eID Applet Developer's Guide

Integrating the eID Applet within your web applications.
Version 1.2.0.Beta1

Frank Cornelis

18 February 2014

Copyright © 2008-2013 Fedict

Abstract

This developer's guide serves as an entry point for integrating the eID Applet in your web applications. The target audience is web developers and web application architects.

1. Introduction

The eID Applet is a browser component that exposes the functionality of an eID card to your web applications. In *Figure 1, "eID Applet Screenshot"* you find a screen shot of the eID Applet.

Please insert your eID card...

Details >>

Figure 1. eID Applet Screenshot

The main features of the eID Applet are:

- Easy to integrate within an existing web application.

- Security and privacy of the citizen is protected.

- Interactive eID card handling.

- Support of CCID secure pinpad readers.

The eID Applet uses Java applet technology. This minimized the client web browser requirements.

eID Applet Source Code

The eID Applet source code is available at *eID Applet Google Code* [http://code.google.com/p/eid-applet/] .

eID Applet Source Code Branching

While some might feel very tempted to branch the eID Applet, given the open source nature of this project, (for example for branding reasons) one needs to take all aspects of branching into consideration. If you branch you will lose all official bug fixes unless you back-port them manually which can be quite time consuming.

eID Applet License

The eID Applet has been released under the GNU LGPL 3.0 open source software license. This implies that, in case you alter the eID Applet, you have to make the eID Applet source code available yourself.

eID Applet Support

Best-effort support on the eID Applet is provided via the *eID Applet Google Group* [http://groups.google.com/group/eid-applet] mailing list. Feel free to join in.

1.1. Mac OS X

Because Apple only supports the Java 6 runtime on their Mac OS X operating systems since Snow Leopard, the identification functionality will not work for Mac OS X 10.4 and 10.5.

The strategy is to no longer support operating systems, but to support a specific Java platform. For the eID Applet this is the Java 6 platform. We can only give advice on how to configure Java 6 on your operatings system.

1.2. Linux

1.2.1. Fedora 9, 10, 11, 12

The Fedora operating system comes by default with the IcedTea JRE which is based on the OpenJDK. If the Firefox browser uses this JRE the eID Applet still has some difficulties to run. Please download the official Sun Java 6 JRE and enable it in the Firefox browser. The Firefox plugins can be configured via symbolic links under: `/usr/lib/mozilla/plugins` . Remove the IcedTea JRE link via: `rm /usr/lib/mozilla/plugins/libjavaplugin.so` . Afterwards add a symbolic link to the Sun JRE plugin, which can be found under: `$JAVA_HOME/jre/plugin/i386/ns7/libjavaplugin_oji.so` . Check the installed plugins in Firefox by navigating to: `about:plugins` .

1.2.2. Ubuntu 9.04, 9.10

Under Linux Ubuntu you can install the Sun JRE 1.6 via the following command: `sudo apt-get install sun-java6-jdk sun-java6-plugin`

1.2.3. Linux: Firefox 3.6 and Chrome

The web browsers Firefox 3.6 and Google Chrome use the next generation Java plugin (libnpjp2.so). So for Firefox you can configure the Java plugin as follows: `sudo ln -s $JAVA_HOME/jre/lib/i386/libnpjp2.so /usr/lib/mozilla/plugins/`

1.2.4. SELinux

Under Fedora Linux you might trigger an SELinux error when the browser tries to run the JRE plugin. The following command prevents the SELinux error: `sudo chcon -t $JAVA_HOME/jre/lib/i386/client/libjvm.so`

2. eID Applet

The eID Applet should be used within a web page as shown in the following example:

```
<script src="https://www.java.com/js/deployJava.js"></script>
<script>
    var attributes = {
        code :'be.fedict.eid.applet.Applet.class',
        archive :'eid-applet-package.jar',
```

```
        width :400,
        height :300
    };
    var parameters = {
        TargetPage :'identity-result.jsp',
        AppletService : 'applet-service',
        BackgroundColor : '#ffffff'
    };
    var version = '1.6';
    deployJava.runApplet(attributes, parameters, version);
</script>
```

Notice that we are using the Deployment Toolkit to load the eID Applet. This avoids browser compatibility issues and features an automatic installation of the required Java browser plugin.

The web application in which the eID Applet is embedded should use SSL for securing the communication between the web browser and the web application server. The eID Applet will not proceed when it detects a non SSL browser session.

The eID Applet will also not proceed when it detects that it has insufficient privileges to do so. This implies that the eID Applet JAR has to be signed and trusted by the citizen. The eID Applet that ships with an officially released eID Applet SDK has been signed by Fedict. In case of a security breach with the eID Applet, Fedict can revoke the corresponding code signing certificate to guarantee maximal safety of the citizen.

Google Chrome

For Google Chrome to be able to load the eID Applet using the Deployment Toolkit Javascript you should host the deployJava.js Javascript locally.

eID Applet JavaScript

The eID Applet SDK contains an `eid-applet-js` artifact that offers a default JavaScript to load the eID Applet.

The available eID Applet parameters are summarized in *Table 1, "eID Applet Parameters"* .

Table 1. eID Applet Parameters

Parameter	Required	Description
`TargetPage`	required	Indicates the page to which the eID Applet navigates after performing the requested eID operation. For example: `result.jsp`

Parameter	Required	Description
AppletService	required	Points to the eID Applet Service server-side component that will handle the communication between the eID Applet and the (servlet) web application container. For example: `applet-service`
CancelPage	optional	Indicates the page to which the eID Applet navigates after the user cancelled the eID operation. For example: `cancel.jsp`
BackgroundColor	optional	The background color that is used by the eID Applet user interface. The default background color is white. For example: `#ffffff`
ForegroundColor	optional	The foreground color that is used by the eID Applet user interface. The default foreground color is black. For example: `#000000`
Language	optional	The language that is used by the eID Applet user interface for internationalization of the status messages. If it is not provided, the eID Applet defaults to the JRE runtime locale settings. For example: `nl`
MessageCallback	optional	Via this parameter a web developer can configure a Javascript callback. This callback function will be invoked everytime the eID Applet displays an info message. The function signature looks like: `function messageCallback(status, message)`
MessageCallbackEx	optional	Via this parameter a web developer can configure a Javascript callback. This callback function will be invoked every time the eID Applet displays an info message. The function signature looks like: `function messageCallback(status, messageId, message)`
UserAgent	optional	Via this parameter you can let the eID Applet to use the given User-Agent HTTP header. Some servers might require that the User-Agent reported by the eID Applet is the same as the one reported by the web browser to be able to use the same HTTP session context. Example value: `navigator.userAgent`
HideDetailsButton	optional	When this parameter is set to `true` the eID Applet will hide the details button. To

Parameter	Required	Description
		compensate for this, the eID Applet will use the Java Console to output detailed logging. Please note that the Java Console is not enabled per default on every platform. The Java Console can be enabled via the `jcontrol` JVM tool.
`NoChunkedTransferEncoding`	optional	When this parameter is set to `true` the eID Applet will not used chunked transfer-encoding when communicating with the eID Applet Service component.

Third-party cookies

Recent browsers might disable third-party cookies by default. As the Java web browser plugin is a third-party component, it's possible that the session cookie is no longer communicated to the Java runtime. A work-around for JSF is given by appending the following to the `AppletService` parameter: `;jsessionid=#{facesContext.externalContext.getSession(false).id}`

Javascript

The eID Applet cannot be accessed from Javascript for cross-site scripting security reasons.

