
JInfoCard Documentation

Release 2.0.1

**Jens Fromm, Steffen Konegen, Jonas Pattberg,
Christopher Taylor**

July 30, 2009

CONTENTS

| | | |
|-----------|--|-----------|
| 1 | Imprint | 1 |
| 2 | Front Matter | 3 |
| 3 | Introduction | 5 |
| 3.1 | History | 5 |
| 4 | InformationCard Description | 7 |
| 4.1 | Basic Information | 7 |
| 4.2 | How InformationCard Works | 7 |
| 4.3 | Interoperability of InformationCards | 8 |
| 4.4 | Advantages | 8 |
| 5 | Technical Specification | 11 |
| 5.1 | Technologies used by JInfoCard | 11 |
| 5.2 | JInfoCard Framework Implementation | 12 |
| 6 | JInfoCard Architecture | 13 |
| 6.1 | Overview | 13 |
| 6.2 | The JInfoCard Framework | 13 |
| 6.3 | The MiniShop Sample Web Application | 15 |
| 7 | Setup and Installation | 17 |
| 7.1 | Getting the Source Code | 17 |
| 7.2 | Framework: | 17 |
| 7.3 | MiniShop: | 17 |
| 8 | Frequently Asked Questions | 21 |
| 9 | Glossary | 23 |
| 10 | Licensing | 25 |
| 10.1 | Fraunhofer Institute | 25 |
| | Bibliography | 27 |
| | Index | 29 |

IMPRINT

Managing Editor:

Jens Fromm

jens.fromm@fokus.fraunhofer.de

Authors:

Jens Fromm jens.fromm@fokus.fraunhofer.de

Stefen Konegen stefen.konegen@fokus.fraunhofer.de

Jonas Pattberg jonas.pattberg@fokus.fraunhofer.de

Christopher Taylor christopher.taylor@fokus.fraunhofer.de

The Fraunhofer Institute for Open Communication Systems FOKUS

Director: Prof. Dr. Dr. h.c. Radu Popescu-Zeletin

Deputy Director: Dipl.-Ing. Berthold Butscher

Kaiserin-Augusta-Allee 31

10589 Berlin

Fon: +49 30 34637000

Fax: +49 30 34638000

email: info@fokus.fraunhofer.de

is an institution of the Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.

Hansastraße 27 c

80686 München

Phone +49 (0) 89 / 12 05- 0

Fax +49 (0) 89 / 12 05-75 31

VAT Identification Number in accordance with §27 a VAT Tax Act: DE 129515865

Court of jurisdiction

Amtsgericht München (district court)

Registered nonprofit association

Registration no. VR 4461

Executive Board

Prof. Dr. Hans-Jörg Bullinger, President,

Corporate Management and Research

Dr. Ulrich Buller, Research Planning

Dr. Alfred Gossner, Finance and Controlling (incl. Business Administration, Purchasing and Real Estate)

Dr. Dirk-Meints Polter, Human Resources and Legal Affairs

FRONT MATTER

JInfoCard has been developed by The Fraunhofer Institute for Open Communication Systems and Microsoft. Microsoft's role in this project has been to administer funding, architectural and technical guidance. JInfoCard is published under the BSD license.

INTRODUCTION

This document describes the JInfoCard, a framework for Java-based web application servers that enables authentication using the Information Card system developed by Microsoft. This chapter provides a general overview of the history of identity management applications, followed by a description of the Information Card technology in the chapter *InformationCard Description*. The chapters *Technical Specification* and *JInfoCard Architecture* offer a detailed description of the implementation of JInfoCard, chapter *Setup and Installation* shows how to integrate Information Card support into your own applications and how to deploy the “MiniShop” sample application.

3.1 History

Internet users have many digital identities, each with its own context. One digital identity on eBay, maybe another one on Amazon, an email provider, a work station, a photo community, home banking, car rental programs and a few more services. Anonymous surfing and reading pages on different boards would still be possible, but anyone who wants to use these web services to contribute to a discussion, send a mail, do bank transactions, rent a car, post a text or a picture would have to identify themselves by registering with a web site. The personal data required for registration needs to be filled out on every web site. Until now it has been possible to handle this by using a registered account which contains personal data provided by the user which is protected with a username and password. The consequence of this, besides many passwords, is that a lot of different information about the user such as their name, post code, birth date, bank account and much more are stored on each web service where the user is registered. The multiplicity of accounts and passwords that users must keep track of and the variety of methods used for authenticating to sites results not only in user frustration but also insecure practices such as reusing the same account names and passwords at many sites. In an effort to address this deficiency, numerous digital identity systems have been introduced, each with its own strengths and weaknesses. But no one single system meets the needs of every digital identity scenario. As a first step towards managing all these different problems Microsoft created .NET Passport. It was supposed to become the standard for registration systems.

