Hard Token Management Framework

Developers Handbook

Written By
Philip Vendil, philip@primekey.se
+46709885814

2007-08-01

1 Introduction/Scope

The Hard Token Management Framework contains the core components to easily develop a smart card managing application that covers the entire life cycle of a smart card or USB dongle.

Since almost every organization probably have an existing work flow in place for issuing access cards for physical access it is often preferred that a new system for VPN and computer log-on access fit in the current routines. And since almost every organization have their own work flow and policies for issuing, revoking access to their system it is very hard to write a ready-made system. Therefore have this framework been created where it is possible to use ready made components that takes care of all the tricky bits of the smart card management and it is only up to the developers to combine them in such a way that conforms with organizations policies.

The design goals of the framework is simplicity (it should be easy for the developer to create simple and easy to use applications for the end users), and flexibility (the framework should be flexible enough to cover all the aspects of hard token management).

The Hard Token Management Framework is an add-on the EJBCA (www.ejbca.org) and is developed as an Java-applet and accessed through a browser.

1.1 Who should read this document.

The intended audience of this document is Java developers interested in creating a smart card managing application. It is preferred that the developer has some knowledge about PKI and smart cards but don't need be an expert in any of the areas.

2 Document History

Version	*Date*	*Name*	*Comment*
1.0	2007-07-29	Philip Vendil	First official version of the document
1.1	2007-08-01	Philip Vendil	Updated with plug-in:able classes

Table of Contents

3 Overview

The Hard Token Management Framework is developed in Java and is basically an applet packaged in a war that is supposed to be deployed to an EJBCA installation.

The Framework is communicating with EJBCA using Web Services calls using client authenticated HTTPS. The hard token management war doesn't have to deployed to the same application server as EJBCA but the end user browser should have access to the EJBCAWS interface.

To access the tokens is done through a PKCS11 interface and the design is that it should be relatively easy to change the token an organization to avoid to be looked to one vendor. But this requires that the vendor have a good PKCS11.

4 The Main Concepts

This chapter will discuss the main concepts of the framework that a developer must know about before starting developing. This will be later be complemented in the next chapter with the tool set of interfaces (to access CA, token, settings and so on) used to perform the logic of the different components.

One of the design goal of the framework is to make the application modulized so it will be easy to create a component and to reuse already written ones. A customized component is done be creating a “Controller” that has a “View”. The View is a swing representation of the view of the component. The controller is responsible of all the logic of the component and the view should contain all the visual stuff. Each controller is responsible for delegating the controller to another controller when the current one is done with its processing. It is possible to define the “main”controller that created when the applet starts in the global properties of the framework.

The application can be themed so it isn't necessary to set the basic properties (as colour and font) to each swing component. This is done automatically. See section “Adjusting the Theme of the GUI” for more information.

4.1 Controller

A Controller is a class in charge of the logic of a components, i.e. what will be done if a button is pressed or a select list is changes that should generate some form of action.

There exists a template view in src/java/org/hardtokenmgmt/core/ui/TemplateController.java that can be copied and used as a skeleton.

4.1.1 org.hardtokenmgmt.core.ui.IController Interface

In the code snippet below in the IController interface shown but it is strongly recommended that every controller inherits the BaseController class that has a lot of help methods for getting access to the different interfaces and switching controller between controllers.

```
public interface IController {

    /**
     * Method that should return true if the logged in administrator
     * is authorized to use this controller
     *
     * @param admin the certificate of the loggen in administrator.
     * @return true if the administrator is authorized to this controller
     */
    public boolean isAuthorizedToController(X509Certificate admin);

    /**
     * Method called by the main hard token applet when
     * it gives the control of the application
     *
     * @param class path to the controller calling this controller or null
     * if this was the main controller.
     */
    public void getControl(String callingController);


    /**
     * Method called by the main applet when it add
     * the view to the main panel.
     */
    public BaseView getView();
}
```

4.1.2 Example of a Controller

Here follows an example of a simple DummyController. This will be started if you configure the setting `controller.main in the file` src/resources/globalsettings/global.conf to the the classpath of the controller.

The logic can either be set in the constructor or in the method getControl(....) . The getControl is called every time by the applet class that the controller should get the control of the application while the constructor is only called once during the execution of the applet. In order to access things from the view you have to mark those Swing components as public since they are set to private by default by the Visual Editor. Usually is this applied to buttons and similar components.

When a Controller is finished with its execution it should call the protected method switchControlTo(< class name>.`class.getName()`)

```
public class DummyController1 extends BaseController {


public DummyController1() {
  super(new DummyView1());


            // Add some logic to the button named nextButton
  getDummyView1().getNextButton().addActionListener
    new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent e) {
      LocalLog.debug("Switching control to :+DummyController2.class.getName());
      getControllerMemory().putData("USERNAME",
        getDummyView1().getUsername().getText());
      switchControlTo(DummyController2.class.getName());
      }
     });


}


private DummyView1 getDummyView1(){
  return (DummyView1) getView();
}

/**
 * Method called by the Main applet when it's time for this
 * controller to take control
 *
 * @see org.hardtokenmgmt.core.ui.IController#getControl(String)
 */
public void getControl(String callingController) {

}
/**
 * Method called by the main applet to check that
 * the administrator is authorized to this controller
 *
 * @see
org.hardtokenmgmt.core.ui.IController#isAuthorizedToController(X509Certificate)
 */
public boolean isAuthorizedToController(X509Certificate admin) {
  return true;
}

}
```

4.1.3 *org.hardtokenmgmt.core.ui.BaseController Interface*

The BaseController have the following help methods that can be used to ease the work of creating a controller. See the Interfaces chapter for more information about more detailed usage.

Method	Comment
void debug(java.lang.String msg)	Help method to print a debug message to the local log.
void debug(java.lang.String msg, java.lang.Throwable throwable)	Help method to print a debug message along with an exception to the local log.
void error(java.lang.String msg)	Help method to print a error message to the local log.
void error(java.lang.String msg, java.lang.Throwable throwable)	Help method to print a error message along with an exception to the local log.
AdministratorSettings getAdministratorSettings()	Help method retrieving the Administrator Settings Interface.
ControllerMemory getControllerMemory()	Help method returning the instance of the controller memory.
String getControllerSetting(String key)	Returns the value of the given controller setting. The controller setting should start with the implementing controllers classname (not path) in lowercase. Ex for the controller named `DummyController` calling `getControllerSetting` with value `'testsetting'` will lookup the property `dummycontroller.testsetting` in `global.properties`.
IEjbcaWS getEJBCAInterface()	Help method retrieving the EJBCA Interface.
GlobalSettings getGlobalSettings()	Help method retrieving the Global Settings Interface.
IToken getProcessableToken()	Help method returning the first processable token. Should only be used by controllers only supporting on processable token
TokenManager getTokenManager()	Help method retrieving the Token Manager Interface
void info(java.lang.String msg)	Help method to print a info message to the local log
void info(java.lang.String msg, java.lang.Throwable throwable)	Help method to print a info message along with an exception to the local log
IsAdmin()	Help method checking if the administrator have

Method	Comment
	the administrator flag set in the administrative privileges.
void switchControlTo(java.lang.String controllerClassPath)	Method that should be used when the controller is complete with its operations and want to give the control to another controller

4.2 View

A View class represents all the visual details of a component, like where i a button located and which icon will it have. The View classes have been designed to work with Eclipse Visual Editor as much as possible. For instance is language resources, images and theme looked up automatically and can be edited directly.

Tip: Make sure you have the visual editor add-on install and use “Open With” -> “Visual Editor” to open views in WYSIWG mode.

There exists a template view in src/java/org/hardtokenmgmt/core/ui/TemplateView.java that can be copied and used as a skeleton.

