Module Authorizers 2

`ModuleAuthorizers` are registered in the Spring module definition. When a user attempts to perform any action in KFS, `KualiRequestProcessor` checks whether the user is active for *any* module. If not, the user will receive an immediate error. Once this check passes and the request processor hands off control to the appropriate Struts Action class implementation, that class is responsible for asking the `KualiModuleService` to authorize the user for the particular `AuthorizationType` in its implementation of the `checkAuthorization` method. The `KualiModuleService` determines which module authorizer to defer to using the package prefixes registered with the authorizer in the module definition. The `AuthorizationType` interface provides access to a target class that access is being authorized for and a name to use in any error report. The following authorization types are predefined, but there is nothing to prevent you from using your own implementations of the `AuthorizationType` interface: `Inquiry`, `Lookup`, `Document`, `AdHocRequest`, and `Default`. All but the `Default` wrap a business object or document class. So, a module authorizer gives you the ability to (for example) prevent anyone who is not in a particular workgroup from using lookups, inquiries, or documents pertaining to bank accounts. In general, action classes that use `AuthorizationType.Default` (any action classes that directly subclass `KualiAction` and do not override the `checkAuthorization` method) simply use their own class as the target, so that a module authorizer can allow anyone who is active for that module to use them. `AuthorizationType.AdHocRequest` is a bit strange in that it is preemptive. It can be used to prevent user X from even sending user Y a document, i.e. deny user Y before an action is requested of him. The base module authorizer in the financial system will only allow users who are considered active by the module to which a document belongs to initiate or receive ad hoc approval requests for documents of that type. The `KualiModuleUser` implementation for the responsible module (also registered in the module definition via specification of a `KualiModuleService` implementation) may determine that a user is active based on explicit access granted in the user document (e.g. the "active KFS user" attribute used to give access to most KFS modules), via university affiliation/role (e.g. the `KualiModuleUserBase.isActiveFacultyStaffAffiliate` check done by the `PurapUser` implementation), etc. Finer grained control for documents is possible via the document authorization framework.