartOfAccountsCode"
targetName="chartOfAccountsCode" />
           <primitiveAttribute sourceName="accountNumber" targetName="accountNumber"
/>
      </relationship>
</relationships>
```

As the frameworks encounter an UniversalUser object, through introspection, the object name in the business object is then found. From the object name, the framework can query the data dictionary for the associated primitive.

Lazy getters on UniversalUser objects
To retrieve UniversalUser objects for various business objects, lazy getters have been implemented to retrieve the object through the user service using the primitive:

Lazy getter on UniversalUser Object - Account.java

```
public UniversalUser getAccountFiscalOfficerUser() {
   accountFiscalOfficerUser =
SpringContext.getBean(UniversalUserService.class).updateUniversalUserIfNecessary
(accountFiscalOfficerSystemIdentifier, accountFiscalOfficerUser);
   return accountFiscalOfficerUser;
}
```

Getting the Current User
The current user can be retrieved within any piece of code by the program statement.

```
UniversalUser currentUser = GlobalVariables.getUserSession().getUniversalUser();
```

`WorkflowUser` is the object that represents institutional user data to KEW. By default the nervous system maps `WorkflowUser` to FS_UNIVERSAL_USR_T also. This same configuration may be maintained by deploying the KFS institutional plugin into KEW.

Many different user identifiers exist. The `UserId` interface has implementations for each unique identifier by which a `UniversalUser` may be retrieved. The `UniversalUserService` has methods to retrieve by the most common identifiers, as well as a method to retrieve by any `UserId` implementation:

```
    public UniversalUser getUniversalUserByAuthenticationUserId( String
authenticationUserId ) throws UserNotFoundException;

    public UniversalUser getUniversalUser(String personUniversalIdentifier) throws
UserNotFoundException;

    public UniversalUser getUniversalUser(UserId userId) throws UserNotFoundException;
```

getUniversalUser returns both **inactive** and **active** UniversalUsers. To determine if the user is active for any module in the application, use the program statement *user.isActiveForAnyModule().* If you want to check the active status within a particular module, you can use *user.isActiveForModule(moduleId)*. You can also obtain the module's KualiModuleUser and use its isActive() method.

Module User Data

Also see the module documentation.

Some individual modules need more information about a user than is provided at the institutional level. For example, the chart and financial processing modules need the user to be associated with an organization and do not allow institutional users who have not been given explicit permission to use KFS access to initiate documents. So, the `ChartUser`} defines "active KFS user", "chart code", and "organization code" properties. Via the module registration process, the institutional user document is aware of these additional fields associated with the chart module and draws them on that section of the page. The `ChartUser` has logic to default the chart and organization code from the department id on `UniversalUser` if those are not explicitly specified in that section of the user document. But, if the "active kfs user" checkbox is not checked then the user in question cannot initiate chart documents such as the account and object code documents. The `FinancialUser` also references the "active kfs user" property of `ChartUser` when authorizing access to its functions and allows specification of an organization which defaults to that of the `ChartUser`. These additional properties are defined in the `KualiModuleUserService` bean definition for a given module and

further described in the business object data dictionary entry for the `KualiModuleUser` implementation class, e.g. `ChartUser.xml`. `UniversalUser` has a `Map` that contains module codes and the associated module user objects. It obtains this information from the `KualiModuleService`. The module-specific user data is stored by the user document in key-value pairs in SH_USR_PROP_T.

Getting the Current ChartUser
The current user can be retrieved within any piece of code by using the following statement:
ChartUser currentUser = (ChartUser)GlobalVariables.getUserSession().getUniversalUser().getModuleUser(ChartUser.MODULE_ID);

Kuali Group

KFS and the Rice Nervous System user KEW workgroups for authorization. So, `KualiGroup` simply wraps a workflow workgroup. The majority of the code does not reference `KualiGroup` directly. Rather it asks about a specific users membership by workgroup name via the `UniversalUser.isMember` method. The `KualiGroup.hasMember` method takes a user and can be used to ask the same question. To override the group service in both workflow and KFS, you can either override the `KualiGroupService` and deploy the KFS institutional plugin into KEW or override the `WorkgroupService` within workflow itself.

Unable to render {include} The included page could not be found.