4.2.1 BaseView

Base view is the base class of a View that all views should inherit. It mainly is used to set the theme.

```
public abstract class BaseView extends JPanel {

	static{
		UIHelper.setTheme();
	}

	private static final long serialVersionUID = 1L;
	/**
	 * This is the default constructor
	 */
	public BaseView() {
		super();
	}



	/**
	 * This method initializes this
	 *
	 * @return void
	 */
	protected abstract void initialize();


}
```

4.2.2 Example of a View

In the following text box is example code of a view. The only thing that is required is the following line in the initialize method:

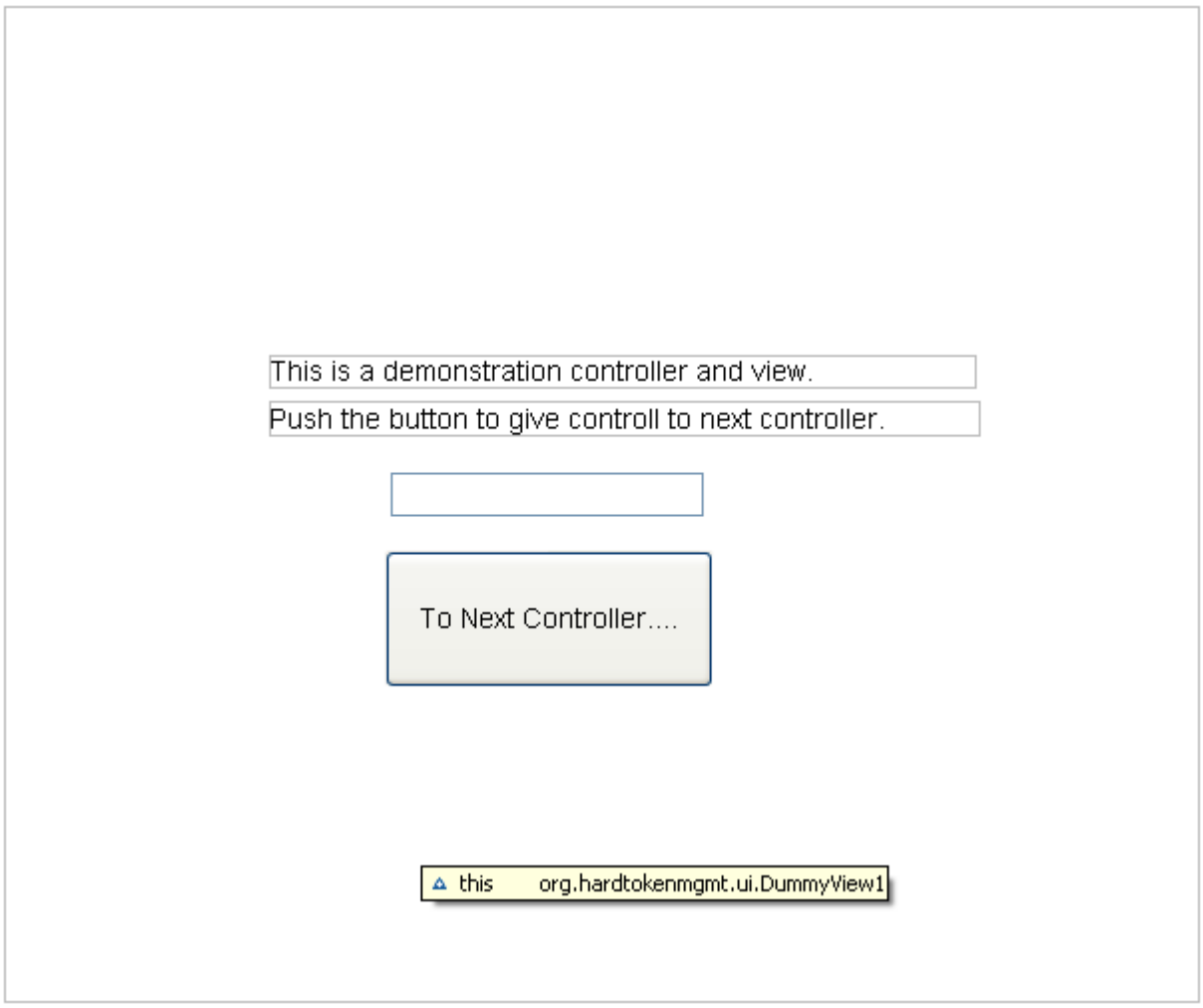
```
this.setSize(new Dimension(UIHelper.getAppletWidth(),UIHelper.getAppletHeight()));
```

To achieve internationalization of the texts use `UIHelper.getText(<property>)`, this call will automatically look up the property in the language files in src/resources/languages/languagefile_xx.properties, just add the property in the language files and it will shop up in the visual editor.

To add images and icons, place the image in the src/resources/languages/images directory and use the method `UIHelper.getImage("filename")`, you should just give the file name and no path. One exception is the top logotype (if you want one there) that should be an image of size 600x132 pixels and should always be fetched with UIHelper.getLogo().

```
public class DummyView1 extends BaseView {

	private static final long serialVersionUID = 1L;
	private JLabel dummyLabel = null;
	private JButton nextButton = null;
	private JLabel jLabel = null;
	private JTextField username = null;

	public DummyView1(){
		super();
		initialize();
	}

	@Override
	protected void initialize() {
		jLabel = new JLabel();
		jLabel.setBounds(new Rectangle(133, 198, 357, 18));
		jLabel.setText(UIHelper.getText("dummy1.message2"));
		this.setSize(new Dimension(UIHelper.getAppletWidth(),
UIHelper.getAppletHeight()));
		this.setLayout(null);
	  dummyLabel = new JLabel();
	  dummyLabel.setText(UIHelper.getText("dummy1.message1"));
	  dummyLabel.setBounds(new Rectangle(133, 175, 355, 17));


	  this.add(dummyLabel, null);
	  this.add(getNextButton(), null);
	  this.add(jLabel, null);
	  this.add(getUsername(), null);
	}

	public JButton getNextButton() {
		if (nextButton == null) {
			nextButton = new JButton();
			nextButton.setBounds(new Rectangle(191, 273, 165, 69));
			nextButton.setText(UIHelper.getText("dummy1.nextbutton"));
		}
		return nextButton;
	}


	public JTextField getUsername() {
		if (username == null) {
			username = new JTextField();
			username.setBounds(new Rectangle(194, 234, 157, 22));
			username.setVisible(true);
		}
		return username;
	}

}
```

This view will have the following visual layout:



4.3 Configuration and Building

4.3.1 Configuring the Global Settings

In the file src/resources/globalsettings/global.properties exists all the global settings used to configure the applet. In this file should all properties be set to the application that is general for all users. Example of properties are supported tokens and which controller that is the main one.

4.3.2 Adjusting the Theme of the GUI

There also exists a property file defining the theme of the Swing GUI so it isn't necessary to define all the characteristics like font, colour for each button and label.

4.3.3 Language Resources

In the directory src/resources/languages are the language resource files. They should be in the format languagefile_xx.properties where xx should be the locale name.

When a language resource is looked up it first checks the file custom_languagefile (see the customization chapter) then in the language file that matches the computers locale and if it isn't found in neither the value in languagefile_en.properties will be used.

4.4 Other Classes

4.4.1 org.hardtokenmgmt.core.ui.HardTokenManagementApplet

This class is the main applet. It takes care of the creation and termination of the applet as well as extracting the information about the logged in administrator. There should be no need to alter this applet class just to configure the starting controller in the global.properties.

5 The Interfaces

5.1 EJBCAWS Interface

The EJBCAWS interface is intended to be access from a controller. If the BaseController is inherited then there exists a protected help method getEJBCAInterface() that returns the interface. Otherwise can the interface be fetched with the static method InterfaceFactory.getEjbcaInterface().

When developing and testing a controller it can be convenient to not use a regular EJBCA installation but a simple dummy interface. This is set in `ejbcaws.implementation` property of the global.properties file. One dummy implementation that can be used is the `org.hardtokenmgmt.core.ejbca.DummyEjbcaWS class.`