3.1.1 Microsoft .Net Passport

Microsoft .NET Passport was a “unified-login” service which was developed as a part of the company’s .NET strategy in 1999. It was based on a Single-Sign-On (SSO) solution that allowed users to log into many web sites with only one account. The condition was that the web site had to support .NET Passport. In the past, it was mostly Microsoft’s own web sites such as MS hotmail or MSN that supported .NET Passport. It was only necessary to register with .NET Passport once in order to use all participant web pages and services. It was not necessary to repeatedly give dates and information to different web services. .NET Passport used standard Web technologies and techniques, such as Secure Sockets Layer (SSL), HTTP redirects, cookies, JavaScript and strong symmetric key encryption to deliver the single sign-in service.

The whole .NET Passport system was administered centrally and all users and participating web services had to register themselves with Microsoft. This meant that all data was stored on Microsoft servers. This was the biggest disadvantage and the main focus of criticism for this SSO solution project. Because of the central structure it was quite often a target for attacks and when participant web services become aware of these gaps in security they stopped supporting .NET Passport. As a result, the use of .NET Passport did not become widespread.

3.1.2 Microsoft CardSpace / InformationCard

In May 2005 Microsoft published MS CardSpace (formerly InfoCard) as a successor to and replacement of MS .NET Passport in order to guarantee an easy and secure way to administer identities. The core idea of MS CardSpace is to be a meta-identity system that is independent of protocols, which further supports HTTP(S), RTSP, RTCP and makes it possible for many different identity systems to play easily together.

CardSpace overcomes barriers like the storage of user data on external servers by storing private data on the user's side and offering the option that trusted third parties could be responsible for the actual storage of authentication data. The card itself holds only Meta information on how to actually access private user data from an *Identity Provider* (IdP). Identity Providers may be trusted third parties holding a *Security Token Service* (STS) or even the user himself. An improved level of protection is reached before various forms of identity attacks such as phishing can occur.

In particular, since the users are in possession of their own data, they can decide which information a website will receive. Different web services can store different data, from different cards. This is possible because InformationCard allows them to have as many digital identities as anyone needs. It is comparable with different credit cards or membership cards in the real world. Moreover with CardSpace it is possible to use various online services with only one user account/card.

Design goals include seamless integration into existing browsers and websites independent of the underlying platforms, as well as improving specific browser issues that affect the user experience directly, like graceful fall-back in case of missing Windows Card Space (formerly InfoCard) support or the possibility to handle WCS with security settings set to a high level. The underlying architecture itself with all its subjects is called "The Identity Metasystem". It was first developed by Microsoft's Identity Architect, Kim Cameron, and it is actually more of a shared vision for solving basic identity challenges than a Microsoft specific initiative.

3.1.3 JInfoCard

In order to enable Information Card support for Java-based web application servers, Fraunhofer FOKUS developed a platform independent plugin called JInfoCard. The project's focus was on the development of an application module that can be easily integrated for core functionality as well as to provide the necessary documentation to include this functionality in the existing authentication mechanisms so that integration can be achieved with minimum effort.

With this OpenSource project JInfoCard it will be possible to simply enhance an already existing web application by adding InformationCard support. The purpose of the Fraunhofer Institute's project is to demonstrate *Information Card* interoperability on heterogeneous platforms written using Java language to support Apache Tomcat and other web application servers.

INFORMATIONCARD DESCRIPTION

Information Cards is a useful application which makes user activities in the net easier to handle. It is gaining user awareness and is supported by a range of sites already. The application is simple to use, but what is happening in the background? How does InformationCard work? What are its advantages? General questions and particularly administrators' questions about InformationCard, its construction and expiry data will be answered in the following sections.

4.1 Basic Information

In order to illustrate the JInfoCard procedure, the basics must be understood first. The extension consists of several components which play together in a framework, thus enabling InformationCard to be used. Basically, the user connects over the two different protocol types HTTP and HTTPS to a *relying party*. HTTPS is the standard for encrypted transference of data between browser and web server. Thereby some *claims* representing personal information about the user are interchanged. The framework further uses a PPID, which means Personal Private Identifier. PPID is unique for each web page, an unequivocal ID. It is generated by InformationCard's master key and a public key. Every InformationCard contains its own PPID for each website. WCS, which stands for Windows Card Space (formerly InfoCard) will be referred to in the following description.

4.2 How InformationCard Works

When a user connects to a web service with CardSpace support, an Identity Selector will open. It manages the user's different digital identities and displays all available cards that may be used for authentication. The user simply selects a card and depending on the card type (managed or self-issued) a security token is sent back with an SSL certificate encrypted to the site from either an IP or the local computer.