Javascript TargetPage

The `TargetPage` applet parameter can also be used to execute a Javascript when the eID operation is finished. Example: `TargetPage: 'javascript:alert("Hello World");'`

3. eID Applet Service

The eID Applet requires a server-side service component to communicate the identity or authentication data from the web browser to the server using a secure channel. We call this component the eID Applet Service. The eID Applet SDK comes with eID Applet Service servlet components to ease integration of the eID Applet within servlet container Java EE based web applications. The eID Applet Service components require at least a servlet version 2.4 container and a JRE version 1.6. The eID Applet and eID Applet Service architecture has been depicted in *Figure 2, "eID Applet Architecture"* .
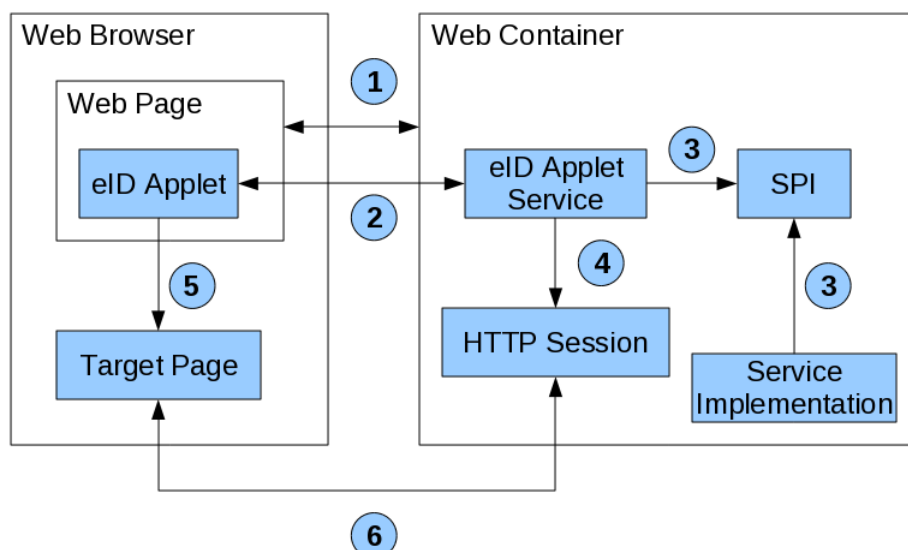
Figure 2. eID Applet Architecture

During the first step (1) the web browser loads the web page containing a reference to the eID Applet. The web browser continues by loading the eID Applet via the JRE web browser plugin. After the eID Applet has been loaded, it initiates a protocol run (2) with the server-side eID Applet Service. For some eID operations the web developer is required to configure service provider components. These service provider components are invoked (3) by the eID Applet Service during a protocol run. At the end of a protocol run (4) the eID Applet Service pushes some attributes into the HTTP session context of the web application container. Finally (5) the eID Applet makes the web browser to navigate to the target page. The target page can now access the eID identity items (6) made available by the eID Applet service.

eID Applet Service implementations

For the moment we only fully support Java EE servlet containers out of the box. At the same time this serves as the reference implementation. For other web application environments like the ASP.NET web application environment and the PHP environment we strongly advise to use the eID Identity Provider and eID Digital Signature Service products to integrate eID within ASP.NET and PHP based web applications.

eID Applet Service HTTP session

When using web frameworks like JBoss Seam you might stumble on conversation preservation issues because of the redirect executed by the eID Applet at the end of the performed eID operation. When using a conversation scoped JBoss Seam managed bean (like the built-in redirect component), you can preserve the conversation across the eID Applet screen flow by adding the following HTTP

> parameter to the TargetPage applet parameter: `TargetPage :'your-target-page.seam?conversationId=#{conversation.id}',`

3.1. eID Identification

By default the eID Applet Service will operate the eID Applet to make it perform an eID identification. This is also known as data capture. Via this eID operation your web application is capable of reading out the identity data (i.e. name, first name, date of birth, address, ...) of the user his eID card.

The eID Applet Service Servlet can be configured via your `web.xml` web deployment descriptor as shown in the following example:

```xml
<servlet>
    <servlet-name>AppletServiceServlet</servlet-name>
    <servlet-class>
        be.fedict.eid.applet.service.AppletServiceServlet
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>AppletServiceServlet</servlet-name>
    <url-pattern>/applet-service</url-pattern>
</servlet-mapping>
```

The eID Applet Service, which can be found in the `eid-applet-service-x.x.x.jar` artifact, has some 3rd party dependencies. These artifacts are located under the `lib/` directory inside the eID Applet SDK package. Depending on your Java EE runtime environment you should place these JAR files under the `META-INF/lib` directory of your web application.

In case that you use Maven as build-system, you can configure the following Maven repository:

```xml
<repository>
    <id>e-contract</id>
    <url>https://www.e-contract.be/maven2/</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
</repository>
```

The eID Applet project comes with a BOM that can be included in your POM file as follows:

```xml
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>be.fedict.eid-applet</groupId>
            <artifactId>eid-applet-bom</artifactId>
            <version>1.2.0.Beta1</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Now you can simply add the eID Applet Service dependency as follows:

```xml
<dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-service</artifactId>
</dependency>
```

Depending on the Servlet container that you use, you might need to exclude certain transitive dependencies.

eID Applet Service availability

One can always check for eID Applet Service availability by manually browsing to the location of the eID Applet Service servlet as configured in your `web.xml` Java EE web deployment descriptor.

After a successful identification took place, the `AppletServiceServlet` eID Applet Service will push at least the `eid.identity` attribute, which holds the parsed identity fields, to the servlet container session. The `eid.identity` session attribute is of Java type `be.fedict.eid.applet.service.Identity` . More information on the exposed attributes can be found in the Javadoc API documentation of the eID Applet Service artifact. The Javadoc documentation is part of the eID Applet SDK package.

eID Session Attributes

To ease integration of the eID Applet Service in web frameworks like JBoss Seam we have provided a top-level `eid` session attribute and getters on all exposed session attribute types. The top-level `eid` session attribute is of Java type `be.fedict.eid.applet.service.EIdData` . This means that the identity is available via both `eid.identity` session attribute and invocation of the

> `getIdentity()` method on the `eid` session attribute. This way we cover as much Java web frameworks as possible.

CDI Support

The eID Applet SDK contains an `eid-applet-service-cdi` artifact that offers initial support for CDI containers. The artifact does not require JBoss Seam 3.1 at run-time. When using Maven, include the CDI artifact via:

```xml
<dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-service-cdi</artifactId>
</dependency>
```

3.1.1. eID Address

During an eID identification operation the address on the eID card can be retrieved by setting the following `init-param` on the `AppletServiceServlet` :

```xml
<init-param>
    <param-name>IncludeAddress</param-name>
    <param-value>true</param-value>
</init-param>
```

After a successful eID identification, the eID address will be available via the `eid.address` session attribute within the servlet container session context. The `eid.address` session attribute is of Java type `be.fedict.eid.applet.service.Address` . The available fields and functions of the `Address` class are described within the Javadoc API documentation which is part of the eID Applet SDK package.

3.1.2. eID Photo

During an eID identification operation the citizen's photo on the eID card can be retrieved by setting the following `init-param` on the `AppletServiceServlet` :

```xml
<init-param>
    <param-name>IncludePhoto</param-name>
    <param-value>true</param-value>
</init-param>
```

After a successful eID identification, the photo will be available via the `eid.photo` session attribute within the servlet container session context. The eID photo is of Java type `byte[]` and uses a JPEG image format.

We provide a `PhotoServlet` to ease visualization of the eID photo within your web application. Configure the `PhotoServlet` as follows:

```xml
<servlet>
    <servlet-name>PhotoServlet</servlet-name>
    <servlet-class>be.fedict.eid.applet.service.PhotoServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>PhotoServlet</servlet-name>
    <url-pattern>/photo.jpg</url-pattern>
</servlet-mapping>
```

After a successful eID identification you can display the eID photo by putting next HTML tag in your web page:

```html
<img src="photo.jpg" />
```

3.1.3. eID Certificates

If you need to have explicit access to the eID citizen certificates, you can instruct the eID Applet to extract the certificates via the following eID Applet Service servlet configuration:

```xml
<init-param>
    <param-name>IncludeCertificates</param-name>
    <param-value>true</param-value>
</init-param>
```

After a successful eID identification, the certificates will be available as session attributes of Java type `java.security.cert.X509Certificate`. The authentication certificate will be available as `eid.certs.authn` session attribute. The non-repudiation (i.e. signature) certificate will be available as `eid.certs.sign` session attribute. The intermediate Citizen CA certificate will be available as `eid.certs.ca` session attribute. The Root CA certificate will be available as `eid.certs.root` session attribute.

3.1.4. Output to PDF

The eID Applet SDK comes with a servlet component that allows you to output the eID identity data to PDF. This can be useful if you want to print the eID identity data from within your web application pages. The PDF servlet can be configured as follows:

```
<servlet>
    <servlet-name>PdfServlet</servlet-name>
    <servlet-class>be.fedict.eid.applet.service.PdfServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>PdfServlet</servlet-name>
    <url-pattern>/identity.pdf</url-pattern>
</servlet-mapping>
```

After a successful eID identification, the PDF is available via:

```
<a href="identity.pdf" target="_blank">View as PDF</a>
```

3.1.5. Output to KML (Google Earth)

The eID Applet Service also comes with a servlet for exporting the eID identity data to (zipped) KML. This can be useful if you want to visualize the eID identity data on a map.

Please note that the eID applet does not provide geocoding services, i.e., addresses are not automatically converted to geographic coordinates. However, the KMZ file can be opened in applications that do provide geocoding services, like Google Earth. *Figure 3, "eID Identity in Google Earth"* shows a screenshot of an eID identity visualized via Google Earth.

Figure 3. eID Identity in Google Earth

The servlet is configured as follows:

```xml
<servlet>
    <servlet-name>KmlServlet</servlet-name>
    <servlet-class>be.fedict.eid.applet.service.KmlServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>KmlServlet</servlet-name>
    <url-pattern>/identity.kmz</url-pattern>
</servlet-mapping>
```
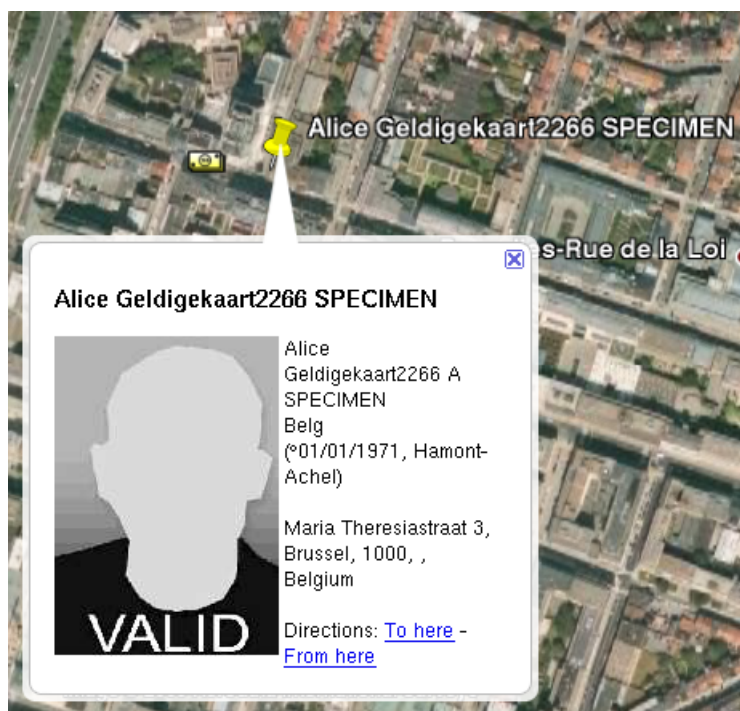
After a successful eID identification, the Google Earth KMZ file is available via:

```html
<a href="identity.kmz" target="_blank">View in Google Earth</a>
```

3.1.6. JSON

The eID Applet SDK comes with a servlet to support eID identity data retrieval inside your web application via JSON. The JSON servlet is configured as follows:

```xml
<servlet>
    <servlet-name>JSONServlet</servlet-name>
    <servlet-class>be.fedict.eid.applet.service.JSONServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>JSONServlet</servlet-name>
    <url-pattern>/identity.js</url-pattern>
</servlet-mapping>
```

The retrieved JSON data object has a structure similar to the following example:

```
{
 identity: {
  name: "SPECIMEN",
  firstName: "Alice Geldigekaart",
  dateOfBirth: "Fri Jan 01 00:00:00 CET 1971",
  gender: "FEMALE"
 },
 address: {
  streetAndNumber: "Meirplaats 1 bus 1",
  municipality: "Antwerpen",
  zip: "2000"
 }
```

```
}
```

3.1.7. Identity Data Integrity

During an eID identification operation the eID Applet Service can perform integrity verification on the retrieved eID identity data. This feature prevents malicious parties to alter critical identity data.

To enable this functionality as part of an eID identification operation, you need to implement the `IdentityIntegrityService` interface. This service provider interface (SPI) can be found in the `eid-applet-service-spi` artifact. The corresponding service component (EJB3) session bean should be registered somewhere in JNDI. The JNDI location of this service component needs to be communicated to the eID Applet Service via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>IdentityIntegrityService</param-name>
    <param-value>
        your/location/in/jndi/IdentityIntegrityServiceBean
    </param-value>
</init-param>
```

The Javadoc documentation of the `IdentityIntegrityService` SPI is part of the eID Applet SDK package.

Java EE Application Classpath

In an EJB Java EE application the `eid-applet-service-spi` artifact should be moved from your web application `WEB-INF/lib` WAR artifact to the EAR scoped classpath. Depending on your used Java EE application server is should be registered in `application.xml` as a Java module or moved to the `lib/` directory of your EAR to avoid classpath issues in your application server.

Java EE 6 Web Profile support

To support the Java EE 6 Web Profile we have foreseen the usage of two types of service component lookups.

The first one is JNDI based. This type of service lookup allows you to utilize EJB3 session beans as service provider interface implementation. Since Java EE 6 the JNDI naming of EJB3 session beans has been standardized. Refering to your component can now be done via, for example, `java:module/AuthenticationRequestServiceBean`

> The second type is via simple Java class name references. This type of service lookup is meant for lightweight servlet container environment. The implementing class needs a default constructor in order for the eID Applet Service to be able to instantiate it.
>
> For example the `SignatureService` interface implementing component can be referred to via both `SignatureService init-param` and via `SignatureServiceClass init-param`. The `SignatureService init-param` will trigger a JNDI lookup of the signature service. The `SignatureServiceClass init-param` will trigger a class instantiation using the default constructor of the given class.

The identity integrity service prevents malicious parties from altering the identity data. However, this does not prevent malicous parties to replace the identity data with that of another citizen. To prevent replacement of identity data, one can use a so called authenticated eID identification.

If the eID identification is preceded with an eID authentication then the eID Applet Service is able to link the authenticated national registry number with the one found in the eID identity file during identity integrity verification. This makes for a bullet-proof eID identification that cannot be forged.

For some applications that need eID identification of citizen B after eID authentication of citizen A, you might want to disable this feature. Do so via:

```
<init-param>
    <param-name>SkipNationalNumberCheck</param-name>
    <param-value>true</param-value>
</init-param>
```

3.1.8. Privacy Service

The application can define an identity data usage description at runtime by means of a privacy service component. To enable this functionality as part of an eID identification operation, you need to implement the `PrivacyService` interface. This service provider interface (SPI) can be found in the `eid-applet-service-spi` artifact. The corresponding service component (EJB3) session bean should be registered somewhere in JNDI. The JNDI location of this service component needs to be communicated to the eID Applet Service via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>PrivacyService</param-name>
    <param-value>your/location/in/jndi/PrivacyServiceBean</param-value>
</init-param>
```

The Javadoc documentation of the `PrivacyService` SPI is part of the eID Applet SDK package.

3.2. eID Authentication

The eID Applet can be used to authenticate an end user via the eID card. eID based entity authentication is much safer than a simple password based authentication scheme since the eID card makes a two-factor authentication possible.

eID Applet Authentication Configuration

There are many different eID Applet configurations possible for eID Authentication. The optimal configuration highly depends on your web application requirements. In case of doubt contact us at the *eID Applet Google Group* [http://groups.google.com/group/eid-applet] mailing list for additional advice.

To perform an eID authentication, you need to implement the `AuthenticationService` interface. This interface can be found as part of the `eid-applet-service-spi` artifact. This service component (EJB3) session bean should be registered somewhere in JNDI. The JNDI location of this service component needs to be set via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>AuthenticationService</param-name>
    <param-value>your/location/in/jndi/AuthenticationServiceBean</param-value>
</init-param>
```

The Javadoc documentation of the `AuthenticationService` SPI is part of the eID Applet SDK package.

After a successful authentication the `eid.identifier` session attribute will contain a unique identifier (the national registration number) for the user. The `eid.identifier` session attribute is of Java type `java.lang.String`.

Usage of the national registration number

To respect the citizen's privacy, the national registration number should not be abuse for linking identity data. Profiling based on eID data linking is forbidden by law. Hence one cannot save the national registration number within a database as primary key without proper authorization.

> **i** **Mac OS X**
>
> You need a version of Mac OS X that supports Java 6.

> **i** **eID Middleware**
>
> The eID Applet is using not using the eID Middleware to perform eID operations. The eID Applet is directly accessing the eID card via the PC/SC interface. This requires a Java 6 client runtime.

By default the eID Applet will sign a sequence similar to `(salt, challenge)` using the authentication private key of the citizen's eID card. The challenge is send over SSL by the eID Applet Service. The salt value is produced by the eID Applet itself. The salt value prevents that the eID Applet is forced into signing a given server-side value. To prevent a certain type of man-in-the-middle attack we can make the eID Applet to sign a sequence similar to `(salt, hostname, challenge)`. This feature can be enabled by setting the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>Hostname</param-name>
    <param-value>www.PutYourSiteHostnameHere.be</param-value>
</init-param>
```

> **!** **Hostname verification**
>
> It is strongly advised to enable this hostname verification feature to reduce security vulnerability.

To prevent DNS attacks one can even make the eID Applet sign the IP address of the server. This feature can be enabled by setting the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>InetAddress</param-name>
    <param-value>1.2.3.4</param-value>
</init-param>
```

If you enable both `Hostname` and `InetAddress` features at the same time, the eID Applet will be signing a sequence similar to `(salt, hostname, IP address, challenge)`. The hostname and IP address are the same as seen by the web browser.