The EJBCA WS Interface have the following methods.

```
/**
 * Interface the the EJBCA RA WebService. Contains the following methods:
 *
 * editUser    : Edits/adds  userdata
 * findUser    : Retrieves the userdata for a given user.
 * findCerts   : Retrieves the certificates generated for a user.
 * pkcs10Req   : Generates a certificate using the given userdata and the public key from the PKCS10
 * pkcs12Req   : Generates a PKCS12 keystore (with the private key) using the given userdata
 * revokeCert  : Revokes the given certificate.
 * revokeUser  : Revokes all certificates for a given user, it's also possible to delete the user.
 * revokeToken : Revokes all certificates placed on a given hard token
 * checkRevokationStatus : Checks the revokation status of a certificate.
 * isAuthorized : Checks if an admin is authorized to an resource
 * fetchUserData : Method used to fetch userdata from an existing UserDataSource
 * genTokenCertificates : Method used to add information about a generated hardtoken
 * existsHardToken : Looks up if a serial number already have been generated
 * getHardTokenData : Method fetching information about a hard token given it's hard token serial number.
 * getHardTokenDatas: Method fetching all hard token informations for a given user.
 * republishCertificate : Method performing a republication of a selected certificate
 * isApproved : Looks up if a requested action have been approved by an authorized administrator or not
 * customLog  : Logs a CUSTOM_LOG event to the logging system
 * deleteUserDataFromSource : Method used to remove user data from a user data source
```

```
 * getCertificate : Returns a certificate given its issuer and serial number
 *
 * Observere: All methods have to be called using client authenticated https
 * otherwise will a AuthorizationDenied Exception be thrown.
 *
 * @author Philip Vendil
 * $Id: IEjbcaWS.java,v 1.6 2007/07/05 05:55:08 herrvendil Exp $
 */
public interface IEjbcaWS {

	public static final int CUSTOMLOG_LEVEL_INFO  = 1;
	public static final int CUSTOMLOG_LEVEL_ERROR = 2;

	/**
	 * Method that should be used to edit/add a user to the EJBCA database,
	 * if the user doesn't already exists it will be added othervise it will be
	 * overwritten.
	 *
	 * Observe: if the user doesn't already exists, it's status will always be set to 'New'.
	 *
	 * Authorization requirements: the client certificate must have the following priviledges set
	 * - Administrator flag set
	 * - /administrator
	 * - /ra_functionality/create_end_entity and/or edit_end_entity
	 * - /endentityprofilesrules/<end entity profile of user>/create_end_entity and/or edit_end_entity
	 * - /ca/<ca of user>
	 *
	 * @param userdata contains all the information about the user about to be added.
	 * @param clearPwd indicates it the password should be stored in cleartext, requeried
	 * when creating server generated keystores.
	 * @throws EjbcaException
	 */
	public abstract void editUser(UserDataVOWS userdata)
				throws AuthorizationDeniedException,
				UserDoesntFullfillEndEntityProfile, EjbcaException,
				ApprovalException, WaitingForApprovalException;

	/**
	 * Retreives information about a user in the database.
	 *
	 * Authorization requirements: the client certificate must have the following priviledges set
	 * - Administrator flag set
	 * - /administrator
	 * - /ra_functionality/view_end_entity
	 * - /endentityprofilesrules/<end entity profile of matching users>/view_end_entity
	 * - /ca/<ca of matching users>
	 *
	 * @param username, the unique username to search for
	 * @return a array of UserDataVOWS objects (Max 100) containing the information about the user or null if user doesn't exists.
	 * @throws AuthorizationDeniedException if client isn't authorized to request
	 * @throws IllegalQueryException if query isn't valid
	 * @throws EjbcaException
	 */

	public abstract List<UserDataVOWS> findUser(UserMatch usermatch)
				throws AuthorizationDeniedException, IllegalQueryException,
				EjbcaException;

	/**
	 * Retreives a collection of certificates generated for a user.
	 *
	 * Authorization requirements: the client certificate must have the following priviledges set
	 * - Administrator flag set
	 * - /administrator
	 * - /ra_functionality/view_end_entity
	 * - /endentityprofilesrules/<end entity profile of the user>/view_end_entity
	 * - /ca/<ca of user>
```

```
 *
 * @param username a unique username
 * @param onlyValid only return valid certs not revoked or expired ones.
 * @return a collection of X509Certificates or null if no certificates could be found
 * @throws AuthorizationDeniedException if client isn't authorized to request
 * @throws NotFoundException if user cannot be found
 * @throws EjbcaException
 */

public abstract List<Certificate> findCerts(String username,
                        boolean onlyValid) throws AuthorizationDeniedException,
                        NotFoundException, EjbcaException;

/**
 * Method to use to generate a certificate for a user. The method must be preceded by
 * a editUser call, either to set the userstatus to 'new' or to add nonexisting users.
 *
 * Observe, the user must first have added/set the status to new with edituser command
 *
 * Authorization requirements: the client certificate must have the following priviledges set
 * - Administrator flag set
 * - /administrator
 * - /ra_functionality/view_end_entity
 * - /endentityprofilesrules/<end entity profile of the user>/view_end_entity
 * - /ca_functionality/create_certificate
 * - /ca/<ca of user>
 *
 * @param username the unique username
 * @param password the password sent with editUser call
 * @param pkcs10 the PKCS10 (only the public key is used.)
 * @param hardTokenSN If the certificate should be connected with a hardtoken, it is
 * possible to map it by give the hardTokenSN here, this will simplyfy revokation of a tokens
 * certificates. Use null if no hardtokenSN should be assiciated with the certificate.
 * @return the generated certificate.
 * @throws AuthorizationDeniedException if client isn't authorized to request
 * @throws NotFoundException if user cannot be found
 */

public abstract Certificate pkcs10Req(String username, String password,
                        String pkcs10, String hardTokenSN)
                        throws AuthorizationDeniedException, NotFoundException,
                        EjbcaException;

/**
 * Method to use to generate a server generated keystore. The method must be preceded by
 * a editUser call, either to set the userstatus to 'new' or to add nonexisting users and
 * the users token should be set to SecConst.TOKEN_SOFT_P12.
 *
 * Authorization requirements: the client certificate must have the following priviledges set
 * - Administrator flag set
 * - /administrator
 * - /ra_functionality/view_end_entity
 * - /endentityprofilesrules/<end entity profile of the user>/view_end_entity
 * - /ca_functionality/create_certificate
 * - /ca/<ca of user>
 *
 * @param username the unique username
 * @param password the password sent with editUser call
 * @param hardTokenSN If the certificate should be connected with a hardtoken, it is
 * possible to map it by give the hardTokenSN here, this will simplyfy revokation of a tokens
 * certificates. Use null if no hardtokenSN should be assiciated with the certificate.
 * @param keyspec that the generated key should have, examples are 1024 for RSA or prime192v1 for ECDSA.
 * @param keyalg that the generated key should have, RSA, ECDSA. Use one of the constants in
 * CATokenConstants.org.ejbca.core.model.ca.catoken.KEYALGORITHM_XX.
 * @return the generated keystore
 * @throws AuthorizationDeniedException if client isn't authorized to request
 * @throws NotFoundException if user cannot be found
```

```
 */

public abstract KeyStore pkcs12Req(String username, String password,
                        String hardTokenSN, String keyspec, String keyalg)
                        throws AuthorizationDeniedException, NotFoundException,
                        EjbcaException;

/**
 * Method used to revoke a certificate.
 *
 * * Authorization requirements: the client certificate must have the following priviledges set
 * - Administrator flag set
 * - /administrator
 * - /ra_functionality/revoke_end_entity
 * - /endentityprofilesrules/<end entity profile of the user owning the cert>/revoke_end_entity
 * - /ca/<ca of certificate>
 *
 * @param issuerDN of the certificate to revoke
 * @param certificateSN of the certificate to revoke
 * @param reason for revokation, one of RevokedCertInfo.REVOKATION_REASON_ constants,
 * or use RevokedCertInfo.NOT_REVOKED to unrevoke a certificate on hold.
 * @throws AuthorizationDeniedException if client isn't authorized.
 * @throws NotFoundException if certificate doesn't exist
 */

public abstract void revokeCert(String issuerDN, String certificateSN,
                        int reason) throws AuthorizationDeniedException, NotFoundException,
                        EjbcaException;

/**
 * Method used to revoke all a users certificates. It is also possible to delete
 * a user after all certificates have been revoked.
 *
 * Authorization requirements: the client certificate must have the following priviledges set
 * - Administrator flag set
 * - /administrator
 * - /ra_functionality/revoke_end_entity
 * - /endentityprofilesrules/<end entity profile of the user>/revoke_end_entity
 * - /ca/<ca of users certificate>
 *
 * @param username unique username i EJBCA
 * @param reasonfor revokation, one of RevokedCertInfo.REVOKATION_REASON_ constants
 * or use RevokedCertInfo.NOT_REVOKED to unrevoke a certificate on hold.
 * @param deleteUser deletes the users after all the certificates have been revoked.
 * @throws AuthorizationDeniedException if client isn't authorized.
 * @throws NotFoundException if user doesn't exist
 */
public abstract void revokeUser(String username, int reason,
                        boolean deleteUser) throws AuthorizationDeniedException,
                        NotFoundException, EjbcaException;

/**
 * Method used to revoke all certificates mapped to one hardtoken.
 *
 * Authorization requirements: the client certificate must have the following priviledges set
 * - Administrator flag set
 * - /administrator
 * - /ra_functionality/revoke_end_entity
 * - /endentityprofilesrules/<end entity profile of the user owning the token>/revoke_end_entity
 * - /ca/<ca of certificates on token>
 *
 * @param hardTokenSN of the hardTokenSN
 * @param reasonfor revokation, one of RevokedCertInfo.REVOKATION_REASON_ constants
 * @throws AuthorizationDeniedException if client isn't authorized.
 * @throws NotFoundException if token doesn't exist
 */
```

```
public abstract void revokeToken(String hardTokenSN, int reason)
                        throws RemoteException, AuthorizationDeniedException,
                        NotFoundException, EjbcaException;

/**
 * Method returning the revokestatus for given user
 *
 * Authorization requirements: the client certificate must have the following priviledges set
 * - Administrator flag set
 * - /administrator
 * - /ca/<ca of certificate>
 *
 * @param issuerDN
 * @param certificateSN a hexadecimal string
 * @return the revokestatus of null i certificate doesn't exists.
 * @throws AuthorizationDeniedException if client isn't authorized.
 * @see org.ejbca.core.protocol.ws.RevokeStatus
 */

public abstract RevokeStatus checkRevokationStatus(String issuerDN,
                        String certificateSN) throws AuthorizationDeniedException,
                        EjbcaException;

/**
 * Method checking if a user is authorixed to a given resource
 *
 * Authorization requirements: a valid client certificate
 *
 * @param resource the access rule to test
 * @return true if the user is authorized to the resource othervise false.
 * @throws AuthorizationDeniedException if client isn't authorized.
 * @see org.ejbca.core.protocol.ws.RevokeStatus
 */
public abstract boolean isAuthorized(String resource) throws EjbcaException;