The unique identifier based on the Personal Private Identifier (PPID) is used to combine a card with the account. Setting managed cards aside for a moment, here is the general flow for self-issued cards:

1. The user connects with a browser to a relying party (RP) holding a desired service
 - selects a service and WCS login appears automatically, or
 - performs a direct login query via WCS where the usual U/P login exists
2. The RP sends back its policy specifying the set of claims it requires and accepts.
3. The browser pops up an Identity Selector presenting a list of cards which meet these requirements.
The user can only select cards which satisfy the requirements of the RP.
4. The user selects a card
5. The Identity Selector connects to the IP/STS - in this case the client itself - and receives a security token.
6. The security token is sent back to the RP. When communicating over a secure connection, is encrypted using the website's SSL certificate.
7. The RP validates the token and grants access to the services

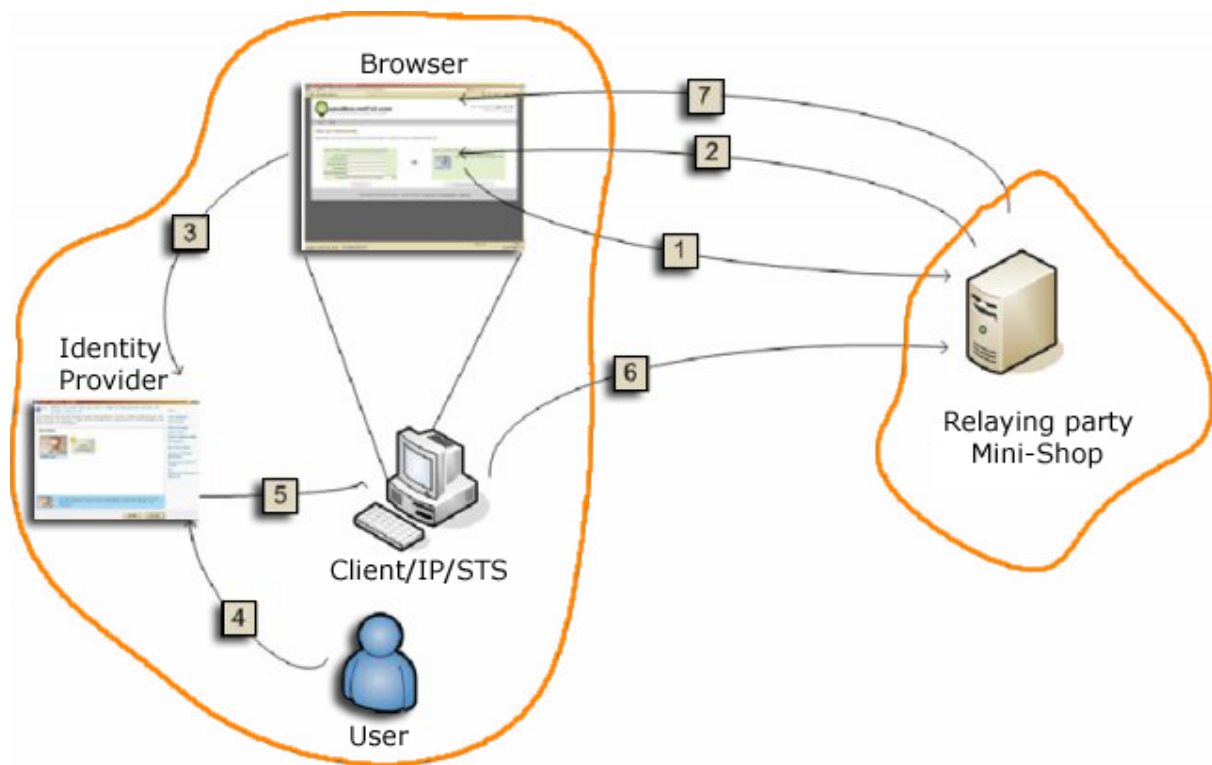


Figure 4.1: How InformationCard works

4.3 Interoperability of InformationCards

“Interoperability means the ability of information and communication technology (ICT) systems and of the business processes they support to exchange data and to enable the sharing of information and knowledge...” [IDABC-EIF]

On the basis of this definition three kinds of interoperability can be distinguished:

- Organizational interoperability
- Semantic interoperability, and
- Technical interoperability

This document focuses on the technical aspects of interoperability defined by the EICTA as “the ability of two or more networks, systems, devices, applications or components to exchange information between them and to use the information so exchanged” [EICTA].

The InformationCard model is built on open, documented communication standards to achieve interoperability.

JInfoCard uses the following open standards:

- XML - Extensible Markup Language
- *SAML* - Security Assertion Markup Language
- WS-Security, WS-Trust, WS-MetadataExchange, WS-SecurityPolicy

4.4 Advantages

If InformationCard is used, many advantages reveal themselves. Above all, with the JInformation- Card extension, trouble free use is also possible for Java carried web sites. Some advantages for client and server are shown in the

following overview.

4.4.1 Server

JInfoCard can be used by existing applications. An existing system can remain unchanged. JInfoCard is an extension which allows using InformationCard functionality with an existing web application with minimum integration effort.

