Inquiries 2

Very often, users of the Kuali Financial System are going to want to see details about various business objects in the system - Accounts, Object Codes, Awards, Vendors, to name just a minor sub-set. Naturally, users shouldn't need access to the database to see the full details of a business object, and furthermore, it should be extremely easy for developers to create a detail page. In the Kuali Nervous System, these detail pages are called "Inquiry" pages. They are typically specifying completely using only the data dictionary for the business object, which makes them easily declared and highly customizable. On this page, we'll explore how to create Inquiries and options for customizing them.

- Specifying an Inquiry in the Data Dictionary
- Customizing Inquiries
  - Changing drill down links
    - What does KfsInquirableImpl actually do?
  - Decorating the business object
  - Customizing the UI

Specifying an Inquiry in the Data Dictionary

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. For example, lookup results often contain underlined values (i.e. a link), and clicking them will display a screen with more information about that field. It has the following format.

Inquiry Definition

```
<inquiry>
    <title>Account Inquiry</title>
    <inquirySections>
        <inquirySection title="Account Details">
            <inquiryFields>
  <field attributeName="chartOfAccounts.codeAndDescription" />
  <field attributeName="accountNumber" />
  <field attributeName="accountName" />
                <field attributeName="accountFiscalOfficerUserPersonUserIdentifier" />
                <field attributeName="accountFiscalOfficerUser.personName" />
            </inquiryFields>
        </inquirySection>
    </inquirySections>
</inquiry>
```

The title is used as the page title on the inquiry screen. Each of the `<field>` definitions describes the BO's properties that will be shown on the inquiry screen.

Note that a field can point to a property of a reference object (or property of a reference of a reference, etc.). In the above example, "chartOfAccounts.codeAndDescription" refers to the code and description property of the chartOfAccounts reference object.

There are other advanced options when defining inquiries. Please consult dataDictionary-1.0.dtd for more details.

Customizing Inquiries

Just as with lookups, we sometimes want the inquiry screen to work a bit differently than most vanilla inquiries. Any business object that is being seen on an Inquiry screen is wrapped in an implementation of `org.kuali.core.inquiry.Inquirable`. There are two default implementations that can be extended: `org.kuali.core.inquiry.KualiInquirableImpl` and its KFS extension, `org.kuali.kfs.inquiry.KfsInquirableImpl`. Most of the time, we'll discuss `KualiInquirableImpl`; `KfsInquirableImpl` adds interesting behavior to the functionality, but one with a smaller set of use cases.

Changing drill down links

A major use case of extending `KuaiInquirableImpl` is to change the behavior of drill down links. When we perform an inquiry on a business object, the Inquiry screen shows drill down links for all related objects. So, for instance, when we look up an Account in the Account Lookup, and click the account number in the lookup results, it shows the Inquiry page for the Account. Then we see that several of the fields have links for related business objects - for instance, there's a link on the chart line. Click that, and the Inquiry page is replaced with the Inquiry page for that specific Chart object. That's got several related object codes; click on those and we see Object Code inquiries. The Inquiry framework figures all of this out simply by looking at how the O/RM layer and the data dictionary defined relationships with other business objects.

For General Ledger and Labor Ledger inquiries, those drill downs need to be changed for certain fields, and therefore, the Abstract Inquirables defined in each module overrides a special method: getInquiryUrl(). Its parameters are the business object that is the subject of the Inquiry, the name of the attribute that should have a drill down, and a boolean parameter named "forceInquiry," which...er...`KualiInquirableImpl` doesn't seem to use.

Again, when overriding, one can write code to simply wait for specific attributes and otherwise defer to the inherited version of the method. There is also a helpful utility class, org.kuali.core.util.UrlFactory that can be called like so:

```
UrlFactory.parameterizeUrl(RiceConstants.INQUIRY_ACTION, parameters);
```

The parameters are a Map of key/value pairs to put into the URL of the Inquiry, typically the method to call, the class of the business object, and any primary keys to find the business object. This, of course, allows for some link substitution, which may well be useful in certain contexts. For instance, based on the class of the object under inquiry or whether a certain attribute is a nested attribute, org.kuali.module.gl.web.inquirable.AbstractGLInquirableImpl's implementation of getInquiryUrl() adds or changes parameters.

One can also send back an empty String to prevent a drill down link for appearing at all for the attribute. The ability to do such drill down suppressions leads us to ask...hey...

What does KfsInquirableImpl actually do?

Not much. But one actually important something, and, of course, it has to do with drill down links.

For some KFS business objects, especially those related to General Ledger and Labor Ledger classes, empty fields are represented by dashes. So, for instance, if a general ledger entry doesn't have a related project code, it doesn't keep a null in that field, but rather a field full of a certain number of dashes.

Naturally, if we viewed such an Entry on an Inquiry screen, we wouldn't want to be able to drill down on the dashes, because that would just be silly. Therefore, `KfsInquirableImpl` makes sure that if the business object has one of three fields consisting of dashes - projectCode, subAccountNumber, or financialSubObjectCode - it returns an empty String as the link url, thus preventing the link from being shown on the Inquiry page. And that's why we might prefer to extend `KfsInquirableImpl` instead of `KualiInquirableImpl`.

Decorating the business object

There aren't too many use cases for overriding the default behavior of KualiInquirableImpl.getBusinessObject() (indeed, `KfsInquirableImpl` d oesn't), but let's cover it, in case there ever is a need.

What this method does is take in a Map of primary keys and returns the corresponding business object. KualiInquirableImpl already knows its business object class, and that can be determined through the getBusinessObjectClass() method. KualiInquirableImpl implementation simply uses either the UniversalUserService or the LookupService to find the appropriate business object, but again, this behavior can be changed by extending KfsInquirableImpl. Business objects can use this method, for instance, to look up related information.

Customizing the UI

Most of the time, an inquiry will base its UI entirely on the inquiry XML in the business object XML. However, if there's some business reason to go beyond what the data dictionary XML can provide, Inquirable exposes a method called getSections() which provides a great deal of control over how the framework displays the inquiry page.

getSections() takes in the business object that is the subject of the inquiry and returns a `List` of `org.kuali.core.web.ui.Section` objects to be displayed about that object. Each Section object is, in turn, an amalgamation of `org.kuali.core.web.ui.Field` objects which can be manipulated by the Inquirable. This is a lot of work and can often lead to situations where inquiry pages don't follow KFS standard practice. However, if there is a need for very fine grained control over how the inquiry page looks, this is the ultimate method to get that tweaking power.

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