/**
 * Method used to fetch userdata from an existing UserDataSource.
 *
 * Authorization requirements:
 * - Administrator flag set
 * - /administrator
 * - /userdatasourcesrules/<user data source>/fetch_userdata (for all the given user data sources)
 * - /ca/<all cas defined in all the user data sources>
 *
 * If not turned of in jaxws.properties then only a valid certificate required
 *
 *
 * @param userDataSourceNames a List of User Data Source Names
 * @param searchString to identify the userdata.
 * @return a List of UserDataSourceVOWS of the data in the specified UserDataSources, if no user data is found will an empty list be
 * returned.
 * @throws UserDataSourceException if an error occured connecting to one of
 * UserDataSources.
 */
public abstract List<UserDataSourceVOWS> fetchUserData(
                        List<String> userDataSourceNames, String searchString)
                        throws UserDataSourceException, EjbcaException, AuthorizationDeniedException;

/**
 * Method used to add information about a generated hardtoken
 *
 * Authorization requirements:
 * If the caller is an administrator
 * - Administrator flag set
 * - /administrator
 * - /ra_functionality/create_end_entity and/or edit_end_entity
 * - /endentityprofilesrules/<end entity profile of user>/create_end_entity and/or edit_end_entity
```

```
 * - /ra_functionality/revoke_end_entity (if overwrite flag is set)
 * - /endentityprofilesrules/<end entity profile of user>/revoke_end_entity (if overwrite flag is set)
 * - /ca_functionality/create_certificate
 * - /ca/<ca of all requested certificates>
 * - /hardtoken_functionality/issue_hardtokens
 *
 * If the user isn't an administrator will it be added to the queue for approval.
 *
 * @param userData of the user that should be generated
 * @param tokenRequests a list of certificate requests
 * @param hardTokenData data containin PIN/PUK info
 * @param hardTokenSN Serial number of the generated hard token.
 * @param overwriteExistingSN if the the current hardtoken should be overwritten instead of throwing HardTokenExists exception.
 * If a card is overwritten, all previous certificates on the card is revoked.
 * @param revocePreviousCards tells the service to revoke old cards issued to this user. If the present card have the label
 * TEMPORARY_CARD
 * old cards is set to CERTIFICATE_ONHOLD otherwice UNSPECIFIED.
 * @return a List of the generated certificates.
 * @throws AuthorizationDeniedException if the administrator isn't authorized.
 * @throws WaitingForApprovalException if the caller is a non-admin a must be approved before it is executed.
 * @throws HardTokenExistsException if the given hardtokensn already exists.
 * @throws ApprovalRequestExpiredException if the request for approval have expired.
 * @throws ApprovalException  if error happended with the approval mechanisms
 * @throws WaitingForApprovalException if the request haven't been processed yet.
 * @throws ApprovalRequestExecutionException if the approval request was rejected
 */

public abstract List<TokenCertificateResponseWS> genTokenCertificates(
                        UserDataVOWS userData,
                        List<TokenCertificateRequestWS> tokenRequests,
                        HardTokenDataWS hardTokenData,
                        boolean overwriteExistingSN,
                        boolean revocePreviousCards) throws AuthorizationDeniedException,
                        WaitingForApprovalException, HardTokenExistsException,
                        UserDoesntFullfillEndEntityProfile, ApprovalException,
                        EjbcaException, ApprovalRequestExpiredException, ApprovalRequestExecutionException;

/**
 * Looks up if a serial number already have been generated
 *
 * Authorization requirements: A valid certificate
 *
 * @param hardTokenSN the serial number of the token to look for.
 * @return true if hard token exists
 * @throws EjbcaException if error occured server side
 */
public abstract boolean existsHardToken(String hardTokenSN)
                        throws EjbcaException;

/**
 * Method fetching information about a hard token given it's hard token serial number.
 *
 * If the caller is an administrator
 * - Administrator flag set
 * - /administrator
 * - /ra_functionality/view_hardtoken
 * - /endentityprofilesrules/<end entity profile of user>/view_hardtoken
 * - /endentityprofilesrules/<end entity profile of user>/view_hardtoken/puk_data (if viewPUKData = true)
 * - /ca/<ca of user>
 *
 * If the user isn't an administrator will it be added to the queue for approval.
 *
 * @param hardTokenSN of the token to look for.
 * @param viewPUKData if PUK data of the hard token should be returned.
 * @param boolean onlyValidCertificates of all revoked and expired certificates should be filtered.
 * @return the HardTokenData
 * @throws HardTokenDoesntExistsException if the hardtokensn don't exist in database.
```

```
     * @throws EjbcaException if an exception occured on server side.
     * @throws ApprovalRequestExpiredException if the request for approval have expired.
     * @throws ApprovalException  if error happended with the approval mechanisms
     * @throws WaitingForApprovalException if the request haven't been processed yet.
     * @throws ApprovalRequestExecutionException if the approval request was rejected
     */
    public abstract HardTokenDataWS getHardTokenData(String hardTokenSN, boolean viewPUKData, boolean onlyValidCertificates)
                        throws AuthorizationDeniedException,
                        HardTokenDoesntExistsException, EjbcaException, ApprovalException, ApprovalRequestExpiredException,
WaitingForApprovalException, ApprovalRequestExecutionException;

    /**
     * Method fetching all hard token informations for a given user.
     *
     * If the caller is an administrator
     * - Administrator flag set
     * - /administrator
     * - /ra_functionality/view_hardtoken
     * - /endentityprofilesrules/<end entity profile of user>/view_hardtoken
     * - /endentityprofilesrules/<end entity profile of user>/view_hardtoken/puk_data (if viewPUKData = true)
     *
     *
     * @param username to look for.
     * @param viewPUKData if PUK data of the hard token should be returned.
     * @param boolean onlyValidCertificates of all revoked and expired certificates should be filtered.
     * @return a list of the HardTokenData generated for the user never null.
     * @throws EjbcaException if an exception occured on server side.
     */
    public abstract List<HardTokenDataWS> getHardTokenDatas(String username, boolean viewPUKData, boolean onlyValidCertificates)
                        throws AuthorizationDeniedException, EjbcaException;

    /**
     * Method performing a republication of a selected certificate
     *
     * Authorization requirements:
     * - Administrator flag set
     * - /administrator
     * - /ra_functionality/view_end_entity
     * - /endentityprofilesrules/<end entity profile of the user>/view_end_entity
     * - /ca/<ca of user>
     *
     * @param serialNumberInHex of the certificate to republish
     * @param issuerDN of the certificate to republish
     * @throws AuthorizationDeniedException if the administratior isn't authorized to republish
     * @throws PublisherException if something went wrong during publication
     * @throws EjbcaException if other error occured on the server side.
     */
    public abstract void republishCertificate(String serialNumberInHex,
                        String issuerDN) throws AuthorizationDeniedException,
                        PublisherException, EjbcaException;

    /**
     * Looks up if a requested action have been approved by an authorized administrator or not
     *
     * Authorization requirements: A valid certificate
     *
     * @param approvalId unique id for the action
     * @return the number of approvals left, 0 if approved othervis is the ApprovalDataVO.STATUS constants returned indicating the statys.
     * @throws ApprovalException if approvalId doesn't exists
     * @throws ApprovalRequestExpiredException Throws this exception one time if one of the approvals have expired, once notified it
wount throw it anymore.
     * @throws EjbcaException if error occured server side
     */
    public abstract int isApproved(int approvalId) throws ApprovalException,
                        EjbcaException, ApprovalRequestExpiredException;

    /**
```

```
     * Generates a Custom Log event in the database.
     *
     * Authorization requirements:
     * - Administrator flag set
     * - /administrator
     * - /log_functionality/log_custom_events
     *
     * @param level of the event, one of IEjbcaWS.CUSTOMLOG_LEVEL_ constants
     * @param type userdefined string used as a prefix in the log comment
     * @param caname of the ca related to the event, use null if no specific CA is related.
     * Then will the ca of the administrator be used.
     * @param username of the related user, use null if no related user exists.
     * @param certificate that relates to the log event, use null if no certificate is related
     * @param msg message data used in the log comment. The log comment will have
     * a syntax of '<type> : <msg'
     * @throws AuthorizationDeniedException if the administrators isn't authorized to log.
     * @throws EjbcaException if error occured server side
     */
    public abstract void customLog(int level, String type, String cAName, String username, Certificate certificate, String msg) throws
AuthorizationDeniedException, EjbcaException;

    /**
     * Special method used to remove existing used data from a user data source.
     *
     * Important removal functionality of a user data source is optional to
     * implement so it isn't certain that this method works with the given
     * user data source.
     *
     * Authorization requirements
     * - Administrator flag set
     * - /administrator
     * - /userdatasourcesrules/<user data source>/remove_userdata (for all the given user data sources)
     * - /ca/<all cas defined in all the user data sources>
     *
     *
     * @param userDataSourceName the names of the userdata source to remove from
     * @param searchString the search string to search for
     * @param removeMultipleMatch if multiple matches of a search string should be removed othervise is none removed.
     * @return true if the user was remove successfully from at least one of the user data sources.
     * @throws AuthorizationDeniedException if the user isn't authorized to remove userdata from any of the specified user data sources
     * @throws MultipleMatchException if the searchstring resulted in a multiple match and the removeMultipleMatch was set to false.
     * @throws UserDataSourceException if an error occured during the communication with the user data source.
     * @throws EjbcaException if error occured server side
     */
    public abstract boolean deleteUserDataFromSource(List<String> userDataSourceNames, String searchString, boolean
removeMultipleMatch) throws AuthorizationDeniedException, MultipleMatchException, UserDataSourceException, EjbcaException;

    /**
     * Method to fetch a issued certificate.
     *
     * Authorization requirements
     * - A valid certificate
     * - /ca_functionality/view_certificate
     * - /ca/<of the issing CA>
     *
     * @param certSNinHex the certificate serial number in hexadecimal representation
     * @param issuerDN the issuer of the certificate
     * @return the certificate (in WS representation) or null if certificate couldn't be found.
     * @throws AuthorizationDeniedException if the calling administrator isn't authorized to view the certificate
     * @throws EjbcaException if error occured server side
     */
    public abstract Certificate getCertificate(String certSNinHex, String issuerDN)   throws AuthorizationDeniedException, EjbcaException;
}
```

