



Meteor 4 Token Provider Implementation Guide

Version 1.2

Contributors:
Richard Allen
John Lazos

Revision History

Version	Revision Description	Author	Date
1.0	Initial document.	John Lazos	11/23/2011
1.1	Revised for implementing TokenProvider without building Meteor from source	John Lazos	12/01/2011
1.2	Fixed package name for TokenProvider interface	John Lazos	12/06/2011

Contents

Revision History	2
Contents.....	3
1. Purpose	4
2. Requirements.....	4
2.1. Developer Skill Set	4
2.2. Developer Tools	4
3. Overview.....	4
4. Interface Descriptions	6
4.1. TokenProvider interface.....	6
4.2. SecurityToken interface.....	7
5. Implementation Instructions.....	9
Appendix	10
A. Token Provider Interface	10
B. SecurityToken interface.....	11
C. Sample Token Provider Implementation.....	15
D. Shibboleth Token Provider Implementation.....	17

1. Purpose

This document explains how to implement a Token Provider in the UI provider.

2. Requirements

2.1. *Developer Skill Set*

Developers of custom token providers must have the following skill set:

- Knowledge of the Java programming language
- Knowledge of the authentication framework that the Token Provider will interact with

2.2. *Developer Tools*

- Java Development Kit (1.5 or greater)
- Integrated Development Environment (e.g. Eclipse IDE)

2.3. *Libraries*

To implement a TokenProvider, the following libraries are required on the build class path:

- Meteor Common Library (meteorlib-4.0.0.jar)
- Meteor SAML Library (meteorsaml-4.0.0.jar)
- Java Servlet API (servlet-api.jar, version 2.5+)

3. Overview

The Meteor UI provider restricts user access by requiring a Security Token be available on each request. If the Security Token is invalid or not available, the user is denied access.

A Token Provider implementation provides the user's Security Token to the UI provider, which passes it on to the Meteor Access Provider software to query the Meteor network. A Token Provider implementation is a Java class that implements the `TokenProvider` interface.

Token Provider implementations are specific to whatever authentication mechanism is used to secure the UI provider from unauthorized access. For example, if user access is restricted using Shibboleth, the UI provider would be configured to use the bundled `ShibbolethTokenProvider`. For custom authentication mechanisms, a custom TokenProvider implementation will have to be implemented.

To implement a Token Provider, one must have knowledge of the Java programming language and the authentication mechanism the UI provider must interact with.

Two Token Provider implementations are provided with the UI provider WAR: `ShibbolethTokenProvider` and `SampleTokenProvider` (located in the `org.meteornetwork.meteor.provider.ui.token` package in the UI provider WAR).

These two implementations can be used as examples for developing custom Token Providers. (See Appendix)

4. Interface Descriptions

4.1. *TokenProvider* interface

The `TokenProvider` interface is packaged in the Meteor Common library:

```
org.meteornetwork.meteor.common.abstraction.token.TokenProvider
```

The `TokenProvider` interface requires the `getSecurityToken()` method be implemented:

```
SecurityToken getSecurityToken(HttpServletRequest request) throws SecurityTokenException;
```

This method returns an instance of `SecurityToken`, which must be populated with Meteor attributes. (See 3.2 `SecurityToken` interface.) The one parameter is of type `javax.servlet.http.HttpServletRequest` and contains all request info submitted by the user.

If a Security Token cannot be returned, a `SecurityTokenException` must be thrown. By default, this will cause the UI provider to display the Access Denied page to the user. Optionally, this exception can be thrown specifying a Cause Code parameter. The Cause Code parameter is used to specify the reason the Security Token could not be created. For example, if the user's session has expired, throwing the `SecurityTokenException` with the `SESSION_EXPIRED` cause code will cause the UI provider to display the Session Expired page to the user instead of Access Denied.

The following Cause Codes can be used with `SecurityTokenException`:

Cause Code	Reasons to throw	User's Result
<code>ACCESS_DENIED</code>	The Security Token cannot be created because the user's request did not contain sufficient information to establish the user's authenticity.	User is presented with the Access Denied page
<code>SESSION_EXPIRED</code>	The user's request establishes the user's authenticity, but the information is no longer valid. (e.g. the user's SAML assertion is expired)	User is presented with the Session Expired page

`SecurityToken`, `SecurityTokenException`, and the `CauseCode` enumeration are all packaged in the Meteor SAML library:

```
org.meteornetwork.meteor.saml.SecurityToken;  
org.meteornetwork.meteor.saml.exception.SecurityTokenException;  
org.meteornetwork.meteor.saml.exception.SecurityTokenException.CauseCode;
```

Source code is available in the Meteor CVS repository and in the Appendix.