A peripheral server structure

Data stored on client and server site, not on a central server like MS Passport

No Cross-Site-Scripting is possible

External code can't be executed on the server or client site

Avoids DNS-Spoofing

The PPID is unique and generated for every single RP. It is a pair wise unique identifier for a given user identity and a relaying party combination

Uses XML

XML files are not bound to a certain system and one can process them and consider them platform independent

4.4.2 Client

Easy identity management

Stored user data are visible and it is possible for users to have different identities

Better and more secure usages of websites

The risk of attacks may be minimized (see on 2.4.1) and InformationCard guarantees comfortable use of web sites

Can self determine which information is provided and to whom

Before sending a Card, the user can control his or her personal details

Avoids phishing

No passwords are necessary in combination with the user name

No more weak passwords

InformationCard uses no password, only an attached card is necessary

No dictionary attacks

No (weak) passwords are required to use InformationCard

TECHNICAL SPECIFICATION

The main goal of the design of the JInfoCard framework has been the integration of the framework into currently existent web sites – it should be done very easily and, without the need to make huge changes to the site. This chapter shows how this goal was (hopefully) achieved.

5.1 Technologies used by JInfoCard

The following sections give a quick overview of the technologies used in JInfoCard. The section *Servlet-Filter Framework* gives a short introduction to servlet filters, *Java Server Pages* and *Expression Language* describe JSPs and EL, respectively.

5.1.1 Java Server Pages

Java Server Pages (JSP) and Servlets allow dynamic web pages to be generated in order to answer to client's web inquiry. Servlets are compiled Java classes and they can be used as a substitute with CGI programs. JSPs are text-based documents, consisting of static html and Java source code. It is used to describe of dynamic behaviour and to generate dynamic contents.

JSP is firstly accessible via java Servlet source code transfers and is compiled afterwards.

The configuration file for JSP/Servlets is called Web.xml and is used to tell the Web server where the various parts of the application can be found, how to access them and, sometimes, who can access them.

5.1.2 Expression Language

The JSP Expression Language (EL) allows for increased separation of concerns when designing Java-based web applications. A JSP using EL is essentially a regular HTML file which only serves as a *view* of the data provided by the back-end layers of the application. It is designed to be easy to manipulate by non-developers, i.e. web designers. This goal is achieved by removing all non presentation related code from the JSP and instead including expressions of the following form:

```
${firstName}
```

For details on the syntax of these expressions and how they are resolved, please refer to [\[EL\]](#).

5.1.3 Servlet-Filter Framework

A Filter is a new component type on Java Servlet specification version 2.3 [\[JSR053\]](#). This filter dynamically intercepts requests and responses in order to transform or use the information contained in the requests or responses. Universal functions are provided which can be attached to any kind of Servlets or JSP page. Filters do not create responses themselves.

What is important here is that filters provide the ability to encapsulate recurring tasks in reusable units. To transform a response from a Servlet or a JSP page the filter can also be used.

Furthermore developers have the opportunity to write transformation components that have been portable across containers since the introduction of the filter.

Other types of functions are encryption, tokenizing, triggering resource access events, logging and auditing, image conversion, data compression, XSL/T transformations of XML content, mime-type chaining, and caching.

The reusable JInfoCard framework component, referred to as “the Framework” in the following sections, is implemented as a Servlet Filter. It is configured in a web application’s `web.xml` configuration file and can be used to protect certain parts of a web application from unauthorized access with very few changes to the application’s source code.

5.2 JInfoCard Framework Implementation

The JInfoCard Framework is structured into three sub-packages of `de.fraunhofer.fokus`. In general, they contain Java interfaces which are implemented in the corresponding sub-sub-package `impl` (i.e. `jic.filter.Framework` is implemented in `jic.filter.impl.FrameworkImpl`):

jic This package contains common classes used by the other packages. Currently this is only the class `JICException`, the exception which is raised by the framework when an error occurs.

jic.filter This package contains the class `JICFilter`, which implements the Servlet Filter interface. It retrieves it’s configuration from the web application’s `web.xml`. When a user accesses a resource it is configured to protect, the filter intercepts the request and extracts the identity information (SAML assertion) from it. `JICFilter` then uses the method `Framework.getID()` to process the token and places the returned `ClaimIdentity` object into the Session from where it can be retrieved by the web application as described in the section *Displaying Claims in Java Server Pages (JSPs)*.

If the identity information the user submitted is invalid or an error occurs when processing it, a `ServletException` is raised. The web application can catch the exception or redirect the user to an error page.

jic.framework The interface `Framework` in this package defines a method `getID` which is overloaded to accept a SAML token in a variety of forms - as a `String`, an `InputStream` or a Document Object Model (DOM) Document (see [DOM3] and [DOM3Java] for details). It decrypts the assertion, if necessary, and validates the signature on it before processing it into a `ClaimIdentity` which can be used by the web application to determine the user’s identity.

The method optionally takes a `PrivateKey` as a parameter, which is used to decrypt the SAML assertion when a secure connection via SSL is used.

The interface `TokenValidator` is intended as a future extension point which may allow client code to perform custom validation of the SAML assertion.