5.2 Token Manager Interfaces

The Token Manager interface is the main interface for accessing and managing connected tokens. It's in charge of maintaining the available slots (card readers) and the tokens (smartcards, usb-dongles) inserted into them.

The Token Manager have the following methods:

```
public interface IToken {

/**
 * Constants to all keys on a token, should only be used
 * with the getCertificates method
 */
public static String KEYTYPE_ALL = "ALL";
/**
 * Constant that refers the basic auth key on the token.
 * All type of IToken implementations might not support all types.
 */
public static String KEYTYPE_AUTH = "AUTH";
/**
 * Constant that refers the signing key on the token.
 * All type of IToken implementations might not support all types.
 */
public static String KEYTYPE_SIGN = "SIGN";

/**
 * Constant that refers the encryption key on the token.
 * All type of IToken implementations might not support all types.
 */
public static String KEYTYPE_ENC = "ENC";

/**
 * Constant that refers to the basic PIN that usually protects the
 * AUTH and ENC key
 * All type of IToken implementations might not support all types.
 */
public static String PINTYPE_BASIC = "BASIC";
/**
 * Constant that refers to the basic PIN that usually protects the
 * AUTH and ENC key
 * All type of IToken implementations might not support all types.
 */
public static String PINTYPE_SIGN = "SIGN";

/**
 * Constant indicating that a RSA key should be generated
 */
public static String KEYALG_RSA = "RSA";

/**
 * Constant indicating that data objects should be processed
 */
public static String OBJECTTYPE_DATA = "DATA";


/**
 * Method that should return the current hard token serial number of the card.
 * If the token doesn't have any serialnumber yet should null be returned.
 */
public String getHardTokenSN() throws TokenException;

/**
```

```
     * Method that should be called by the TokenManager only when creating
     * an instance.
     */
    public void init(boolean useVirtualSlots, Token token) throws TokenException;

    /**
     * Method used for knowing if an implementation supports
     * the current token in process.
     *
     * @param token the current token inserted into the slot.
     *
     * @return true if the current IToken implementation supports this token.
     *
     * @throws TokenException if a pkcs11 related problem occurs
     */
    public boolean isTokenSupported(Token token) throws TokenException;

    /**
     * Method that should return the tokens supported PIN types
     * Either PINTYPE_ constants or customdefined labels
     */
    public String[] getSupportedPINTypes();

    /**
     * Method that should initialize a token, cards that non-ereasable
     * should throw OperationNotSupportedException
     *
     * Keys protected by each PIN may be genereated but isn't required
     *
     * @param tokenlabel the label of the token, can be null if no tokenlabelshould be set.
     * @param pintypes an array of PINTYPE_contants or custom label strings. Indicates which key that
```

```
     * should have
     * which PIN and PUK
     * @param pins an array of pin codes used in the initialization, should be used
     * with the keytype array get hold of key to use.
     * be defined in the getKeyIndex type.
     * @param puks  an array of puk codes for each PIN
     *
     * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     */
    public void initToken(String tokenlabel, String[] pintypes, String[] pins, String[] puks) throws
    OperationNotSupportedException, TokenException;


    /**
     * Method that should clear a token, either erase the card completely
     * or remove all objects on the card for non-ereasable tokens.
     *
     * @param pintypes optional parameter used by some cards with non-erasable filesystem and
     * needs the PUK code to erease some areas. Should contain an array of PINTYPE_contants
     * or custom label strings
     * @param puks optional parameter used by some cards with non-erasable filesystem and
     * needs the PUK code to erease some areas.
     *
     * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     */
    public void clearToken(String[] pintypes, String[] puks) throws OperationNotSupportedException,
    TokenException;

    /**
     * Method that should generate a key on the card with the specified algorithm.
     *
     * @param pintype one of the PINTYPE_ indicating the PIN that should protect the key
     * @param pin the pin to unlock  (May not be required, then can null be used)
     * @param basicpin pin to unlock the basic area, might be used for signature keys
     * @param keytype one of the KEYTYPE_ constants or the label of the object for custom keys
     * @param algorithm one of the KEYALG_ constants
     * @param keysize the size of the key
     * @param label a reference to the key to use.
     *
     * @throws ObjectAlreadyExistsException if the keytype already exists.
     * @throws OperationNotSupportedException  if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException  for other token related failures.
     */
    public void genKey(String pintype, String pin, String basicpin, String keytype, String algorithm,
    int keysize, String label) throws ObjectAlreadyExistsException, OperationNotSupportedException,
    TokenException;

    /**
     * Method that generates a PKCS10 request using the specified key.
     *
     * @param keytype which key that should be used for the request.
     * @param pintype of the PIN needed to unlock the token
     * @param pin the pin to unlock  (May not be required, then can null be used)
     * @return a PKCS10CertificateRequest for the specified key.
     * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     */
    public PKCS10CertificationRequest genPKCS10(String keytype, String pintype, String pin) throws
    OperationNotSupportedException, TokenException;

    /**
     * Downloads a certificate to the token. Should mainly be used for
     * root certificates.
     *
```

```
     * @param label the label used to mark the object on the token.
     * @param pintype of the PIN needed to unlock the token
     * @param pin the pin to unlock  (May not be required, then can null be used)
     * @param basicpin to unlock the certificate store.
     *
     * @throws ObjectAlreadyExistsException if a certificate with the label already exists.
     * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     */
    public void downloadCert(String label,String pintype, String pin, String basicpin, X509Certificate
    cert)throws ObjectAlreadyExistsException, OperationNotSupportedException, TokenException;

    /**
     * Method that downloads a keystores key and certificate on the token.
     *
     * @param keytype  one of the KEYTYPE_ constants or the label of the object for custom keys
     * @param pintype of the PIN needed to unlock the token
     * @param pin the pin to unlock  (May not be required, then can null be used)
     * @param certLabel label the label used to mark the object on the token.
     * @param keyStore the java keystore to extract the keystore from
     * @param keyStorePasswd the passwordused to lock the keystore
     *
     * @throws ObjectAlreadyExistsException if a certificate or key with the label already exists.
     * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     */
    public void downloadKeyStore(String keytype, String pintype, String pin, String certLabel, KeyStore
    keyStore, String keyStorePasswd)throws ObjectAlreadyExistsException, OperationNotSupportedException,
    TokenException;

    /**
     * Method that retrieves all certificates stored on the token.
     *
     * @param pintype of the PIN that are connected to the certificates
     *
     * @return a Collection of X509Certificate
     * @throws OperationNotSupportedException if this operation isn't supported
     * @throws TokenException for other token related failures.
     */
    public Collection getCertificates(String pintype) throws OperationNotSupportedException,
    TokenException;

    /**
     * Method that tries to find a certificate stored on the token wiht
     * the specified label. The method will traverse through all supported
     * pin types.
     *
     * @param certficateLabel of the certficate to look for
     *
     * @return the X509Certificate on null of it couldn't be found.
     * @throws OperationNotSupportedException if this operation isn't supported
     * @throws TokenException for other token related failures.
     */
    public X509Certificate getCertificate(String certficateLabel) throws OperationNotSupportedException,
    TokenException;


    /**
     * Method removing the specified certificate from the token.
     *
     * @param pintype of the PIN needed to unlock the token
     * @param pin the pin to unlock  (May not be required, then can null be used)
     * @param basicpin pin to unlock the basic area, might be used for signature certificates
     * @param cert
     * @throws OperationNotSupportedException  if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
```

```
     */
    public void removeCertificate(String pintype, String pin, String basicpin, X509Certificate cert)
    throws OperationNotSupportedException, TokenException;

    /**
     * Method to get all labels of the keys (private) on the card.
     *
     * @param pintype of the PIN needed to unlock the key
     * @return a collection of String
     * @throws OperationNotSupportedException   if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     */

    public Collection getKeyLabels(String pintype)throws OperationNotSupportedException, TokenException;

    /**
     * Method removing a specified key from a card
     *
     * @param pintype of the PIN needed to unlock the token
     * @param pin the pin to unlock  (May not be required, then can null be used)
     * @param basicpin pin to unlock the basic area, might be used for signature certificates
     * @param label of the key
     * @throws OperationNotSupportedException  if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     */
    public void removeKey(String pintype, String pin, String basicpin, String label) throws
    OperationNotSupportedException, TokenException;


    /**
     * Adds a object (data, domainparameter) to the token
     *
     * @param pintype of the PIN that should protect the key, null for no protection
     * @param pin the pin to unlock the private area (May not be required, then can null be used)
     * @param object a  object to add to the token
     * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     *
     *
     */
    public void addObject(String pintype, String pin, IObject object) throws
    OperationNotSupportedException, TokenException;

    /**
     * Returns all objects of type (data or domain parameters) stored on the card
     * @param pintype of the PIN needed to unlock the token, if neccessary
     * @param pin the pin to unlock  (May not be required, then can null be used)
     * @param objectType of object to return, one of the OBJECTTYPE_ constants
     * @return a Collection of IObject
     * @throws OperationNotSupportedException if this operation isn't supported
     * @throws TokenException for other token related failures.
     *
     */
    public Collection getObjects(String pintype, String pin, String objectType)  throws
    OperationNotSupportedException, TokenException;

    /**
     * Method that removes a Data or Domain Parameters object from the card
     *
     * @param pintype of the PIN needed to unlock the token
     * @param pin the pin to unlock  (May not be required, then can null be used)
     * @param object the iaik.pkcs.pkcs11.objects.Object object to remove
     * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
     * supported
     * @throws TokenException for other token related failures.
     *
     */
```

```
public void removeObject(String pintype, String pin, IObject object)   throws
OperationNotSupportedException, TokenException;


/**
 * Method that returns the current PIN related info about the
 * given PIB
 *
 * @param pintype one of the PINTYPE_ constants
 * @return a PINInfo object
 *
 * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
 * supported
 * @throws TokenException for other token related failures.
 */
public PINInfo getPINInfo(String pintype) throws OperationNotSupportedException, TokenException;

/**
 * Method used to change the value of the
 *
 * @param pintype one of the PINTYPE_ constants
 * @param oldpin the old pinvalue
 * @param newpin the new pinvalue
 *
 * @return a PIN info object containing the current status of the PIN
 * @throws OperationNotSupportedException
 * @throws TokenException
 */
public PINInfo changePIN(String pintype, String oldpin, String newpin) throws
OperationNotSupportedException, TokenException;

/**
 * Method used to login to a PIN in order
 * to be able to perform operations on the card,
 * this is not the same as unblock PIN
 *
 * @param pintype one of the PINTYPE_ constants
 * @param pin the pin that should be used for login
 *
 * @return a PIN info object containing the current status of the PIN
 * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
 * supported
 * @throws TokenException for other token related failures.
 */
public PINInfo unlockPIN(String pintype, String pin) throws OperationNotSupportedException,
TokenException;

/**
 * Method used to enter the PUK code to unblock a PIN code
 * after to many erronious tries.
 *
 * @param pintype  one of the PINTYPE_ constants
 * @param puk the PUK that should be used to unblock
 * @param newpin the new PIN.
 * @return a PINInfo object indicating the new status of the PIN
 *
 * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
 * supported
 * @throws TokenException for other token related failures.
 */
public PINInfo unblockPIN(String pintype, String puk, String newpin) throws
OperationNotSupportedException, TokenException;

/**
 * Method used to block a PIN, used for administrators to later change it
 *
 * @param pintype  one of the PINTYPE_ constants
 * @return a PINInfo object indicating the new status of the PIN
 *
```

```
 * @throws OperationNotSupportedException if this operation or some of it's parameters isn't
 * supported
 * @throws TokenException for other token related failures.
 */
public PINInfo blockPIN(String pintype) throws OperationNotSupportedException, TokenException;


/**
 * Method instructing the token to clear it's certificate cache if it have any.
 * It's up to the implementation if there should be any caching of certificates or not.
 */
public void clearCertificateCache();
}
```

