KRAD - Messages from Central Repository

(Draft in progress below ... not completed)

Purpose

This is a technical architecture refinement to achieve the following functional objective. Note that the term "message" here and in the Rich Messages requirement, equates to all UI text elements.

- Message content is separable from the code base for ease of managing: reviewing, editing, updating, making global changes, translating.

This has multiple potential benefits, including:

- higher quality messages - as a result of making it easier for development teams to manage, review and improve message content without impacting the logic/code base.
- messages enabled for localization/internationalization (of this part of the user interface)
- diagnostics/fix information is easier to search for/locate error messages on the web, based on making it easier for teams to publish their error message ids & corresponding content on the web because they are in a centralized and structured repository

This requirement ideally includes all UI text elements.. The work for each element will be prioritized and the 'delta' to the work, sized, assessed and planned for.   Its conceivable that a particular milestone (or the entire 2.2 R1) could include just the centralized error messages, for example, with the other UI elements to follow (in later milestones or releases).   Below is the list of UI text elements -- the "other" are not in prioritized order. This is a work in progress - tbd is reviewing this list to ensure comprehensiveness and best terminology, and then we'll prioritize the items:

- field labels
- instructional text
- constraint text
- watermark text
- checkbox items
- drop-down items
- radio button items
- error messages (includes inline validation messages)
- hover help (tooltips)
- other help content (from help icon)
- button labels (action field labels)
- tab labels (tablist items)
- tree item labels
- link titles
- navigation group element labels
- field group and section titles
- view header
- page header
- application header and footer content
- banner content

- the data itself (this one is out of scope of this release)

Note that there are related requirements:

- One that covers message structure itself, to enable rich UI in the messages (links, images). In addition to the "rich" attributes supported in the message structure, the structured aspect supports being able to display message syntax/variables in a different order and format, depending on the end users' language grammar (for example, right to left, left to right, and grammatical/syntax differences). See Rich Messages.
- One that covers the online help architecture, which relates to this, is also structured and separable from the code base for the same reasons. See Help Framework.

Detailed Description

The data structure and messaging APIs will be determined, but it is assumed that messages will be in a database, or structured message table, or structured resource file, or some other structured repository/format so that:

1. Application code can pass the appropriate message ID to the Rice 'message service' and be returned the appropriate message content to display through the user interface to the end user.
2. Message content can be reviewed, edited, and updated during development through one central message repository (in addition to being viewed in context when using the software).

As we work on this structure, we will also need to keep in mind a model that will support the following:

- 3. Through this separation of messages from the logic/codebase, it should be possible to swap out an English message version with another language message version (this could be when the application is built, compiling in the runtime) -- this is the localization aspect of internationalization.
- 4. Further out, it should be possible to include the messages in multiple languages but have the logic/codebase point to the appropriate message version dynamically, depending on the user interface's language setting. This assumes more dynamic determination of language, rather than at build time - this last aspect would be considered a multiple language user interface.
- 5. Internationalization support for the data itself (the business data that is stored elsewhere, not in Rice, but that is accessed through Rice and the Kuali applications built with KRAD.

This requirement does not include building the capability that will implement aspects 3, 4, or 5 but does include building the message structure and API(s) that will enable these aspects to be implemented. And other requirements will follow in future releases (separating out other UI elements, through a similar centralized data structure).

Usage Scenarios

Usage scenario One:

A user of a Kuali application encounters an error, but the software can't access the error message repository to pull in and display the richer message content (the error message repository is on the server and the application can't get to it, temporarily). Rather than hang, instead the user receives the default cryptic error message that includes the message ID (the message ID is embedded in the logic/codebase, and is the variable sent to the repository to pull down the matched content). The default cryptic error message includes a link to the public web, with the message ID. Or the user can link to his/her preferred web search engine and search on the message ID. The public search results finds the appropriate content (does not require getting to the error message repository), so the user can see the message content. In addition to the message content, the user can also see more details on how to fix this condition. The user also finds there are social aspects, community additions with comments on how they fixed this problem.

For examples, see

- results of search on an Intuit message ID
- results of search on a Microsoft message ID
- results of search on an Oracle message ID
- results of searches that pulled up community involvement:
    - website of error messages on outdated PC technology
    - anchor within book that mentions the outdated PC technology error message in question

Usage scenario Two:

(coming)

Mocks and Diagrams

Include any mocks (for UI enhancements) or diagrams that might be helpful in understanding the issue:

Performance

If applicable, list expectations for performance (optimal and worst cases would be fine, give time in seconds):

References

Enabling for internationalization includes two major aspects:

- Enabling for localizability, by separating UI elements, such as error messages, from the logic/codebase.
  This is a common, standard software architectural direction and is usually the first step in the internationalization process. It supports creating multiple versions of software, each of which could be a different runtime/build (one for each language). It doesn't necessarily assume that multiple languages are incorporated into a single running version.

- Enabling for globalization first requires localization, but goes beyond this, incorporating multiple languages, with dynamic capabilities to determine which is appropriate at the time, based on other settings/values.   Full globalization assumes that each user would receive their preferred display language, which could vary across users on the same system, application, or website.

Other references:

- http://userguide.icu-project.org/i18n
- http://www.ibm.com/developerworks/java/library/j-jspapp/
- http://www.javaworld.com/javaworld/jw-03-2000/jw-03-ssj-jsp.html
- http://www.ibm.com/developerworks/java/tutorials/j-i18n/

- http://java.sun.com/javase/technologies/core/basic/intl/
- http://msdn.microsoft.com/en-us/goglobal/bb978454
- http://en.wikipedia.org/wiki/Multilingual_User_Interface

Related JIRAs:

- KULRICE-173

Requirements Listing

List all requirements (individual verifiable statements) that indicate whether the work for this item has been complete. If there are requirements that are not absolutely essential to the functionality as a bare minimum but are desired / would be nice to have if time allows, enter those under 'Secondary':

Primary:

1. *item*
2. *item*

Secondary:

1. item
2. item

Dependencies

List any functional or technical work that must be completed before work on this item can begin:

1. item

Issues

List any issues that need to be resolved before work on this item can begin:

Functional:

1. item
2. item

Technical:

1. If we could get the API for the KS Message service and the message table structure that would be helpful for design purposes
2. Instead of moving all the current text out of the dictionary and into the external message repo, we could leave them in there as defaults. Then if any message exists in the repo that would override the default in the dictionary. This would be nice since there would always be a value for a property, and easier to develop. Essentially the message service could just be an override mechanism but you could override everything.

QA or Regression Testing Plan

List steps needed to test the basic functionality of this update, enhancement, bug fix

1. test/steps
2. test/steps

Checkoff

Functional Lead: Candace Soderston, csoders@uw.edu

Functional Analysis Complete? **No**

Needs Review by KAI? **Yes**

Technical Analysis Complete? **No**

Needs Review by KTI? **Yes**

Estimate: **30 hours**

Technical Design: External Messages Repository

Jira: KULRICE-6676

Final Documentation: Link Here