3.2.1. Non-reversible Citizen Identifier

After a successful eID authentication took place, the `eid.identifier` session attribute will contain the national registry number. In some cases the national registry number cannot be used as is for unique user identifier. The eID Applet Service features Non-Reversible Citizen Identifiers (NRCID) to transform the national registry number into an application domain specific identifier. The NRCID is based on the HMAC-SHA1 of the National Registry Number, optionally appended with an application identifier and/or organization identifier. This feature can be enabled by setting the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>NRCIDSecret</param-name>
    <param-value>place-your-application-secret-here</param-value>
</init-param>
```

The secret should be hexadecimal encoded and at least 128 bits (16 bytes) long. Thus the hexadecimal encoded secret should be at least 32 characters long.

The optional application identifier and organization identifier can be specified via the `NRCIDAppId` and `NRCIDOrgId` init parameters.

3.2.2. Secure Channel Binding

Tunneled entity authentication protocols like the one implemented by the eID Applet are subject to man-in-the-middle attacks without proper secure channel binding put in place. Cryptographic end-point channel binding has been implemented by means of digesting the TLS server certificate as part of the authentication signature. This option can be activated via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>ChannelBindingServerCertificate</param-name>
    <param-value>/path/to/your/server/certificate.der</param-value>
</init-param>
```

The server certificate should be in DER encoded format or in PEM format.

> ### Server Certificate Channel Binding
>
> It is strongly advised to activate server certificate cryptographic channel binding to have equivalent security properties compared to mutual TLS entity authentication.

The server certificate used to verify the secure channel binding can also be provided at runtime by implementing an SPI component. This option can be activated via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>ChannelBindingService</param-name>
    <param-value>your/location/in/jndi/ChannelBindingServiceBean</param-value>
</init-param>
```

The Javadoc documentation of the `ChannelBindingService` SPI is part of the eID Applet SDK package.

Besides server certificate channel binding the eID Applet also supports unique channel binding using the TLS session identifier. This option can be activated via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>SessionIdChannelBinding</param-name>
    <param-value>true</param-value>
</init-param>
```

This will make the authentication signature to also digest the TLS session identifier.

Channel Binding

Secure channel binding based on unique channel binding using the TLS session identifier alone is not enough! Always use at least server certificate cryptographic channel binding. You can combine this with (unsecure) unique channel binding using the TLS session identifier if appropriate.

3.2.3. Explicit PIN entry

The eID card offers caching of the PIN authorization when creating an authentication signature. Some applications might require a PIN entry upon each authentication request. This can be achieved by performing an eID card logoff right before the creation of the authentication signature. Activate this feature via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>PreLogoff</param-name>
    <param-value>true</param-value>
</init-param>
```

3.2.4. Authenticated Identification

It is possible to combine an eID authentication operation with an eID identification operation. Activate the eID identification as part of the eID authentication via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>IncludeIdentity</param-name>
    <param-value>true</param-value>
</init-param>
```

As was the case for eID identification this will make the eID identity available as attributes within the HTTP servlet session context.

Also include the eID address via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>IncludeAddress</param-name>
    <param-value>true</param-value>
</init-param>
```

Also include the eID photo via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>IncludePhoto</param-name>
    <param-value>true</param-value>
</init-param>
```

The identity integrity service can also be activated when combining eID authentication with eID identification by configuring an `IdentityIntegrityService` SPI implementation.

3.2.5. eID Certificates

If you need to have explicit access to the eID citizen certificates, you can instruct the eID Applet to extract the certificates via the following eID Applet Service servlet configuration:

```
<init-param>
    <param-name>IncludeCertificates</param-name>
    <param-value>true</param-value>
</init-param>
```

After a successful eID authentication, the certificates will be available as session attributes of Java type `java.security.cert.X509Certificate` . The authentication certificate will be available as `eid.certs.authn` session attribute. The non-repudiation (i.e. signature) certificate will be available as `eid.certs.sign` session attribute. The intermediate Citizen CA certificate will be available as `eid.certs.ca` session attribute. The Root CA certificate will be available as `eid.certs.root` session attribute.

3.2.6. eID secure PIN pad card reader

FedICT developed a new secure PIN pad card reader that features eID transaction confirmation on the hardware device itself. Basically this secure card reader intercepts signature (both authentication and non-repudiation) computations that use a specific hash algorithm. The OID for this hash algorithm is `2.16.56.1.2.1.3.1` . For this specific hash algorithm the digest value is considered as being plain text that can be visualized by the hardware device. This allows for hardware based transaction confirmation. This feature gives applications additional means of ensuring the user consent in the context of a certain application level transaction.

The eID Applet has explicit support for such transaction messages as part of the authentication process. To enable this feature you have to implement the `SecureCardReaderService` SPI. The JNDI location of this service component needs to be set via the following `init-param` on the `AppletServiceServlet` :

```
<init-param>
    <param-name>SecureCardReaderService</param-name>
    <param-value>
        your/location/in/jndi/SecureCardReaderServiceBean
    </param-value>
</init-param>
```

Smart card readers that do not offer support for this plain text hash algorithm can of course not visualize the transaction message.

3.3. eID Signatures

The eID Applet can also be used to create digital signatures using the non-repudiation eID certificate. The supported signature algorithms are `SHA1-RSA-PKCS1` , `SHA224-RSA-PKCS1` , `SHA256-RSA-PKCS1` , `SHA384-RSA-PKCS1` , `SHA512-RSA-PKCS1` , `RIPEMD128-RSA-PKCS1` , `RIPEMD160-RSA-PKCS1` , `RIPEMD256-RSA-PKCS1` , `SHA1-RSA/PSS-PKCS1` , and `SHA256-RSA/PSS-PKCS1` .

> ⚠ **Legally Binding eID Digital Signatures**
>
> Please be aware that the eID digital signatures are legally binding by law. Don't make the citizen sign digital documents unless it is absolutely necessary from a legal point of view for the correct functioning of your business work flow.

To use this functionality you need to implement the `SignatureService` interface. This interface can be found in the `eid-applet-service-spi` artifact. This service component (EJB3) session bean should be registered somewhere in JNDI. The JNDI location of this service component needs to be set via the following `init-param` on the `AppletServiceServlet` :

```xml
<init-param>
    <param-name>SignatureService</param-name>
    <param-value>your/location/in/jndi/SignatureServiceBean</param-value>
</init-param>
```

The Javadoc documentation of the `SignatureService` SPI is part of the eID Applet SDK package.

During pre-sign phase you can receive the non-repudiation certificate chain via:

```xml
<init-param>
    <param-name>IncludeCertificates</param-name>
    <param-value>true</param-value>
</init-param>
```

You can even receive the signing identity by means of the `IncludeIdentity` , `IncludeAddress` and `IncludePhoto` init-params.

The eID Applet Service can be configured to perform two basic types of digital signatures:

- The digest value to be signed originates solely from the `SignatureService` implementing service component.

- The eID Applet first sends over a set of digest values calculated from local files. These files are selected by the citizen via an eID Applet file user interface. Out of this set of digest values the `SignatureService` implementing service component then calculates a super digest value. This digest value is signed using the eID Applet.

  The supported file digest algorithms are `SHA-1` , `SHA-256` , `SHA-384` , and `SHA-512` .

  This type of digital signature operation can be used to construct for example XML Signatures, XAdES Signatures or PDF Signatures.

The type of digital signature created by the eID Applet is completely determined by the implementation of the `SignatureService` SPI. We provide several base implementation of the `SignatureService` SPI as part of the `eid-applet-service-signer` artifact. The most important signature service implementations provided by the eID Applet SDK are:

- ODF 1.2 signatures (supported by OpenOffice.org 3.1/3.2)

- Office OpenXML (supported by Microsoft Office 2007/2010)

- CMS signatures (PKCS#7)

Besides different signature service implementations we also provide a XAdES-X-L v1.4.2 implementation as an XML signature service facet.

> ### e-Signatures Service Directive
>
> The expert group on the e-Signatures Service Directive has proposed XAdES as standard signature format.

> ### eID Digital Signature Service
>
> Instead of directly using the eID Applet to create digital signatures, one can also use the eID Digital Signature Service SOA product developed by FedICT. The eID DSS product is available at the *eID DSS* [http://code.google.com/p/eid-dss/] site.