5.2.1 org.hardtokenmgmt.core.token.BaseToken

There exists a BaseToken containing much of the common functionality of accessing tokens through a PKCS11 interface. So inheriting this class can save a lot of work when implementing support for new tokens.

5.2.2 Implementing support for your own token.

If you are going to support a new token you should create a class implementing the org.hardtokenmgmt.core.token.IToken interface and if possible inherit the BaseToken.

It is preferable that you access to tokens through the PKCS11 API from IAIK (See Java API documentation for IAIK) but it is nothing that says it isn't possible to implement calls through JNI, the OpenCard framework

After implementing a IToken make sure you have added as supported in the setting `token.implementations in` global.properties. And after that run it through the test script described in the 'Testing the Framework' chapter.

5.3 Global Settings

It possible to configure the framework centrally by a global property file that defines the characteristics of the application. These are built into the applet and accessed through the getGlobalSettings() call in the BaseController, but can alternatively be accessed through the InterfaceFactory.getGlobalSettings();

Here are some of the global settings can be configured, for more details look into the actual property file.

Setting	*Description*	*Default Value*
`controller.main`	Setting indicating which controller that should start the application. Should point to a class in the class path implementing the org.hardtokenmgmt.core.ui.IController	`org.hardtokenmgmt.ui.AdminConfigController`

Setting	*Description*	*Default Value*
	interface`org.hardtokenmgmt.ui.AdminConfigController`	
`pkcs11.name`	Setting defining the name of the pkcs11 to be used by the applet. The pkcs11 must be in the system path.	`SetToki.dll`
`token.implementations`	Setting defining available token implementation, should contain a ',' separated list of class-path of IToken implementations. When finding which class to use, its isTokenSupported method is called. The first one returning true in the list is used when processing tokens	`org.hardtokenmgmt.core.token.SetCos431InstantEIDToken,org.hardtokenmgmt.core.token.SetCos441InstantEIDToken`
`token.admincertmatch`	Setting indicating with field in DN that should be matched with the administrator certificate to avoid processing slots with the administrator card inserted. Valid values are SN (Serial Number in DN (not cert serial)) and CN (Common Name)	`CN`
`token.alwaysrequireadmintoken`	Used to always require the administrator card to be inserted into the reader. And gives an error messages whenever the card is removed.	`True`
`ejbcaws.implementation`	EJBCA WS implementation. Uncomment to set a testing EJBCA WS implementation and not to connect to the real EJBCA server	`org.hardtokenmgmt.core.ejbca.DummyEjbcaWS`
`ejbcaws.url`	EJBCAWS URL, The address to the EJBCA Web Service interface	`https://localhost:8443/ejbca/ejbcaws/ejbcaws`
`ejbcaws.softkeystore`	Tells the framework to use a soft key store to authenticate to EJBCA. This is very handy during development to avoid having to insert an administration card every time a controller is tested.	`Commented out by default`
`ejbcaws.softkeystore.password`	The password of the key store	`foo123`
`applet.height`	Setting defining the height of the applet	`500`
`applet.width`	Setting defining the width of the applet	`600`
`logo.name`	Name of the logo image placed in the src/resources/images directory	`genlogo.gif`
`logoselector.cert`	Defines which class implementing the org.hardtokenmgmt.core.ui.CertSelector interface that should be used to select the logotype and name that is related to a certificate.	`org.hardtokenmgmt.core.ui.DefaultCertSelector`
`displaytokenselector.cert`	Defines which class implementing the org.hardtokenmgmt.core.ui.DisplayTokenSelector interface that should be used to select the logotype and name that is related to a token	`org.hardtokenmgmt.core.ui.DefaultDisplayTokenSelector`

5.4 Administrator Settings

It is also possible to dynamically save persistent data with a scope of the administrator. It is accessed through the BaseControllers getAdministratorSettings() or InterfaceFactory.getAdministratorSettings() and contains the methods getProperty() and setProperty(). It is up to the controllers to define the values of the keys.

