Kepler - Bug #4394

Need to develop requirements for configuration subsystem

09/17/2009 03:53 PM - David Welker

Status:	Resolved	**Start date:**	09/17/2009
Priority:	Immediate	**Due date:**	
Assignee:	Chad Berkley	**% Done:**	0%
Category:	core	**Estimated time:**	0.00 hour
Target version:	2.0.0	**Spent time:**	0.00 hour
Bugzilla-Id:	4394		

Description

I have noticed some development activity with respect to configuration that seems to consist of developing a fairly low level design (i.e. an API) and implementation documents. I think this is problematic.

We need to come to an agreement on configuration requirements (i.e. how will the configuration system work from the users perspective) before we even think about design documents.

History

#1 - 09/17/2009 04:57 PM - Matt Jones

I agree that we need requirements and a good design before coding begins. No coding has begun. The current configuration project has design documents only in SVN, which includes both a preliminary requirements list and some initial sequence diagrams and thoughts about a high-level API. We will be expanding and refining these design documents over the near future, and the development tasks will be turned into bugs before coding begins.

#2 - 09/17/2009 09:39 PM - David Welker

I see that there are some requirements. But, in my opinion, those requirements are: (1) much too low-level. They are not that far from being implementation details. (2) Not agreed upon.

As far as whether designing is necessary before coding, I definitely don't agree that it is necessary. I don't think it is particularly harmful either, as long as the activity does not handcuff the implementer.

The critical activity that we need to engage in is not design (that can be left to the implementer) but rather requirements. And when I say requirements, I mean from a user perspective. What do we want to be configured. What should configuration files look like? What is the relationship between configuration and persistent user preferences? What do files that enable internationalization look like, and how will users change those files.

What I have seen so far, including the file referred to as containing requirements, contains much lower level details. And just as importantly, none of these requirements have been discussed.

Design artifacts such as sequence diagrams (which are of dubious value when the use case is simple) are definitely premature when requirements are not even understood nor agreed to.

#3 - 09/28/2009 11:16 AM - Sean Riddle

I'm curious about the following implementation detail on the wiki:

8) The system will not allow modules to change the value of a property based solely on its name.

What would one need in addition to the name? The module name, perhaps? What is the purpose of this requirement?

#4 - 09/28/2009 11:31 AM - Matt Jones

The point is that property names are scoped within their module and namespace in their module, so methods for changing property values will have to at least reference the module to which the property belongs. The reason for the requirement is to eliminate the use of unscoped properties and thereby reduce potential accidental property conflicts among modules.

#5 - 09/28/2009 11:32 AM - Sean Riddle

Oh, alright. And there's no concept of permissions, right? So any module can change any other module if they know the name of the module and the name of the property?

#6 - 09/28/2009 11:39 AM - Matt Jones

Correct, there is no concept of permissions. We discussed this, and decided it would be essentially impossible for a module author to determine ahead of time how another module might want to interact, and therefore really not feasible to declare properties as private to a module without compromising extensibility.

There is, however, a concept of 'mutable', which allows a module to mark a property as one which is/should be monitored at runtime for changes, and the application will respond appropriately without restart. If a property is not marked as 'mutable', then other modules should assume a Kepler restart is needed for their property change to take effect. If it is mutable, then they can assume property changes take effect at the time of the change, and they should be careful to monitor further property changes that might affect them at runtime using the listener interfaces.

#7 - 09/28/2009 12:50 PM - David Welker

I have looked at the "requirements" and I noticed that they are fairly low-level. In fact, they aren't all that distinguishable from implementation details.

If you look at the label "Configuration Implementation Details" and look at the list items under it, you can see that all of these list items there would all fit comfortably as list items along with the items "Configuration System Requirements."

I feel as though there is a fundamental misunderstanding here about the difference between requirements and implementation details.

Requirements address the question of "what" but also should hint or state explicitly the question "why." Requirements are less flexible, as these are the things we actually want the system to accomplish. Implementation details, in contrast, answer the question of "how." They are more flexible, in that there are many paths to meeting the same requirements.

So, why are we starting with a very low-level design that seems to address the question of how, before we have established the prerequisite questions of "what" and "why"? Isn't this premature? Shouldn't our design should be driven by our requirements, and not the other way around?

I also have looked at the design documents proposed including the class diagram and the associated sequence diagram. I don't have any particular objections to either of these things. (Mainly because we have not come to any agreement on requirements.) However, I do not think they add much value at this stage. First of all, sequence diagrams are not something that are efficient to produce for a plain vanilla design which is fairly straightforward, as this one is. Second, the class diagram isn't particularly useful either, especially as it consists primarily of getters and setters and other trivial methods. It would be fairly straightforward to develop such a simple API at implementation time.