> ### PKI Validation
>
> The eID Applet Service does not perform any PKI validation. So the signature service component, authentication service component and the identity integrity component need to implement PKI validation of the citizen certificates itself. PKI validation is out of scope of the provided eID Applet Service.
>
> A PKI validation module tailored for the Belgian eID PKI is available at the *jTrust Google Code* [http://code.google.com/p/jtrust/] site.
>
> Besides the jTrust Java library we also offer an eID Trust Service SOA product to perform eID PKI validations via an XKMS2 based web service. More information on the eID Trust Service product is available at the *eID Trust Service Google Code* [http://code.google.com/p/eid-trust-service/] site.

3.4. eID Administration

The eID Applet allows for some administrative eID tasks like changing the PIN and unblocking the PIN. This feature has been implemented to break the hard dependency on the eID Middleware.

The eID PIN change administrative task can be executed by setting the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>ChangePin</param-name>
    <param-value>true</param-value>
</init-param>
```

The eID unblock PIN administrative task can be executed by setting the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>UnblockPin</param-name>
    <param-value>true</param-value>
</init-param>
```

3.5. Generic eID Applet Service settings

The settings listed in this section apply to eID identification operations, eID authentication operations, eID signature operations, and eID administration operations.

3.5.1. Secure Client Environment

The eID Applet offers functionality to check whether the client environment is secure enough given the application requirements. In case the eID Applet Service detects an insecure client environment the eID Applet can:

- show an error message and abort the requested eID operation.

- show a warning message and ask the citizen whether he/she wants to continue or not.

To activate this functionality you need to implement the `SecureClientEnviromentService` interface. This interface can be found in the `eid-applet-service-spi` artifact. This service component (EJB3) session bean should be registered somewhere in JNDI. The JNDI location of this service component needs to be set via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>SecureClientEnvironmentService</param-name>
    <param-value>
        your/location/in/jndi/SecureClientEnvironmentServiceBean
    </param-value>
</init-param>
```

The Javadoc documentation of the `SecureClientEnvironmentService` SPI is part of the eID Applet SDK package.

Additional client environment information can be pushed to the eID Applet Service by adding the following eID Applet parameters within your web page eID Applet configuration:

```
NavigatorUserAgent : navigator.userAgent,
NavigatorAppName : navigator.appName,
NavigatorAppVersion : navigator.appVersion
```

3.5.2. eID Card Removal

The eID Applet can ask the citizen for eID card removal after performing the selected eID operation. This option can be used to limit the window of opportunity for malware to abuse the eID card.

The eID card removal can be activated by setting the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>RemoveCard</param-name>
    <param-value>true</param-value>
</init-param>
```

3.5.3. eID Card Logoff

After an eID authentication, eID signature, or eID administration task (i.e. PIN change) the eID card will re-use the PIN authorization for future eID authentication operations. This feature was originally implemented on the eID JavaCard Applet (which is located inside the eID chip) to allow for mutual authenticated SSL without the need to re-enter the PIN on each SSL session renewal. Although this makes sense in the context of SSL, it actually makes for a serious eID security weakness: SSO should be handled at the IdP level, not at the card level. Only an IdP can have notion of trust domains between different web applications. Luckily the eID card foresees in an eID card logoff. This eID logoff feature can be enabled during both eID authentication or eID signature operations by setting the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>Logoff</param-name>
    <param-value>true</param-value>
</init-param>
```

> ⚠ **Enable eID card logoff**
>
> It is strongly advised to enable the eID card logoff feature to prevent abuse of the authentication functionality of the eID card.

3.5.4. Auditing

To comply with certain regulations one might need to have an audit trace of the activities performed on the eID Applet Service by clients. The eID Applet Service offers auditing support by means of the SPI design pattern.

To activate the audit functionality you need to implement the `AuditService` interface. This interface can be found in the `eid-applet-service-spi` artifact. This service component (EJB3) session bean should be registered somewhere in JNDI. The JNDI location of this service component needs to be set via the following `init-param` on the `AppletServiceServlet`:

```
<init-param>
    <param-name>AuditService</param-name>
    <param-value>your/location/in/jndi/AuditServiceBean</param-value>
</init-param>
```

The Javadoc documentation of the `AuditService` SPI is part of the eID Applet SDK package.

3.5.5. Alternative UI

The eID Applet offers its own user interface for interactive handling of eID card events. As some web application technologies (like Flex) like to construct their own user interface we created a Javascript based callback mechanism so that these web technologies can visualize the info messages themselves.

The web developer can install the info message callback inside a web page as follows:

```
<script src="https://www.java.com/js/deployJava.js"></script>
<script>
    var attributes = {
        code :'be.fedict.eid.applet.Applet.class',
        archive :'eid-applet-package.jar',
        width :1,
        height :1,
        mayscript :'true'
    };
    var parameters = {
        AppletService :'applet-service',
        MessageCallback :'messageCallback',
```

```
        MessageCallbackEx : 'messageCallbackEx'
    };
    var version = '1.6';
    deployJava.runApplet(attributes, parameters, version);
</script>
<script>
    function messageCallback(status, message) {

document.getElementById('appletMessage').innerHTML='<b>'+status+':'+message+'</
b>';
    }
    function messageCallbackEx(status, messageId, message) {

document.getElementById('appletMessageEx').innerHTML='<b>'+status+':'+messageId+':'+message+'</
b>';
    }
</script>
<div id="appletMessage">Message placeholder</div>
<div id="appletMessageEx">Message placeholder</div>
```

As you can see the web developer can install a Javascript callback function by setting the
`MessageCallback` eID Applet parameter. The `status` parameter can be either `NORMAL` or `ERROR`
. In our example we simply display the incoming message via some dynamic HTML. Of course
more complex visualizations are possible here. Via the `MessageCallbackEx` eID Applet parameter
you can even receive a machine processable message identifier.

> **mayscript**
>
> Don't forget the `mayscript:'true'` attribute, else the eID Applet will not be able
> to invoke Javascripts inside the browser window.

3.5.6. Requiring a secure smart card reader

The eID Applet Service can be configured to make the eID Applet to check whether the eID
operation that requires the user to enter the eID PIN code (in case of authentication or non-
repudiation signature, PIN change or PIN unblock) is being executed using a CCID secure smart
card reader. Although this feature could be spoofed it aims to increase the security awareness as
required for some applications. This feature can be enabled by setting the following `init-param`
on the `AppletServiceServlet`:

```
<init-param>
    <param-name>RequireSecureReader</param-name>
    <param-value>true</param-value>
</init-param>
```

> **⚠ Not everybody has a secure pinpad reader**
>
> Before enabling this feature, make sure that your target audience indeed has access to a secure pinpad reader.

3.5.7. Run-time selection of required eID identity data

Places where you can use `IncludeIdentity` , `IncludeAddress` , `IncludePhoto` or `IncludeCertificates` one can also use the `IdentityService` SPI to have run-time selection of the eID identity data objects that have to be retrieved from the eID card. To activate this functionality you need to implement the `IdentityService` interface. This interface can be found in the `eid-applet-service-spi` artifact. This service component (EJB3) session bean should be registered somewhere in JNDI. The JNDI location of this service component needs to be set via the following `init-param` on the `AppletServiceServlet` :

```
<init-param>
    <param-name>IdentityService</param-name>
    <param-value>your/location/in/jndi/IdentityServiceBean</param-value>
</init-param>
```

The Javadoc documentation of the `IdentityService` SPI is part of the eID Applet SDK package.

3.5.8. Identity Data Files

Some applications might require access to the actual raw identity data files. You can configure the eID Applet Service to push the raw identity data files (if available as specified during the request) into the HTTP session via the following `init-param` on the `AppletServiceServlet` :

```
<init-param>
    <param-name>IncludeDataFiles</param-name>
    <param-value>true</param-value>
</init-param>
```

This will make the eID identity file available within the HTTP session under the `eid.data.identity` session attribute as byte array. The eID address file will be available within the HTTP session under the `eid.data.address` session attribute as byte array.

4. Maven Integration

In this section we'll discuss the different aspects related to integrating the eID Applet within Maven based projects.

First of all, add the following repository configuration under `<repositories>` within you `pom.xml` Maven configuration file :

```xml
<repository>
    <id>e-contract</id>
    <url>https://www.e-contract.be/maven2/</url>
    <releases>
        <enabled>true</enabled>
    </releases>
</repository>
```

We assume that you're using a `<dependencyManagement>` element within your `pom.xml` Maven project file. Put the following declaration:

```xml
<dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-bom</artifactId>
    <version>1.2.0.Beta1</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