The property file is saved in the administrators home directory and have the name of `adminsettings_<Admin Certificate CN>.properties`, if no client certificate is used will the string "default" be used.

5.5 Controller Memory

The Controller memory is a non-persistent storage that can be used to pass information between the different controllers. It is built around a simple single-ton hash map.

It can be accessed through BaseController.getControllerMemory() or InterfaceFactory.getControllerMemory() and have two main methods getData(key) and putData(key, value), the value of the key is up to the implementation but a naming making the it controller unique is recommended, ex "DummyController.SomeProperty".

5.6 Local Log

The LocalLog is an interface to write to the local log file at the administrators workstation. This log should mainly be used for error and debugging and not for security purposes.

The LocalLog can be accessed by calling the different static methods of the LocalLog or using the info(), error() and debug() methods inherited from the BaseController.

The local log is written to the users home directory with the file name:
`.hardtokenmgmt<number>.log`

5.7 Other Plug-in:able Classes

It is possible to customize the behaviour of the ready made components in ToLiMa by creating own implementations of plug-in:able interfaces. Some of these interfaces are CertSelector and DisplayTokenSelector.

5.7.1 CertSelector

The CertSelector is used during the generation and viewing of a certificate. To select which logotype that should be shown and were on the card to put the certificate.

Configuring which CertSelector that should be using in the application is done in global.properties.

```
public interface CertSelector {

      /**
       *
       * @param cert that should be analysed
       * @return the image filename of the logotype.
       */
      public String getLogo(X509Certificate cert);

      /**
       *
       * @param cert that should be analysed
       * @return name related to the certificate.
       */
      public String getName(X509Certificate cert);

      /**
       * Method that should return the certificate label for a specified
       * certificate.
       *
       * A certificate label is a marking in the card profile to identify
       * the certificate.
       *
       * @param cert to retrieve the certificate label for.
       * @return the certificate label.
       */
      public String getCertLabel(X509Certificate cert);

      /**
       * Method that should return only one of the IToken.PINTYPE_ constant
       * of which pin type a certificate should be associated with.
       * @param cert to get the PIN type for
       * @return a IToken.PINTYPE_ constant
       */
      public String getAssociatedPINType(X509Certificate cert);

}
```

5.7.2 DisplayTokenSelector

The DisplayTokenSelector is used when displaying the token, which name the type of token have (for example 'project card') and it's logo.

There exists a default implementation that determines the type of token by its validity.

Configuring which DisplayTokenSelector that should be using in the application is done in global.properties.

```
public interface DisplayTokenSelector {

	/**
	 * Method that should return the name of the logo display, associated
	 * with a particular hard token data ws.
	 * @param hardTokenDataVOWs fetched from EJBCA
	 * @return a string pointing to a smart card icon image
	 */
	String getLogo(HardTokenDataWS hardTokenDataVOWs);

	/**
	 * Method that should return the name of the logo display, associated
	 * with a particular IToken
	 * @param token read from the card reader
	 * @return a string pointing to a smart card icon image
	 */
	String getLogo(IToken token);

	/**
	 * Method that should return the name displayed along with the token.
	 * This name should already have been translated inside the
	 * implementation.
	 *
	 * @param hardTokenDataVOWs fetched from EJBCA
	 * @return the name of the token type displayed.
	 */
	String getName(HardTokenDataWS hardTokenDataVOWs);

	/**
	 * Method that should return the name displayed along with the token.
	 * This name should already have been translated inside the
	 * implementation.
	 *
	 * @param token read from the card reader.
	 * @return the name of the token type displayed.
	 */
	String getName(IToken token);

}
```

6 Using Customized Source

It is possible to have your own code compiled into the main distribution but have separated in its own code tree and revision system. There exists a sample custom project in the subversion tree at sourceforge.net (https://hardtokenmgmt.svn.sourceforge.net/svnroot/hardtokenmgmt/customhardtokenmgmt) that can be checked out and used as template.

To link the framework with the custom project change the property `customproject.location` in the `hardtokenmgmt.properties` file to the base directory of the project and let it have the following structure.

/src/java	: Put all Java code here.
/src/test	: Put all JUnit test scripts here
/src/resources/images	: Put all the images here.
/src/resources/languages/custom_languagefile.properties	: Language resources that will be overridden

7 Using Eclipse for development

The framework and it's application suite ToLiMa have been developed with the IDE Eclipse and is well integrated with it. Using the Java Visual Editor to draw the Swing View classes works well with both finding the image and language resources as well as global configuration.

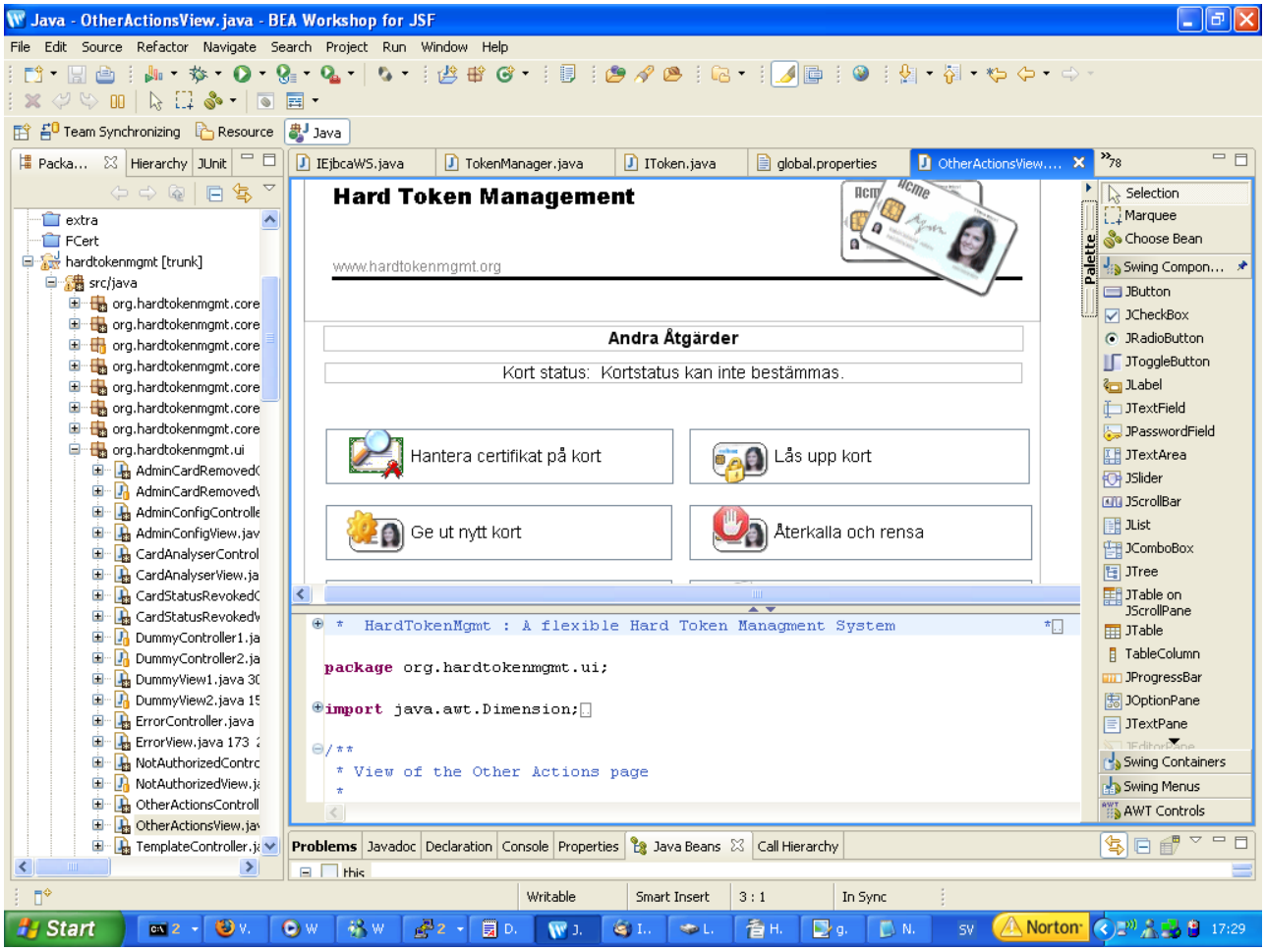


Illustration 1: Example of drawing 'Other Actions' Views in Eclipse

Here are some tips that might be helpful setting up your Eclipse IDE:

- Make sure the image, globalsettings and language resource directories are added as source directories in project 'Properties -> Build Path -> Source'. I.e the following directories should be in the source
  - src/java
  - src/test
  - src/resources/images

- src/resources/languages
- src/resources/globalsettings

- If you have a custom source code that you want compiled make sure to 'link source' the custom project in the main hardtokenmgmt project. Linking is done in the project 'Properties -> Build Path -> Source' page. The following directories must be linked as source in your custom project:
  - <customproject>/src/resources/images -> hardtokenmgmt/customimages
  - <customproject>/src/resources/languages -> hardtokenmgmt/customlanguages
  - <customproject>/src/java -> hardtokenmgmt/customjava
  - <customproject>/src/test -> hardtokenmgmt/test

- If you test the application using 'Run as Applet' you can make short cuts to which controller you want to start with by editing the setting `controller.main` in global.properties.