Note: The core Framework, composed of the interface `jic.framework.Framework` and it’s implementation `jic.framework.impl.FrameworkImpl` have no connection to Java Servlet technology and thus can be reused even outside a web application context.

jic.identity This package contains the `ClaimIdentity` which is returned by `Framework.getID()`. It is essentially a subclass of `java.util.Map<String, Claim>`, however the method `get()` is overridden to provide a range of “convenience accessors” for the standard claims defined in [ISIPv15]. Please see the method’s documentation for the exact list.

The Class `Claim` is a regular `JavaBean` which holds the claim type and -value, as well as a reference to the `ClaimIdentity` of which it is a part, for a single claim.

JINFOCARD ARCHITECTURE

6.1 Overview

The JInfoCard Framework is composed of two distinct components, the Framework which can be integrated into third-party web applications and a sample web application, called MiniShop, which shows how such an integration is possible. The components are packaged as separate projects and can be used individually as described in the installation and setup instructions below.

6.2 The JInfoCard Framework

The Framework is JInfoCard's central component. It transforms the SAML token sent by the user into a ClaimIdentity object containing the user's identity attributes.

This functionality is realized in the class `FrameworkImpl` which implements the interface `Framework`, found in the Java package `de.fraunhofer.fokus.jic.framework.impl`. The method `getID` is overloaded to accept the SAML input in a variety of formats, ranging from a plain text serialisation to an already parsed DOM document. If a private key is supplied, the JInfoCard Framework will attempt to decrypt the token. This is necessary if the relying party can be accessed over an SSL connection, because in this case the client will encrypt the token before it is sent.

In order to insure that the token cannot be modified or otherwise misused by a third party, it contains a digital signature and a validity period, both of which are verified by the JInfoCard Framework.

Third party web applications can easily integrate the framework by configuring the Servlet filter `de.fraunhofer.fokus.jic.filter.JICFilter` in their `web.xml` configuration file as follows:

```
<filter>
  <filter-name>JICFilter</filter-name>
  <filter-class>
    de.fraunhofer.fokus.jic.filter.JICFilter
  </filter-class>
  <init-param>
    <!--
      the name of the HTTP request parameter containing the SAML token
      supplied by the user.
    -->
    <param-name>request_param_name</param-name>
    <param-value>xmltoken</param-value>
  </init-param>
  <init-param>
    <!--
      The name of the Session attribute into which the resulting
      ClaimIdentity object should be placed.
    -->
    <param-name>userid_request_attr</param-name>
    <param-value>userdata</param-value>
  </init-param>
</filter>
```

```

</init-param>
<init-param>
  <!--
  The default character encoding to assume when not supplied by
  the client. Note: to comply with the HTTP and Servlet specs, set
  this setting to iso-8859-1. JInfoCard uses UTF-8 as default in
  order to circumvent a bug in Windows CardSpace.
  -->
  <param-name>default_character_encoding</param-name>
  <param-value>UTF-8</param-value>
</init-param>

<!--
SSL setup.
When deploying an HTTP-only (noSSL) relying party,
you can omit these settings.
-->
<init-param>
  <!--
  the path should be relative to the WEB-INF/classes
  directory.
  -->
  <param-name>keystore_file</param-name>
  <param-value>/path/to/keystore.p12</param-value>
</init-param>
<init-param>
  <param-name>keystore_alias</param-name>
  <param-value>tomcat</param-value>
</init-param>
<init-param>
  <param-name>keystore_pswd</param-name>
  <param-value>s3cr3t</param-value>
</init-param>
<init-param>
  <param-name>private_key_pswd</param-name>
  <param-value>3v3n_m0r3_s3cr3t</param-value>
</init-param>
<init-param>
  <!-- can be PKCS12 or JKS -->
  <param-name>keystore_type</param-name>
  <param-value>PKCS12</param-value>
</init-param>
</filter>
<filter-mapping>
  <!--
  specify the URLs or servlets to which the filter should be
  applied.
  -->
  <filter-name>JICFilter</filter-name>
  <url-pattern>/shop.jsp</url-pattern>
</filter-mapping>
<error-page>
  <!--
  the Servlet or JSP to use if an invalid token is detected
  by the JInfoCard Framework.
  -->
  <exception-type>javax.servlet.ServletException</exception-type>
  <location>/error.jsp</location>
</error-page>

```

6.3 The MiniShop Sample Web Application

The sample webapp demonstrates an easy way to integrate Information Cards into a web application. It is implemented with Java Server Pages (JSPs) and makes use of the JICFilter Framework component described above. Please refer to the chapter *Setup and Installation* for detailed instructions on how to setup and deploy the sample web application.

6.3.1 Overview

The file `index.jsp` contains the `<object>` and `<form>` tags used to invoke the Identity Selector on the user's computer. When the user submits his or her digital identity to the MiniShop (`shop.jsp`), the JICFilter is invoked, thus causing the SAML assertion to be processed. The resulting `ClaimIdentity` is placed into the Session attribute "userdata".