4.2. SecurityToken interface

The `SecurityToken` interface specifies methods to get and set SAML and Meteor attributes. `SecurityTokenImpl` (`org.meteornetwork.meteor.saml`) represents the current Meteor Security Token implementation and should be used in all custom Token Provider implementations.

The following properties may be set on a Security Token implementation. Note: some attributes are required.

Property	Description	Required
SAML Properties		
Assertion Id	The ID attribute of the assertion element. This property is overridden with a generated value on a call to <code>toXML()</code> or <code>toXMLString()</code> .	Yes
Issuer	The issuer of this assertion. (see SAML 2 specification)	No
Subject	The Subject element of the assertion. (see SAML 2 specification)	Yes
Subject Locality IP Address	Default value is the result of call to <code>InetAddress.getLocalHost().getHostAddress()</code> , or null if <code>UnknownHostException</code> is thrown.	No
Subject Locality DNS Address	Default value is the result of call to <code>InetAddress.getLocalHost().getHostName()</code> , or null if <code>UnknownHostException</code> is thrown.	No
Conditions Not Before	Earliest time the assertion is valid. Defaults to current time when <code>toXML()</code> or <code>toXMLString()</code> is called	No
Conditions Not On Or After	The time when the assertion is no longer valid. Defaults to the value set using <code>setConditionsNotBefore()</code> plus 1 hour.	No
Meteor Attributes		
User Handle	The name of the authenticated user.	Yes
Role	The user's role. (Enumeration of type <code>org.meteornetwork.meteor.saml.Role</code>)	Yes
Authentication Process ID	The ID of the process used to authenticate the user.	Yes
Level	The authentication level.	Yes
Organization Id	For FAA role only. The ID of the organization the FAA belongs to.	No
Organization Id Type	For FAA role only. The type of ID that the organization ID is. (e.g. OPEID)	No
Organization Type	For FAA role only. The type of organization the FAA user belongs to. (e.g. School)	No
SSN	For BORROWER role only. The SSN of the user.	No
Lender	For LENDER role only. The OPEID of the Lender's organization.	No

`SecurityTokenImpl` contains the static `fromXML()` method to create a Security Token from a SAML 2.0 assertion XML string.

5. Implementation Instructions

1. Download required libraries:
 - Meteor Common library (meteorlib-4.0.0.jar)
 - Meteor SAML library (meteorsaml-4.0.0.jar)
 - Java Servlet API (servlet-api.jar, version 2.5+)

The Meteor libraries can be downloaded from the [Meteor website](http://www.meteornetwork.org) (<http://www.meteornetwork.org>). The Java Servlet API is provided as a shared library with most application servers.

NOTE: The Java Servlet API is only required to compile. It should NOT be packaged with your library, since it is provided by most application servers.

2. Create a new Java project that compiles into a JAR library. Add a class that implements

```
org.meteornetwork.meteor.common.abstraction.token.TokenProvider
```

Add this class to a package and note the fully qualified class name. It will be needed to configure the UI provider.

3. Implement the `getSecurityToken()` method. Use section 4. Interface Descriptions and the example implementations as references.
4. Compile the project and package it in a JAR.
5. Edit `uiprovider.properties` (available in `WEB-INF/lib` in the UI provider WAR). Change `uiprovider.tokenproviderclass` to be the fully qualified class name of the Token Provider implementation.
6. Deploy the UI provider software to the application server. Put the compiled JAR with the `TokenProvider` implementation in the UI provider's `WEB-INF/lib` directory.
7. Start the application server. The UI provider will initialize the custom `TokenProvider` implementation and use it to accept authenticated users.