- Use a soft key store to authenticate to EJBCA if you don't want to insert the administration card every time, but remember to change the `token.alwaysrequireadmintoken` setting as well.

8 Existing Components

The ToLiMa application suite contains quite a few ready made controllers (with views) that can be used for other projects, either as they are or to be used as examples how to develop similar. The ToLiMa components is in the org.hardtokenmgmt.ui package and it's subpackages. To get details about the classes it's best to read the Java documentation at http://www.hardtokenmgmt.org/api/index.html .

9 Testing the Framework

9.1 Junit tests

There exists a few (not many) junit tests that tests some aspects of the Hard Token Management Framework. They are run by the command 'ant test:run' in the hardtokenmgmt home directory.

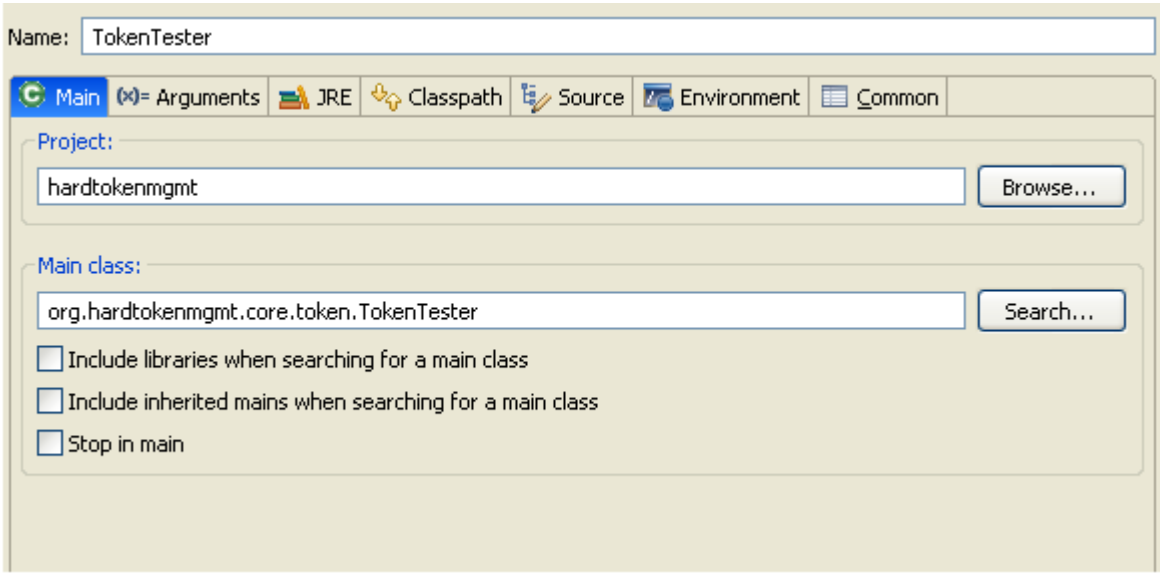
After the test run is a protocol generated in html in the directory tmp\bin\junit\reports\html.

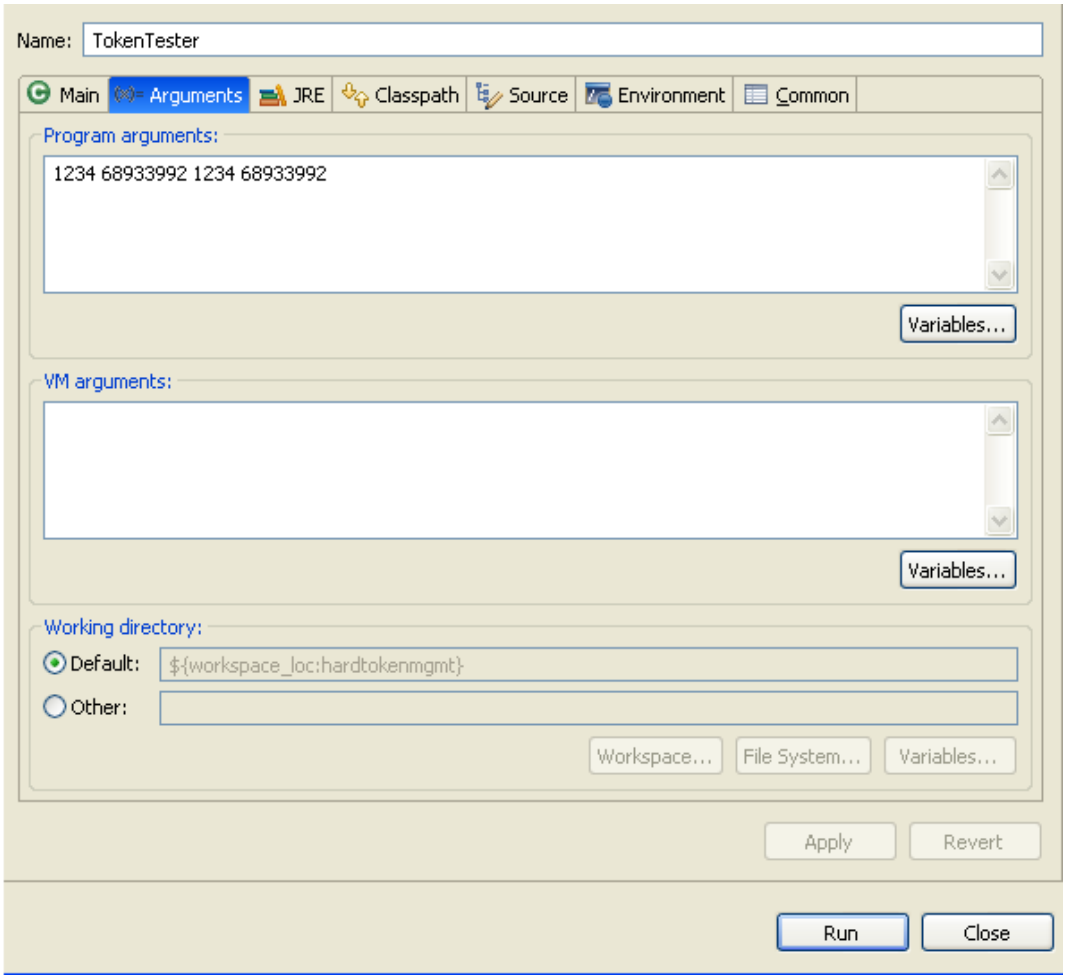
9.2 Testing Token and PKCS11 implementations

There also exists another test script used to test compatibility the tokens or more specific their PKCS11 implementation. This script is usually run as a regular application from Eclipse.To add this script to Eclipse do the following. Goto to 'Run ...', mark 'Java Application' and click on 'New'. Then configure the application according to the following two images, for argument it is <basicpin> <basicpuk> <signaturepin> <signaturepuk>.

After you have save the configuration just launch the application and it will prompt for the inserted card. Remember that you first have to configure the PKCS11 and add the Token implementation to the list of supported ones in the global.properties file.

It is not always necessary for a Token to go through all the tests, some PKCS11 handles PIN/PUK status differently and the Setec cards fail some of those tests. But see the supported tokens section on the homepage for more details.




Name: TokenTester
Main
Arguments
JRE
Classpath
Source
Environment
Common
Program arguments:
1234 68933992 1234 68933992
Variables...
VM arguments:
Variables...
Working directory:
Default:
${workspace_loc:hardtokenmgmt}
Other:
Workspace...
File System...
Variables...
Apply
Revert
Run
Close

10 More Information

More information about the Hard Token Management Framework and ToLiMa can be found at:

Main Website, http://www.hardtokenmgmt.org

EJBCA.org, http://www.ejbca.org