The MiniShop makes use of a custom claim (which is supplied by a trusted Identity Provider) to determine whether a user is of full age. This claim is processed in a second Servlet Filter, `AuthorisationFilter` (in the Java Package `de.fraunhofer.fokus.minishop`), which extracts the value of said claim and adds additional attributes to the Session which the JSP `shop.jsp` then uses to determine which articles should be displayed to the user. The `AuthorisationFilter` is configured as follows:

```
<filter>
  <filter-name>AuthorisationFilter</filter-name>
  <filter-class>
    de.fraunhofer.fokus.jic.minishop.AuthorisationFilter
  </filter-class>
  <init-param>
    <param-name>error_page</param-name>
    <param-value>error.jsp</param-value>
  </init-param>
  <init-param>
    <param-name>userid_request_attr</param-name>
    <param-value>userdata</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>AuthorisationFilter</filter-name>
  <url-pattern>/shop.jsp</url-pattern>
</filter-mapping>
```

6.3.2 Displaying Claims in Java Server Pages (JSPs)

After the processing has taken place, the Java Server Page `shop.jsp` can now use standard Expression Language (EL) markup to display the claims that the user sent:

```
<li>First Name: ${userdata.firstName[0]}</li>
<li>Last Name: ${userdata.lastName[0]}</li>
<li>Email: ${userdata.email[0]}</li>
```

Note: Since each claim may occur more than once, array notation is used to access the first claim in the `userdata` session variable.

The class `ClaimUriis` defines standard JavaBean accessor methods for the claim types defined in [ISIPv15], so that the following syntax may be used instead after the class is imported into the JSP:

```
<jsp:useBean
  id="uris"
  class="de.fraunhofer.fokus.jic.identity.ClaimUriis"
  scope="page" />
```

[...]

```
<li>Email: ${userdata[uris.emailAddress][0]}</li>
```

Custom, site-specific claims can be accessed by using the following alternative notation:

```
<li>  
PPID:  
${userdata["http://example.com/cusomclaims/mycustomclaimtype"][0]}  
</li>
```

SETUP AND INSTALLATION

7.1 Getting the Source Code

The primary source code repository for JInfoCard is hosted at [Sourceforge](#). It can be retrieved from the project's Subversion repository by executing the following command:

```
svn co https://informationcard.svn.sourceforge.net/svnroot/informationcard/trunk JInfoCard
```

This will create a folder `JInfoCard` in the current working directory, containing the two subdirectories `framework` and `minishop`. The first contains the JInfoCard framework, a library which can be integrated into any Java-based web application to enable the use of claims-based authentication via information cards. The directory `minishop` contains the sample web application hosted at <https://www.jinfocard.org/> (and <http://www.jinfocard.org/>) which illustrates how the user identity generated by the framework can be integrated into a sample web application.

7.2 Framework:

JInfoCard uses [Maven](#) for the build system. This greatly simplifies dependency handling and deployment, so that once Maven is installed, the framework can be built and installed by issuing the command

```
mvn install
```

from within the `framework` directory. This will download all the dependencies, build the jar file for the framework and install it into the local maven repository, so it is ready to be used as a dependency from other projects, such as the `minishop` example website described below.

Note: If your internet connection requires you to use a proxy, you must instruct Maven to use it. You can do this by specifying the settings on the command line for each command as follows:

```
mvn -Dhttp.proxyHost=proxy.example.com -Dhttp.proxyPort=8080 ...
```

Alternatively, you can edit your Maven configuration to make these settings permanent. Please refer to the Maven documentation at <http://maven.apache.org/guides/mini/guide-proxies.html> for details on how to do this.

7.3 MiniShop:

The Website requires a little more setup to adapt it to your environment. This documentation shows the steps necessary to deploy the sample web application on the [Tomcat 6](#) webapp container. If you're planning to use a different server, please refer to the documentation of your container for the relevant details.

7.3.1 Prerequisites

While Maven handles the dependency tracking for the libraries JInfoCard depends upon, some extra setup is necessary to enable the strong cryptography used by JInfoCard.

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6

Because JInfoCard requires strong cryptography support from the JVM, it requires the “Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6”, available from <http://java.sun.com/javase/downloads/index.jsp> (under “Other Downloads” at the bottom of the page), to be installed. Please refer to the documentation of the JCE policy files for instructions on how to install them into your JVM.

Endorsed libraries

In addition, JInfoCard requires that a current version of the Apache Xalan and Xerces libraries as well as the BouncyCastle JCE provider be endorsed. To do this, run the command

mvn dependency:copy-dependencies

in the `minishop` directory. This downloads all dependencies and places them in the directory `target/dependency`, from which the files listed below must be copied into either the directory `$TOMCAT_HOME/endorsed` (`$TOMCAT_HOME/common/endorsed` for Tomcat version 5.5 and below) or the JRE endorsed directory (usually located at `$JAVA_HOME/lib/endorsed`).