Integration of the eID Applet within a Java EE web application consists out of two tasks. First we have to add the eID Applet itself as web resource to the web application. In order to do so, add the following dependencies under `<dependencies>` :

```xml
<dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-package</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-js</artifactId>
    <scope>provided</scope>
</dependency>
```

Now you can include the eID Applet JAR and corresponding Javascript as web resource via:

```xml
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifactId>
```

```xml
    <executions>
        <execution>
            <id>unpack</id>
            <phase>process-resources</phase>
            <goals>
                <goal>unpack</goal>
            </goals>
            <configuration>
                <artifactItems>
                    <artifactItem>
                        <groupId>be.fedict.eid-applet</groupId>
                        <artifactId>eid-applet-js</artifactId>
                        <outputDirectory>
                                ${project.build.directory}/${project.artifactId}-
${project.version}
                        </outputDirectory>
                    </artifactItem>
                </artifactItems>
                <includes>*.js</includes>
            </configuration>
        </execution>
        <execution>
            <id>copy</id>
            <phase>process-resources</phase>
            <goals>
                <goal>copy</goal>
            </goals>
            <configuration>
                <artifactItems>
                    <artifactItem>
                        <groupId>be.fedict.eid-applet</groupId>
                        <artifactId>eid-applet-package</artifactId>
                        <type>jar</type>
                        <outputDirectory>
                                ${project.build.directory}/${project.artifactId}-
${project.version}
                        </outputDirectory>
                    </artifactItem>
                </artifactItems>
            </configuration>
        </execution>
    </executions>
</plugin>
```

The inclusion of the eID Applet Service depends somehow on what the runtime already provided by itself and what functionality you want to use exactly. However, most of the time the following dependency should suffice:

```xml
<dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-service</artifactId>
    <exclusions>
        <exclusion>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
        </exclusion>
        <exclusion>
            <groupId>com.lowagie</groupId>
            <artifactId>itext</artifactId>
        </exclusion>
        <exclusion>
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

5. eID Applet Web Application Deployment

You can deploy your eID Applet enabled web application over a lot of different network topologies, depending on the setup of your infrastructure. The easiest configuration is a setup where you terminate the SSL on the Application Server itself.

5.1. AJP proxy

Another option is to use an AJP proxy. An example of how to configure the Apache HTTPD AJP proxy is given below. In `/etc/httpd/conf.d/proxy_ajp.conf` you put:

```
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
```

Now you can configure (in some site specific `httpd` config file) the following:

```
<IfModule mod_proxy_ajp.c>
 ProxyRequests On
 ProxyVia On

 <Location /eid-applet-test>
  Order allow,deny
  Allow from all
  ProxyPass ajp://localhost:8009/eid-applet-test
 </Location>
```

```
</IfModule>
```

This AJP proxy can then terminate the SSL in a transparent way towards the Application Service.

On an Ubuntu server you can add the following configuration to your SSL site (which will most likely live under /etc/apache2/sites-enabled/default-ssl).

```
<IfModule mod_proxy_ajp.c>
                ProxyRequests On
                ProxyVia On

                <Location /eid-applet-test>
                        Order allow,deny
                        Allow from all
                        ProxyPass ajp://localhost:8009/eid-applet-test
                </Location>

                <Location /eid-applet-beta>
                        Order allow,deny
                        Allow from all
                        ProxyPass ajp://localhost:8009/eid-applet-beta
                </Location>
</IfModule>
```

5.2. Reverse proxy

Some configuration use non-AJP aware reverse proxies. An example on how to configure the Apache HTTPD as a reverse proxy is given below:

```
ProxyRequests Off

<Proxy *>
        Order deny,allow
        Allow from all
</Proxy>

<Location /eid-applet-test/>
        ProxyPass http://localhost:8080/eid-applet-test/
        ProxyPassReverse http://localhost:8080/eid-applet-test/
</Location>
```

Because the Application Server no longer receives the SSL information as provided by the AJP protocol, the eID Applet Service can no longer detect whether it's using a secure connection or

not. The eID Applet Service can be configured to skip the secure connection check using the following `init-param` on the `AppletServiceServlet`:

```xml
<init-param>
    <param-name>SkipSecureConnectionCheck</param-name>
    <param-value>true</param-value>
</init-param>
```

It is furthermore important to have a servlet container session cookie without the `HttpOnly` flag set. Else the eID Applet Service will push the eID identity credentials in the wrong Application Server HTTP session.

5.3. Tomcat 7

When running the eID Applet Service on Tomcat 7, you might receive the following error message within the eID Applet:

```
ERROR: no session cookie detected!
```

This error occurs because Tomcat 7 will per default set the `HttpOnly` flag on the `JSESSIONID` session cookie. This prevents Javascript and web browser plugins like the Java runtime plugin to receive the session cookie. It is important that the Java runtime plugin is capable of receiving the session cookie as the eID Applet Service must be able to push the eID data within the same web session as seen by the web browser.

There are several solutions:

- One can check the `useHttpOnly` attribute within the `<Context>` element and set it back to `false`.

- Or you can add `;jsessionid=...` to the `AppletService` eID Applet parameter so that the eID Applet Service will receive the correct `JSESSIONID` reference.

6. Accessibility

6.1. Java Accessibility Bridge

The Java Accessibility Bridge provides a bridge between the accessibility features of Java desktop applications - including applets - running inside the JVM, and the native assistive technologies of the operating system. This is not eID-specific, by the way, the JAB is part of the Java SE desktop platform.

There is no configuration required on the server side to enable the accessibility features of the eID applet. On the client side, however, users should verify that the JAB is installed on their computer and is supported by the assistive software of their choice.

6.1.1. Mac OS X

The bridging code is built-in on Mac OS X, no additional installation is required. Starting with v10.4, Mac OS X includes the screen reader Voice Over.

See also the *Java Development Guide for Mac OS X.* [http://developer.apple.com/mac/library/documentation/Java/Conceptual/Java14Development/04-JavaUIToolkits/JavaUIToolkits.html]

6.1.2. Windows

Windows users should install the latest version of the JAB for Windows, available for free from *Oracle's website.* [http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136191.html]

6.1.3. Linux

Ubuntu users should install the libaccess-bridge-java-jni package.

See also the documentation on the current and future accessibility architecture of *GNOME* [http://live.gnome.org/Accessibility/BonoboDeprecation] and *KDE* [http://accessibility.kde.org/developer/bridge.php] .

6.2. Screen Reader Support

Please note that, although recent screen readers should be able to read out the dialog boxes and status messages of the eID applet, the exact behavior depends on operating system, screen reader software and browser version.

For instance, some configurations do not automatically read out non-focusable text inside dialog boxes. However, it is often possible to read out the dialog boxes by using a keyboard command (JAWS and NVDA users may want to try **Insert+B**). Please consult your screen reader's manual for more information.

7. eID Applet Protocol

In this section we will elaborate on the eID Applet protocol used in the communication between the eID Applet and the eID Applet Service. If you use the eID Applet Service servlet implementation that comes with the eID Applet SDK you actually don't need to know the details of the eID Applet protocol. However, this information can be useful for web application developers that use other web frameworks than a Java EE servlet container based framework.

The eID Applet Protocol is based on the HTTP protocol using the POST method. Parameters are passed as HTTP headers and for binary data the HTTP body is used. The messages should be transported over a secure SSL connection.

In *Figure 4, "eID Applet Protocol Graph"* you find an automatically generated graph representation of the eID Applet protocol. A protocol run starts with the `HelloMessage` message sent by the eID

Applet to the eID Applet Service, which has been marked by the green vertex. Depending on the eID Applet Service configuration different paths will be followed ending in some red vertex.
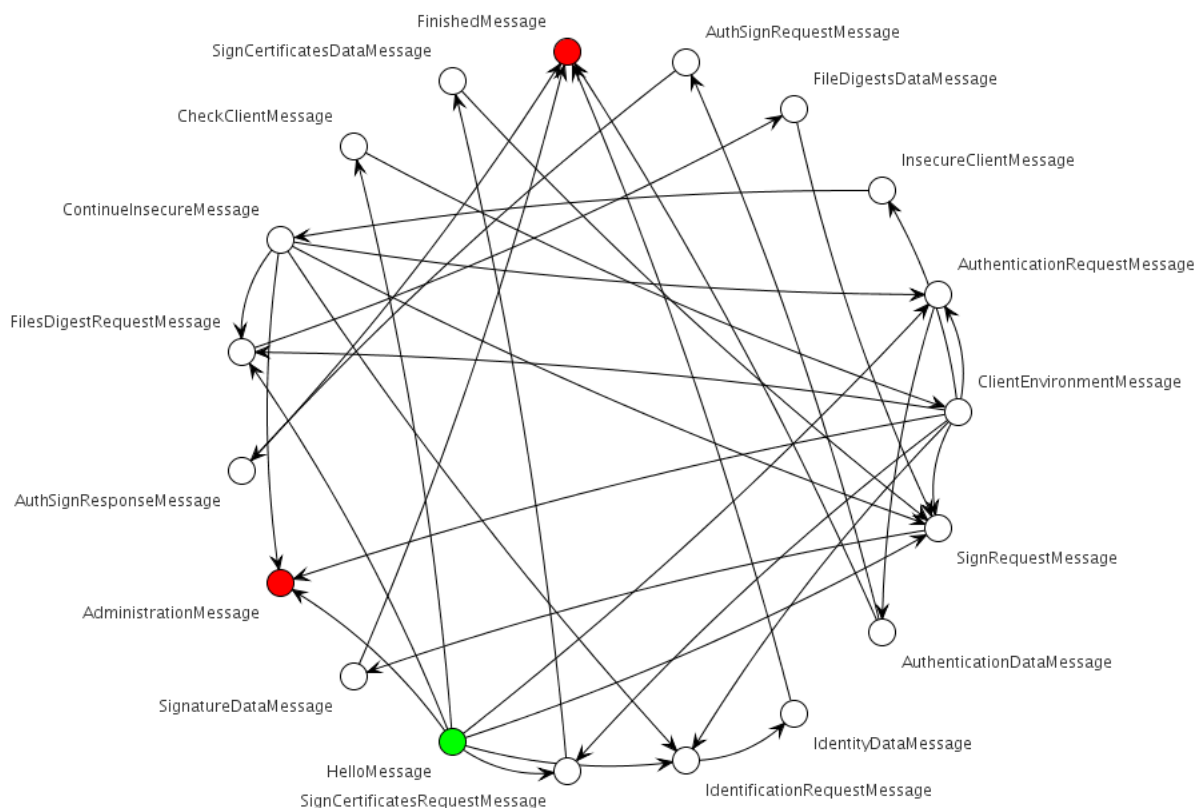
Figure 4. eID Applet Protocol Graph

> ⚠ **eID Applet Service implementations**
>
> Instead of implementing your own eID Applet Service for ASP.NET or PHP it might be easier to integrate the eID within your web applications by using FedICT products like the eID Identity Provider and the eID Digital Signature Service. As these products implement services based on open standards like OpenID you already have various implementations available for integrating via these open protocols.

7.1. eID Applet Protocol Messages

The following documentation has been generated automatically.

7.1.1. HelloMessage

This message starts a communication session between eID Applet and eID Applet Service. It sets the protocol state to: INIT

Table 2. HelloMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	HelloMessage
X-AppletProtocol-Language	false	Some String value.
X-AppletProtocol-Version	true	1

Allowed eID Applet Service response messages are: *Section 7.1.10, "IdentificationRequestMessage" Section 7.1.11, "CheckClientMessage" Section 7.1.13, "AuthenticationRequestMessage" Section 7.1.15, "AdministrationMessage" Section 7.1.16, "SignRequestMessage" Section 7.1.17, "FilesDigestRequestMessage" Section 7.1.18, "SignCertificatesRequestMessage"*

7.1.2. ClientEnvironmentMessage

This message is only accepted if the eID Applet Service protocol state is: ENV_CHECK

Table 3. ClientEnvironmentMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	ClientEnvironmentMessage
X-AppletProtocol-JavaVersion	true	Some String value.
X-AppletProtocol-JavaVendor	true	Some String value.
X-AppletProtocol-OSName	true	Some String value.
X-AppletProtocol-OSArch	true	Some String value.
X-AppletProtocol-OSVersion	true	Some String value.
X-AppletProtocol-NavigatorUserAgent	false	Some String value.
X-AppletProtocol-NavigatorAppName	false	Some String value.
X-AppletProtocol-NavigatorAppVersion	false	Some String value.
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

Allowed eID Applet Service response messages are: *Section 7.1.10, "IdentificationRequestMessage" Section 7.1.12, "InsecureClientMessage" Section 7.1.13, "AuthenticationRequestMessage" Section 7.1.15, "AdministrationMessage" Section 7.1.16, "SignRequestMessage" Section 7.1.17, "FilesDigestRequestMessage" Section 7.1.18, "SignCertificatesRequestMessage"*

7.1.3. AuthenticationDataMessage

This message is only accepted if the eID Applet Service protocol state is: AUTHENTICATE

Table 4. AuthenticationDataMessage HTTP headers

Header name	Required	Value
`X-AppletProtocol-Type`	true	`AuthenticationDataMessage`
`X-AppletProtocol-SignatureValueSize`	true	Some Integer value.
`X-AppletProtocol-SaltValueSize`	true	Some Integer value.
`X-AppletProtocol-SessionIdSize`	false	Some Integer value.
`X-AppletProtocol-AuthnCertFileSize`	true	Some Integer value.
`X-AppletProtocol-CaCertFileSize`	true	Some Integer value.
`X-AppletProtocol-RootCaCertFileSize`	true	Some Integer value.
`X-AppletProtocol-SignCertFileSize`	false	Some Integer value.
`X-AppletProtocol-IdentityFileSize`	false	Some Integer value.
`X-AppletProtocol-AddressFileSize`	false	Some Integer value.
`X-AppletProtocol-PhotoFileSize`	false	Some Integer value.
`X-AppletProtocol-IdentitySignatureFileSize`	false	Some Integer value.
`X-AppletProtocol-AddressSignatureFileSize`	false	Some Integer value.
`X-AppletProtocol-NationalRegistryCertFileSize`	false	Some Integer value.
`X-AppletProtocol-ServerCertFileSize`	false	Some Integer value.
`X-AppletProtocol-TransactionMessageSignatureSize`	false	Some Integer value.
`X-AppletProtocol-Version`	true	`1`

HTTP body should contain the data.

Allowed eID Applet Service response messages are: *Section 7.1.19, "FinishedMessage"* *Section 7.1.14, "AuthSignRequestMessage"*

7.1.4. AuthSignResponseMessage

This message is only accepted if the eID Applet Service protocol state is: AUTH_SIGN

Table 5. AuthSignResponseMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	AuthSignResponseMessage
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

Allowed eID Applet Service response messages are: *Section 7.1.19, "FinishedMessage"*

7.1.5. SignatureDataMessage

This message is only accepted if the eID Applet Service protocol state is: SIGN

Table 6. SignatureDataMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	SignatureDataMessage
X-AppletProtocol-SignatureValueSize	true	Some Integer value.
X-AppletProtocol-SignCertFileSize	true	Some Integer value.
X-AppletProtocol-CaCertFileSize	true	Some Integer value.
X-AppletProtocol-RootCaCertFileSize	true	Some Integer value.
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

Allowed eID Applet Service response messages are: *Section 7.1.19, "FinishedMessage"*

7.1.6. FileDigestsDataMessage

This message is only accepted if the eID Applet Service protocol state is: DIGEST

Table 7. FileDigestsDataMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	FileDigestsDataMessage

Header name	Required	Value
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

Allowed eID Applet Service response messages are: *Section 7.1.16, "SignRequestMessage"*

7.1.7. ContinueInsecureMessage

This message is only accepted if the eID Applet Service protocol state is: INSECURE

Table 8. ContinueInsecureMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	ContinueInsecureMessage
X-AppletProtocol-Version	true	1

Allowed eID Applet Service response messages are: *Section 7.1.10, "IdentificationRequestMessage" Section 7.1.13, "AuthenticationRequestMessage" Section 7.1.15, "AdministrationMessage" Section 7.1.16, "SignRequestMessage" Section 7.1.17, "FilesDigestRequestMessage"*

7.1.8. SignCertificatesDataMessage

This message is only accepted if the eID Applet Service protocol state is: SIGN_CERTS

Table 9. SignCertificatesDataMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	SignCertificatesDataMessage
X-AppletProtocol-SignCertFileSize	true	Some Integer value.
X-AppletProtocol-CaCertFileSize	true	Some Integer value.
X-AppletProtocol-RootCaCertFileSize	true	Some Integer value.
X-AppletProtocol-IdentityFileSize	false	Some Integer value.
X-AppletProtocol-AddressFileSize	false	Some Integer value.
X-AppletProtocol-PhotoFileSize	false	Some Integer value.
X-AppletProtocol-IdentitySignatureFileSize	false	Some Integer value.

Header name	Required	Value
X-AppletProtocol-AddressSignatureFileSize	false	Some Integer value.
X-AppletProtocol-NationalRegistryCertFileSize	false	Some Integer value.
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

Allowed eID Applet Service response messages are: *Section 7.1.16, "SignRequestMessage"*

7.1.9. IdentityDataMessage

This message is only accepted if the eID Applet Service protocol state is: IDENTIFY

Table 10. IdentityDataMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	IdentityDataMessage
X-AppletProtocol-IdentityFileSize	true	Some Integer value.
X-AppletProtocol-AddressFileSize	false	Some Integer value.
X-AppletProtocol-PhotoFileSize	false	Some Integer value.
X-AppletProtocol-IdentitySignatureFileSize	false	Some Integer value.
X-AppletProtocol-AddressSignatureFileSize	false	Some Integer value.
X-AppletProtocol-RrnCertFileSize	false	Some Integer value.
X-AppletProtocol-RootCertFileSize	false	Some Integer value.
X-AppletProtocol-AuthnCertFileSize	false	Some Integer value.
X-AppletProtocol-SignCertFileSize	false	Some Integer value.
X-AppletProtocol-CaCertFileSize	false	Some Integer value.
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

Allowed eID Applet Service response messages are: *Section 7.1.19, "FinishedMessage"*

7.1.10. IdentificationRequestMessage

Table 11. IdentificationRequestMessage HTTP headers

Header name	Required	Value
`X-AppletProtocol-Type`	true	`IdentificationRequestMessage`
`X-AppletProtocol-IncludeAddress`	false	Some boolean value.
`X-AppletProtocol-IncludePhoto`	false	Some boolean value.
`X-AppletProtocol-IncludeIntegrityData`	false	Some boolean value.
`X-AppletProtocol-RemoveCard`	false	Some boolean value.
`X-AppletProtocol-IncludeCertificates`	false	Some boolean value.
`X-AppletProtocol-IdentityDataUsage`	false	Some String value.
`X-AppletProtocol-Version`	true	1

This message will perform an eID Applet protocol state transition to: IDENTIFY

7.1.11. CheckClientMessage

Table 12. CheckClientMessage HTTP headers

Header name	Required	Value
`X-AppletProtocol-Type`	true	`CheckClientMessage`
`X-AppletProtocol-Version`	true	1

This message will perform an eID Applet protocol state transition to: ENV_CHECK

7.1.12. InsecureClientMessage

Table 13. InsecureClientMessage HTTP headers

Header name	Required	Value
`X-AppletProtocol-Type`	true	`InsecureClientMessage`
`X-AppletProtocol-WarnOnly`	false	Some boolean value.
`X-AppletProtocol-Version`	true	1

This message will perform an eID Applet protocol state transition to: INSECURE

7.1.13. AuthenticationRequestMessage

Table 14. AuthenticationRequestMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	AuthenticationRequestMessage
X-AppletProtocol-RemoveCard	false	Some boolean value.
X-AppletProtocol-IncludeHostname	false	Some boolean value.
X-AppletProtocol-IncludeInetAddress	false	Some boolean value.
X-AppletProtocol-Logoff	false	Some boolean value.
X-AppletProtocol-PreLogoff	false	Some boolean value.
X-AppletProtocol-SessionIdChannelBinding	false	Some boolean value.
X-AppletProtocol-ServerCertificateChannelBinding	false	Some boolean value.
X-AppletProtocol-IncludeIdentity	false	Some boolean value.
X-AppletProtocol-IncludeCertificates	false	Some boolean value.
X-AppletProtocol-IncludeAddress	false	Some boolean value.
X-AppletProtocol-IncludePhoto	false	Some boolean value.
X-AppletProtocol-IncludeIntegrityData	false	Some boolean value.
X-AppletProtocol-RequireSecureReader	false	Some boolean value.
X-AppletProtocol-NoPKCS11	false	Some boolean value.
X-AppletProtocol-TransactionMessage	false	Some String value.
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

This message will perform an eID Applet protocol state transition to: AUTHENTICATE

7.1.14. AuthSignRequestMessage

Table 15. AuthSignRequestMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	AuthSignRequestMessage
X-AppletProtocol-DigestAlgo	true	Some String value.
X-AppletProtocol-Message	true	Some String value.
X-AppletProtocol-Logoff	false	Some boolean value.
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

This message will perform an eID Applet protocol state transition to: AUTH_SIGN

7.1.15. AdministrationMessage

This message stops a communication session between eID Applet and the eID Applet Service.

Table 16. AdministrationMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	AdministrationMessage
X-AppletProtocol-ChangePin	false	Some boolean value.
X-AppletProtocol-UnblockPin	false	Some boolean value.
X-AppletProtocol-RemoveCard	false	Some boolean value.
X-AppletProtocol-Logoff	false	Some boolean value.
X-AppletProtocol-RequireSecureReader	false	Some boolean value.
X-AppletProtocol-Version	true	1

7.1.16. SignRequestMessage

Table 17. SignRequestMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	SignRequestMessage
X-AppletProtocol-DigestAlgo	true	Some String value.
X-AppletProtocol-Description	false	Some String value.
X-AppletProtocol-RemoveCard	false	Some boolean value.
X-AppletProtocol-Logoff	false	Some boolean value.

Header name	Required	Value
X-AppletProtocol-RequireSecureReader	false	Some boolean value.
X-AppletProtocol-NoPKCS11	false	Some boolean value.
X-AppletProtocol-Version	true	1

HTTP body should contain the data.

This message will perform an eID Applet protocol state transition to: SIGN

7.1.17. FilesDigestRequestMessage

Table 18. FilesDigestRequestMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	FilesDigestRequestMessage
X-AppletProtocol-DigestAlgo	true	Some String value.
X-AppletProtocol-Version	true	1

This message will perform an eID Applet protocol state transition to: DIGEST

7.1.18. SignCertificatesRequestMessage

Table 19. SignCertificatesRequestMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	SignCertificatesRequestMessage
X-AppletProtocol-IncludeIdentity	false	Some boolean value.
X-AppletProtocol-IncludeAddress	false	Some boolean value.
X-AppletProtocol-IncludePhoto	false	Some boolean value.
X-AppletProtocol-IncludeIntegrityData	false	Some boolean value.
X-AppletProtocol-Version	true	1

This message will perform an eID Applet protocol state transition to: SIGN_CERTS

7.1.19. FinishedMessage

This message stops a communication session between eID Applet and the eID Applet Service.

Table 20. FinishedMessage HTTP headers

Header name	Required	Value
X-AppletProtocol-Type	true	FinishedMessage
X-AppletProtocol-ErrorCode	false	Some ErrorCode value.
X-AppletProtocol-Version	true	1

A. eID Applet Developer's Guide License

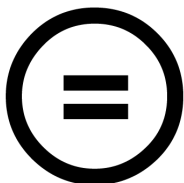This document has been released under the Creative Commons license.

You are free to Share — to copy, distribute and transmit the work.

You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

You may not use this work for commercial purposes.

You may not alter, transform, or build upon this work.

More information about the Creative Commons license conditions can be found at *Creative Commons organization* [http://creativecommons.org/] .

B. eID Applet License

The eID Applet source code has been released under the GNU LGPL version 3.0.

