Business Object Data Dictionary 2

Role of the data dictionary

The data dictionary provides metadata about business objects. BOs are java objects that represent data in a system, often corresponding to a row in a database. In DD terminology, an attribute generally maps to getter/setter methods in the java object, which maps to a field in the database.

The data dictionary contains information about:

- Descriptive labels for each attribute
- Metadata about each attribute
- How input fields on HTML pages should be rendered for an attribute (e.g. textbox, drop down, etc.)

The data dictionary does not contain information about:

- Which BO does a table correspond to (responsibility of persistence layer, e.g. OJB)
- How fields in the BO correspond to database columns (responsibility of persistence layer, e.g. OJB)

Creating a new Business Object Data Dictionary file

DD file name

The name of the DD file for a particular BO **must** be the simple class name (i.e. no package name), with the .xml extension. For example, the class `org.kuali.module.chart.bo.Account` would have a DD file named Account.xml.

DD file layout

The account DD file has the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dictionaryEntry PUBLIC
    "-//Kuali Project//DTD Data Dictionary 1.0//EN"
    "../../../core/datadictionary/dataDictionary-1_0.dtd">

<dictionaryEntry>
    <businessObject businessObjectClass="org.kuali.module.chart.bo.Account">
        <!-- inquiry definition -->
        <!-- lookup definition -->
        <!-- attribute definition -->
        <!-- relationship definition -->
    </businessObject>
</dictionaryEntry>
```

Typical DD files consist of 4 primary sections (as noted by the commented XML above): inquiry, lookup, attribute, and relationship. Each of these is represented by a comment in the above XML segment and will be discussed in this tutorial.

There are other components in the complete Account DD file, which are described within the documentation in the DD DTD file.

Inquiry Definition

The inquiry definition indicates which fields are displayed in what order on inquiry screens. Inquiry screens are used to allow users to drill down and find detailed information about a row in the database. Check out the inquiry documentation for a full treatment of how to specify and customize Inquiries using the data dictionary.

Lookup definition

The lookup definition indicates which fields are used to perform a lookup, as well as which fields are listed in the search results. Lookup screens are those that are used to search for a particular row in a table. Check out the lookup documentation for a full treatment of how to declare and

customize a lookup for a business object.

Attribute definition

Attributes definition are used to provide metadata about the attributes (i.e. fields) of a business object. Since there are many attributes in the Account object, only a few attribute definitions are given.

```
<attributes>
    <attributeReference name="chartOfAccountsCode"
sourceClassName="org.kuali.module.chart.bo.Chart"
sourceAttributeName="chartOfAccountsCode">
    </attributeReference>

    <attributeReference name="chartOfAccounts.codeAndDescription"
sourceClassName="org.kuali.module.chart.bo.Chart"
sourceAttributeName="codeAndDescription">
        <label>Chart Code</label>
        <shortLabel>Chart Code</shortLabel>
        <required>false</required>
        <control>
            <hidden />
        </control>
    </attributeReference>

    <attribute name="accountNumber" forceUppercase="true">
        <label>Account Number</label>
        <shortLabel>Account</shortLabel>

        <maxLength>7</maxLength>
        <validationPattern>
            <alphaNumeric exactLength="7" allowWhitespace="false"
allowUnderscore="false" />
        </validationPattern>
        <required>true</required>