If another web application container is used, please refer to its documentation for the correct location of endorsed libraries.

List of libraries to endorse:

- `bcprov-jdk15-138.jar`
- `resolver.jar`
- `serializer.jar`
- `xalan.jar`
- `xercesImpl.jar`
- `xml-apis.jar`

Basic setup

Copy (or rename) the file `example-web.xml` to `web.xml` in the directory `src/main/webapp/WEB-INF` and edit it to reflect your local setup.

SSL support

If you’re planning to enable SSL support for your deployment, you’ll also need to copy the Keystore (in PKCS#12 or JKS format) containing your site’s certificate and private key into the directory `src/main/resources` and edit the `web.xml` file (see above) to enable the framework to access the private key.

The Tomcat web application server requires some configuration in order to accept SSL connections. The configuration file `$TOMCAT_HOME/conf/server.xml` needs to be adapted to contain the following entry:

```
[...]  
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"  
    maxThreads="150" scheme="https" secure="true"  
    clientAuth="false" sslProtocol="TLS"  
    keystoreFile="/path/to/keystore.p12"  
    keystorePass="s3cr3t"  
    alias="your_certificate_alias" />  
[...]
```

The default `server.xml` file already contains such an entry (commented out), which can be used as a template. A detailed description of all the options and other methods of configuring SSL support are available in the [Tomcat documentation](#).

While it is *strongly* recommended to use an SSL certificate issued by a trusted Certification Authority (CA), it is possible to use the following [OpenSSL](#) commands to create a self-signed keystore *for testing purposes*:

```
# generate self-signed CA cert
openssl req -x509 -new -out cacert.crt -keyout cacert.key -days 99999

#generate private key
openssl genrsa -des3 -out example.org.key 2048

# generate certificate signing request
# when asked for the common name, enter the hostname of the url
# you'll be using for JInfoCard.
openssl req -new -key example.org.key -out example.org.csr

# sign the request
openssl x509 -req -in example.org.csr \
  -CA cacert.crt -CAkey cacert.key -CAcreateserial \
  -out example.org.crt -days 99990

# generate cert bundles
cat cacert.crt example.org.key example.org.crt > example.org.pem
openssl pkcs12 -export -in example.org.pem -out example.org.p12 \
  -name ""
```

The new PKCS#12 keystore can be found in the file `example.org.p12`; the passwords used in the various steps should be used to configure JInfoCard as described above.

Warning: Using a self-signed certificate as described in the preceding paragraph will require all potential users/customers of the site using JInfoCard to add the CA certificate (`cacert.crt`) generated in the first step to their systems' trusted certificate store. Depending on the user's operating system and browser, this can be a fairly involved and error-prone procedure which not all users may be able to accomplish, so, as mentioned above, it is *strongly* recommended to obtain a proper certificate from a trusted CA.

Build the MiniShop

Just execute the command

mvn package

in the directory containing the minishop source. This will again retrieve necessary dependencies (including the framework you built above) and build everything. You can then deploy the generated Web Application, located at `target/minishop2.war`, into your Tomcat installation.

FREQUENTLY ASKED QUESTIONS

We tried to cover most of the questions in the previous chapters. However, we have collected some FAQ which might help you further.

1. Where are InformationCards used?

InformationCards can be used where a registration about user name and password required to login, for example in communities, online banking and in modern eGovernment.

2. Which information are stored on InformationCard?

Personal data are required by users. These data can be different and individually required by the administrator.

3. Where is the personal data stored?

All data about users are stored decentral on the client site.

4. What kinds of InformationCard are available?

There are two types available. On the one hand self issued card, on the other managed card. Self issued cards are generated and managed by the user himself. A managed card is an InformationCard issued by an Identity Provider.

5. Is it possible to have more than one InformationCard?

Yes, users can have severally identities and therewith several InformationCards.

6. How can web sites tell apart different users using Information Cards?

They identify the user by the Private Personal Identifier (PPID). This is an ID that identifies a specific card for a certain *relying party*.

In case of a self issued card, the user's identity selector calculates the PPID as a combination of the relying party certificate and something unique about the card.

For managed cards, the identity selector will provide the *identity provider* with a cryptographic seed which it can use to calculate the PPID.

7. What are certificates and for what are they used?

Digital certificates are used to certify the identity of persons or computers. The function similarly to identification cards such as passports and drivers' licenses. JInfoCards certificate is generated with installation. This happened with an ant script.

8. How to change the server.xml and why?

Please see the section *SSL support* for details.

9. How to set JAVA_HOME?

The JAVA_HOME variable must point at the home directory of JDK installation. On Microsoft Windows systems, it is can be modified throug system properties -> advanced -> environment variables.

10. The JInfoCard log file contains the following error message:

```
GRAVE: error setting up decryption.  
org.apache.xml.security.encryption.XMLEncryptionException: Illegal key size
```

The JCE Unlimited Strength Jurisdiction Policy Files are not installed. Please see the section *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6* for details.