Appendix

A. Token Provider Interface

```
/*
 * Copyright 2002 National Student Clearinghouse
 *
 * This code is part of the Meteor system as defined and specified
 * by the National Student Clearinghouse and the Meteor Sponsors.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library 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 library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
package org.meteornetwork.meteor.common.abstraction.token;

import javax.servlet.http.HttpServletRequest;

import org.meteornetwork.meteor.saml.SecurityToken;
import org.meteornetwork.meteor.saml.exception.SecurityTokenException;

/**
 * Provides a Meteor Security Token for the authenticated user. Implementations
 * create security tokens based on whatever login system is employed (Sample
 * logon, Shibboleth, OpenAM etc)
 *
 * @author jlazos
 */
public interface TokenProvider {

    /**
     * Produces a Meteor Security Token for the authenticated user.
     *
     * @return the token representing the authenticated user
     * @throws SecurityTokenException
     *         thrown when security token cant be returned for some reason.
     *         The cause code parameter in the constructor of
     *         SecurityTokenException affects the message returned to the
     *         user by the UI provider. For example, if the cause code is
     *         ACCESS_DENIED (default), the user is presented with the
     *         Access Denied page. If the cause code is SESSION_EXPIRED, on
     *         the other hand, the user is presented with the more innocent
     *         "Please login again" page. Consult the SecurityTokenException
     *         javadoc for further details.
     */
    SecurityToken getSecurityToken(HttpServletRequest request) throws SecurityTokenException;
}

```

B. SecurityToken interface

```
/*
 * Copyright 2002 National Student Clearinghouse
 *
 * This code is part of the Meteor system as defined and specified
 * by the National Student Clearinghouse and the Meteor Sponsors.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library 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 library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
package org.meteornetwork.meteor.saml;

import org.joda.time.DateTime;
import org.meteornetwork.meteor.saml.exception.SecurityTokenException;
import org.w3c.dom.Document;

/**
 * Creates or parses a SAML 2.0 assertion. The getter/setter methods of this
 * class are used to access parts of the assertion.
 *
 * When using an implementation of this interface to create a SAML 2.0
 * assertion, the programmer only needs to consider setting the following
 * properties:
 *
 * @author jlazos
 */
public interface SecurityToken {

    /**
     * Creates a new SAML 2.0 assertion using the parameters of this security
     * token (except the assertion ID, which is generated anew)
     *
     * @return DOM representation of the SAML 2.0 assertion
     * @throws SecurityTokenException
     *         wraps any exceptions that occur while processing
     */
    Document toXML() throws SecurityTokenException;

    /**
     * Creates a new SAML 2.0 assertion using the parameters of this security
     * token (except the assertion ID, which is generated anew)
     *
     * @return XML string representation of the SAML 2.0 assertion
     * @throws SecurityTokenException
     *         wraps any exceptions that occur while processing
     */
    String toXMLString() throws SecurityTokenException;

}
```

```

    * Validates assertion has not yet expired
    *
    * @return true if assertion has not expired, false otherwise
    */
    boolean validateConditions();

    String getAssertionId();

    /**
     * The ID attribute of the assertion element. This property is overridden
     * with a generated value on a call to toXML() or toXMLString().
     *
     * @param id
     */
    void setAssertionId(String id);

    String getIssuer();

    /**
     * Optional. The issuer of this assertion.
     *
     * @param issuer
     */
    void setIssuer(String issuer);

    String getSubjectName();

    /**
     * The Subject element of the assertion.
     *
     * @param subjectName
     */
    void setSubjectName(String subjectName);

    String getSubjectLocalityIpAddress();

    /**
     * Optional. Default value is the result of call to
     * InetAddress.getLocalHost().getHostAddress(), or null if
     * UnknownHostException is thrown.
     *
     * @param subjectLocalityIpAddress
     */
    void setSubjectLocalityIpAddress(String subjectLocalityIpAddress);

    String getSubjectLocalityDnsAddress();

    /**
     * Optional. Default value is the result of call to
     * InetAddress.getLocalHost().getHostName(), or null if UnknownHostException
     * is thrown.
     *
     * @param subjectLocalityDnsAddress
     */
    void setSubjectLocalityDnsAddress(String subjectLocalityDnsAddress);

    DateTime getConditionsNotBefore();

    /**
     * Optional. Earliest time the assertion is valid. Defaults to current time
     * when toXML() or toXMLString() is called
     *
     * @param dateTime
     */
    void setConditionsNotBefore(DateTime dateTime);

    DateTime getConditionsNotOnOrAfter();

    /**
     * Optional. The time when the assertion is no longer valid. Defaults to the
     * value set using setConditionsNotBefore() plus 1 hour.