        <control>
            <text size="10" />
        </control>
    </attribute>
</attributes>
```

An <attribute> tag is used to describe metadata for an attribute.

As is common throughout KFS, a field representing the same type of data occurs in many different tables. For example, a chart code may occur in many different tables. In these circumstances, the use of an <attributeReference> definition allows the reuse of parts of a standard definition from the "master" business object. For example, the chart object would be the "authorative" source of metadata about the chart of accounts code. An <attributeReference> is used to copy metadata about an attribute from "master" BO's attribute.

Note that an attributeReference does not indicate that there's a foreign key relationship between fields in the BO and the source BO. It merely indicates that the metadata between the two fields are shared (or possibly that there is semantic similarity between two attributes). A foreign key relationship is defined either in the persistence layer (e.g. OJB), or using DD-defined relationships (described in the next section).

For the accountNumber attribute, we specify:

- that the account number code should always be uppercased
- what labels should be used when displaying this attribute
- that this field is required on a maintenance document
- that it should be an alphanumeric string with 7 characters with no whitespace or underscores
- that a text field with a max length of 10 should be used to render this attribute when inputting

For the "chartOfAccountsCode" attribute, an attribute reference is specified. This reference indicates that all of the metadata for this attribute will come from the `Chart` BO's "chartOfAccountsCode" attribute.

For the "chartOfAccounts.codeAndDescription" attribute, an attribute reference is specified as well to a `Chart` BO attribute. However, instead of copying all of the metadata, this definition overrides several metadata attributes about the BO attribute.

AttributeReferenceDummy
The `AttributeReferenceDummy` DD file has many definitions for generic fields such as yes/no flags, dates, dollar amounts, etc. Using a reference when appropriate simplifies allows for a logical linking of metadata, improves consistency of validation, and simplifies the changing of any metadata for a field, since only the metadata for AttributeReferenceDummy needs to be changed. Check out the `AttributeReferenceDummy` DD file for a detailed list of the attributes.

All attribute names that appear in any tag in the `<lookup>`, `<inquiry>`, and other tags should have a `<attribute>` or `<attributeReference>` defined for them. In particular, nested properties should use an attribute reference so that metadata for the field can be carried over from the nested BO, and selected attribute metadata can be overridden as necessary.

There are other advanced options when defining attributes. Please consult the DTD for more details.

Relationship definition

Foreign key relationships are usually defined within the persistence layer (e.g. OJB) when the relationship can actually be modeled within the DBMS.

Foreign key references often result in a referenced BO being linked to on the main business object. For example, the `Account` BO has a field for the chart of accounts code. Because a chart of accounts code uniquely identifies a Chart BO (i.e. it's the primary key), we can also include a Chart attribute on the Account BO as a reference object.

However, there are situations when FK relationships cannot be mapped within the persistence layer because they do not necessarily exist on the database level. The primary example for this is the universal user "table". Although the base KFS implementation places the universal user table within the same database as KFS, KFS was designed to easily allow another datasource to be used (e.g. LDAP). In this case, there is no database-level FK relationship because the data are potentially external to the main KFS database.

Therefore, we need to define relationships within the DD. The following example demonstrates one such mapping.

```
<relationships>
    <relationship objectAttributeName="accountFiscalOfficerUser" >
        <primitiveAttribute sourceName="accountFiscalOfficerSystemIdentifier"
targetName="personUniversalIdentifier" />
        <supportAttribute sourceName="accountFiscalOfficerUser.personUserIdentifier"
targetName="personUserIdentifier" identifier="true" />
        <supportAttribute sourceName="accountFiscalOfficerUser.personName"
targetName="personName" />
    </relationship>
</relationships>
```

In the above example, the `<relationship>` tag is describing the keys of the accountFiscalOfficerUser reference object of the Account BO. This attribute is of type UniversalUser (see the java class definition). The `<primitiveAttribute>` describes the foreign key relationship. In this case, this is saying that the Account's accountFiscalOfficerSystemIdentifierattribute references the personUniversalIdentifier attribute of the reference object, analogous to how it would be defined within a database foreign key. One `<primitiveAttribute>` definition is needed for each field of the pseudo foreign key relationship.

The `<supportAttribute>` specifies other fields that may be returned from a lookup, but are not foreign key fields. In the above example, when returning a value from a lookup on the accountFiscalOfficerSystemIdentifier attribute of the Account BO (i.e. a nested lookup), not only will the return the UniversalUser's personUniversalIdentifier attribute (i.e. the primary key), the personUserIdentifier and the personName fields will be returned as well.

There are other advanced options when defining relationships. Please consult the DTD for more details.

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