My point here is not to criticize the hard work that went into these things. My point is that we are focusing on the wrong things and in the wrong order. The main questions associated with the configuration system that need to be agreed upon are the questions of "what" and "why." The actual implementation after that point will be somewhat time-consuming, but will also be fairly straightforward.

Out of the documentation thus far produced, the most useful part of it by far is the part labeled "Misc Notes." Here, we are actually getting into some details about how the configuration will actually work and feel in practice. That is what is actually what we should have focused on first. After we develop requirements, whether we will actually need a design which includes class diagrams and especially sequence diagrams is doubtful, given that the actual system will likely be straightforward. I don't feel that these things are optimal in terms of documentation; well-written English, an under-appreciated skill amongst many developers, is usually a better sort of documentation. Especially since the contents of class diagrams are easily perceived by looking at the Javadoc, and sequence diagrams are not usually very enlightening, especially since they are likely to drift away from the actual code. Are we going to be updating sequence diagrams with every change in code?? As far as guiding actual coding efforts, I do not believe that either class diagrams or sequence diagrams are useful, especially for the experienced developers we have here. So, if we aren't really helping the implementation process or producing quality documentation, what exactly are we accomplishing with class diagrams and sequence diagrams in this context? We do not have unlimited resources, so I think we should think carefully about this question of what design documents we really want to use (and agree on!) before proceeding with implementation. Do we really need class diagrams that consists mostly of trivial items like getters and setters and other trivial and obvious methods? What insights are revealed by such a class diagram? Further, what insights does the sequence diagrams reveal that justifies the time that went into producing it? Is the sequence diagram supposed to be some form of documentation for future developers, and if so are we going to invest the resources to keep the sequence diagram synchronized with the code? If the sequence diagram is a form of documentation, are the non-obvious insights that exist within it (and for the record, I don't know what those insights are) better expressed in English?

Anyway, I would like to add the following items to the agenda for Friday. (1) A discussion of the real requirements for the configuration system. (2) A discussion of the design process we are going to use, which in my view should not consist of the mechanical production of design artifacts but instead should be critically focused so that we produce the design documents that are actually most useful and effective given resource constraints.

That said, I would like to thank the people who have put in hard work in getting the ball rolling on this.

#8 - 10/05/2009 11:53 AM - Chad Berkley

After having a conference call about the requirements for the config system last friday, I think we are in agreement on requirements. We now need to have a discussion about design details. The first discussion should be on alternate representation formats for serialized config files.

Below are the agreed upon requirements. They are also listed at
https://kepler-project.org/developers/teams/framework/kepler-configuration/proposed-future-kepler-configuration-system

1) A module is able to add configuration properties to the configuration

2) A module is able to read its own configuration properties

3) A module is able to add or override configuration properties added by other modules.

4) A module is able to overwrite another module's configuration properties

5) A module can indicate whether a property is mutable (i.e., if property changes are noticed and incorporated without restarting the application). At runtime, only mutable properties can be overwritten. It is the responsibility of all modules to utilize mutable properties to monitor those properties and respond appropriately to changes.

6) A configuration property can have either or both of a single scalar value or a list of subordinate configuration properties. Subordinate configuration

properties must have unique names within their parent property.

7) The configuration system is able to notify modules or other registered listeners that a configuration change has taken place

8) The configuration system is able to serialize each module's configuration properties as text

9) The configuration serialization should be able to store different language versions of each property file (internationalizable)

10) Each module can have its own configuration and can organize its configuration into any number of namespaces, allowing the module to organize its properties into groups. One use case for this is to allow a module to separate its UI strings from other configuration properties.

11) The configuration system is able to tell which module owns a configuration property

12) The configuration system is able to store default configuration properties separately from user-modified configuration properties. Consider whether user preference files, including loading and sharing of such files, is in scope or not.

13) The configuration system should be loaded under all variants of the Kepler application and should have minimal dependencies.

14) The configuration system should be able to store and make accessible documentation about configuration properties (name, description(s), etc.)

15) The configuration system stores strings only. It is up to a module to cast strings to implied value types.
-- desire to include data types as as part of the model (rather than only accept strings)
-- desire to have utility methods for doing casting (e.g., getValueAsInt)
-- change this requirement to: The configuration system will support basic types. If no type is specified, the type will default to String.
-- basic types TBD.

16) The config system should define the set of allowable characters for use in names (e.g., no spaces, or should be java identifiers)

17) The configuration system should specifically consider if and how configuration values or sets of values from the command-line, environment variables, module.txt and other sources should be merged or provided with values from configuration files

#9 - 10/05/2009 11:54 AM - Chad Berkley

Since requirements are now decided, I will close this bug. All future posts on this subject should be made to bug 3948.

#10 - 03/27/2013 02:26 PM - Redmine Admin

Original Bugzilla ID was 4394