GLOSSARY

Assertion A collection of claims, which are, for example transmitted with SAML

Browser A program for viewing multimedia-content, for browsing

Card Selector same as Identity Selector

Claims A range of assertion which are yet to be.

Identity data Meta data like: name, addresses, phone numbers, date of birth, etc.

Identity Provider The agency (company, website, organization, yourself) that is asserting claims about a subject (person)

Identity Selector Dialog that presents the user's different digital identities and allows the selection of a relevant identity for submission to the Relying Party

Information Card An Information Card is a set of data about yourself that you can send to a website or online service. Like cards in wallet, cards sent with CardSpace present information about you.

Interoperability Allow cross-platform communication between it systems and applications

Keystore (java) class which represents in-memory collection of keys and certificates. Keys and certificates are accessed via aliases.

.NET Framework 3.0 is the name of a .NET-based program interface which develops Microsoft primarily for Windows Vista

Relying Party The agency (website, server, or other party) who will rely on the claims presented by the subject

RTCP RealTime Control Protocol (RTCP)

RTSP RealTime Streaming Protocol (RTSP)

SAML Security Assertion Markup Language based on XML, for exchanging authentication and authorization data between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). The single most important problem that SAML is trying to solve is the Web Browser Single Sign-On (SSO) problem.

Security Token Service Responsible for issue, examination, renewal and depreciation of a token

Self-issued card "Personal cards" are cards that have been self created by the user. The personal information that you enter on a card is stored on client site. To help keep it safe, the information is encrypted

Servlet Is a mix between Server and Applet.

SSO (Single-Sign-on) Is a one off registration. After authentication access to services is allowed, dependent on access privileges

Token (Sign) „One-Time Password“

WCS Windows Card Space (formerly InfoCard)

LICENSING

JInfoCard is free software and falls under the BSD license.

Summarized, according to this license JInfoCard may be redistributed in any way with or without modification in source or binary form under the condition that a copy of the license is included and that the name of the company (Fraunhofer) is not used for selling or promoting the product itself. Finally the license consists of the usual disclaimer about rejecting any warranties on the product and the possible consequences involved in modifying and/or using it.

10.1 Fraunhofer Institute

The Fraunhofer-Gesellschaft, the largest organization for applied research in Europe, sees itself as a service company, offering its various scientific and technological knowhow to private and public organisations and partners. The Fraunhofer-Gesellschaft is made up of 56 institutes staffed by a total of roughly 12,500 employees, with a researched budget of about 1 billion EUR annually.

The Fraunhofer Institute for Open Communication Systems (FOKUS) is an institute of the Fraunhofer Gesellschaft and has developed for 15 years solutions for partners from industry, research and public administrations in areas such as integration of networks, technologies, adaption and management of services, applications as well as measuring and testing of diversified telecommunication systems.

Fraunhofer FOKUS offers in the form of their interoperability laboratory industry partners and public organizations comprehensive technical infrastructure and all the expertise needed to evaluate interoperability of IT systems from different vendors on the organizational, semantic and technical level.

BIBLIOGRAPHY

- [IDABC-EIF] Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens (IDABC): “*The European Interoperability Framework*”. Available from <http://ec.europa.eu/idabc/en/document/3473/5585>
- [EICTA] European Information & Communications Technology Industry Association: “*EICTA Interoperability White Paper*”. Available from http://www.eicta.org/fileadmin/user_upload/document/document1166548285.pdf
- [EL] Sun Microsystems, Inc.: “*JavaServer Pages (TM) v2.0 Syntax Reference*”. Available from <http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html>
- [JSR053] Sun Microsystem, Inc. “*Java (TM) Servlet 2.3 and JavaServer Pages (TM) 1.2 Specifications*”. Available from <http://jcp.org/aboutJava/communityprocess/final/jsr053/>
- [DOM3] Arnaud Le Hors and Philippe Le Hégarret: “*W3C Document Object Model Level 3 Core*” (W3C Recommendation). Available from <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/>
- [DOM3Java] Arnaud Le Hors and Philippe Le Hégarret: “*W3C Document Object Model Level 3 Core*” (W3C Recommendation), Appendix G: “*Java Bindings*”. Available from <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/java-binding.html>
- [ISIPv15] A. Nanda and M. Jones: “*Identity Selector Interoperability Profile V1.5 and companion guides*”. Available from <http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764>

INDEX

Symbols

.NET Framework 3.0, 23

A

Assertion, 23

B

Browser, 23

C

Card Selector, 23

Claims, 23

I

Identity data, 23

Identity Provider, 23

Identity Selector, 23

Information Card, 23

Interoperability, 23

K

Keystore, 23

R

Relying Party, 23

RTCP, 23

RTSP, 23

S

SAML, 23

Security Token Service, 23

Self-issued card, 23

Servlet, 23

SSO (Single-Sign-on), 23

T

Token, 23

W

WCS, 23