PolicytoolDev

This page is part of the Policytool development documentation. It's about the architecture and methods used to implement Policytool.

Overview

The source of policytool is located in the package **org.apache.harmony.tools.policytool**. The entry point is the Main class.

The **MVC** *(Model-View-Controller)* pattern was used to implement the tool.
The *model package contains the classes modeling a policy file decomposed to entries and put into a tree or a list.
The *view package contains the GUI elements of the user interface.
The *control package contains the classes handling the policy files and policy syntax, and a main Controller class which handles the user interactions and connects the view and the model.

The model package

The base class of all entries is the PolicyEntry class. Defines an abstract method to get set the string representation of a policy entry. There are basically 2 entry types (which are 3):
*GrantEntry: - this is used to define access and grant to various system resources.
*KeystoreEntry and KeystorePasswordURLEntry: these 2 define the location and other parameters of the keystore

The grant entries might include one or several principals and permissions which are represented with the respective classes (Principal and Permission).

In order to remember the real structure of the editable policy file, extra spaces and comments are stored in CommentEntry instances.

The view package

The frame of policytool is represented by the MainFrame class. It contains a menu bar, a text field to display the edited policy file, and a tabbed pane for the 2 editor panel.

The editor panels are inherited from the EditorPanel class, which defines methods for setting/getting the edited policy text as a string, and provides an interface to check if the panel contains unsaved changes. There are 2 subclasses of EditorPanel:
*DirectTextEditorPanel: provides a text area to view and edit the policy text as-is
*GraphicalEditorPanel: provides the well-known GUI with some improvements available with swing

The data inputs regarding to the policy text are handled via BaseFormDialog-s. This dialog provides some utility methods and the 2 common OK and Cancel buttons. The classes profiting from BaseFormDialog:
*KeystoreEntryEditFormDialo: to view/edit data to locate the keystore
*GrantEntryFormDialog: to view/edit grant entries
*PrincipalEntryFormDialog: to view/edit principals
*PermissionEntryFormDialog: to view/edit permissions

Policytool contains several entities which has to be listed, has to be available for editing and removing, and provide an interface for adding new entities. Such entities are grant entries, principals, permissions. The generic ListAndEditPanel class is responsible to provide such an interface which is used parameterized when used for the listed entities.

The LAEFormDialog is a specialized FormDialog which is intended to handle the data view/edit interface for entities available through a ListAndEditPanel component. The GrantEntryFormDialog, PrincipalEntryFormDialog and PermissionEntryFormDialog classes in fact are subclasses of the LAEFormDialog class.

The WarningLogDialog class is a dialog class with a text area to store and display previous warning and error messages. New feature (compared to Sun's policytool): the warning log dialog is non-modal which is a lot more useful here because we support direct editing, and switching between the editor panels can generate errors/warnings frequently (not just when opening a new file or validating form values). The warning log can be open while we edit the policy text in any editor. If we use the "View Warnign Log" menu while it's visible, it will be centered again.

The control package

The Controller class drives the GUI and connects it to the model. It is the handler of the menu items, and it is responsible to passing over the edited policy text in case of editor panel switching. Handles the unsaved changes before certain operations (exit, open a new file, load a file).

The PolicyFileHandler class is responsible to gain the policy text out of a policy file and to save a policy text to a policy file being aware of the mandatory UTF-8 encoding.

The PolicyTextParser class is responsible to parse a policy text, and return an equivalent list of policy entries from it. If the policy text is invalid, an InvalidPolicyTextException will be thrown with a proper error message. Sun's policytool takes out all the comments from an opened policy file. It's a new feature that this parser parses and remembers comments between the entries in the policy text.

JIRA issues related to Policytool

*https://issues.apache.org/jira/browse/HARMONY-5952
*https://issues.apache.org/jira/browse/HARMONY-5949
*https://issues.apache.org/jira/browse/HARMONY-5944
*https://issues.apache.org/jira/browse/HARMONY-5927
*https://issues.apache.org/jira/browse/HARMONY-5912
*https://issues.apache.org/jira/browse/HARMONY-5898
*https://issues.apache.org/jira/browse/HARMONY-5886

Other links

Specification of the policy file: http://java.sun.com/javase/6/docs/technotes/guides/security/PolicyFiles.html