```
This is free software; you can redistribute it and/or modify it under the terms
of the GNU Lesser General Public License version 3.0 as published by the Free
Software Foundation.

This software is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along
with this software; if not, see http://www.gnu.org/licenses/.
```

C. Revision history

Table C.1. Revision history

Date	Author	Description
26 Jan 2009	Frank Cornelis	Initial version.
22 Apr 2009	Frank Cornelis	1.0.0-beta-1
29 May 2009	Frank Cornelis	1.0.0-beta-2
24 Jul 2009	Frank Cornelis	1.0.0-beta-3
18 Sep 2009	Frank Cornelis	1.0.0-beta-4
22 Nov 2009	Frank Cornelis	1.0.0-rc-1
16 Dec 2009	Frank Cornelis	1.0.0-rc-2
6 Jan 2010	Frank Cornelis	1.0.0-rc-3
11 Jan 2010	Frank Cornelis	1.0.0.GA
25 June 2010	Frank Cornelis	1.0.1.RC1
5 Aug 2010	Frank Cornelis	1.0.1.RC2
18 Aug 2010	Frank Cornelis	1.0.1.RC3
15 Sep 2010	Frank Cornelis	1.0.1.GA
25 Feb 2011	Frank Cornelis	1.0.2.GA
20 Jun 2011	Frank Cornelis	1.0.3.GA
9 Dec 2011	Frank Cornelis	1.0.4.GA
2 Oct 2012	Frank Cornelis	1.0.5.GA

Date	Author	Description
4 Nov 2013	Frank Cornelis	1.1.0.GA