```

```

*
* @param dateTime
*/
void setConditionsNotOnOrAfter(DateTime dateTime);

String getOrganizationId();

/**
 * For FAA role only. The ID of the organization the FAA belongs to.
 *
 * @param organizationId
 */
void setOrganizationId(String organizationId);

String getOrganizationIdType();

/**
 * For FAA role only. The type of organization ID (e.g. OPEID)
 *
 * @param organizationIdType
 */
void setOrganizationIdType(String organizationIdType);

String getOrganizationType();

/**
 * For FAA role only. They type of organization (e.g. SCHOOL)
 *
 * @param organizationType
 */
void setOrganizationType(String organizationType);

String getAuthenticationProcessId();

/**
 * Required. The identifier of the process used to authenticate the user.
 *
 * @param authenticationProcessId
 */
void setAuthenticationProcessId(String authenticationProcessId);

Integer getLevel();

/**
 * Required. The authentication level of the user authentication process
 *
 * @param level
 */
void setLevel(Integer level);

/**
 * Required.
 *
 * @return
 */
String getUserHandle();

void setUserHandle(String userHandle);

Role getRole();

/**
 * Required. User role
 *
 * @param role
 */
void setRole(Role role);

String getSsn();

/**

```

```
    * Optional. Sets the SSN the user is authorized to view. Only applicable
    * for the BORROWER role.
    */
    void setSsn(String ssn);

    String getLender();

    /**
     * Optional. Sets the lender OPEID. Only applicable for the LENDER role.
     */
    void setLender(String lender);

    /**
     * Get all meteor-specific attributes in one bean
     *
     * @return
     */
    TokenAttributes getMeteorAttributes();

    /**
     * Set all meteor-specific attributes with one bean
     *
     * @param attributes
     */
    void setMeteorAttributes(TokenAttributes attributes);
}
```

C. Sample Token Provider Implementation

```

/*****
 * Copyright 2002 National Student Clearinghouse
 *
 * This code is part of the Meteor system as defined and specified
 * by the National Student Clearinghouse and the Meteor Sponsors.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library 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 library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *****/
package org.meteornetwork.meteor.provider.ui.token;

import java.io.IOException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;

import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.apache.cxf.helpers.IOUtils;
import org.meteornetwork.meteor.saml.SecurityToken;
import org.meteornetwork.meteor.saml.SecurityTokenImpl;
import org.meteornetwork.meteor.saml.exception.SecurityTokenException;
import org.meteornetwork.meteor.saml.exception.SecurityTokenException.CauseCode;

/**
 * Works with the Meteor example login provider
 *
 * @author jlazos
 */
public class SampleTokenProvider implements TokenProvider {

    private static final Logger LOG = LoggerFactory.getLogger(SampleTokenProvider.class);

    @Override
    public SecurityToken getSecurityToken(HttpServletRequest request) throws SecurityTokenException {

        String tokenId = request.getParameter("tokenId");
        String tokenString = null;

        if (tokenId == null || "".equals(tokenId)) {
            tokenString = (String)
                request.getSession().getAttribute("SampleSecurityToken");
            if (tokenString == null || "".equals(tokenString)) {
                throw new SecurityTokenException(CauseCode.SESSION_EXPIRED);
            }
        } else {
            try {
                tokenString = getTokenFromURL(request.getParameter("url"),
                    tokenId);
                request.getSession().setAttribute("SampleSecurityToken",
                    tokenString);
            }
        }
    }
}

```

```

        } catch (IOException e) {
            LOG.debug("Could not get security token from sample login
                provider", e);
            throw new SecurityTokenException();
        }
    }

    SecurityToken token;
    try {
        token = SecurityTokenImpl.fromXML(tokenString);
    } catch (Exception e) {
        throw new SecurityTokenException(e);
    }

    if (!token.validateConditions()) {
        LOG.debug("Token expired");
        throw new SecurityTokenException();
    }
    return token;
}

private String getTokenFromURL(String urlStr, String tokenId) throws IOException {
    String charset = "UTF-8";

    String query = String.format("artifactId=%s", URLEncoder.encode(tokenId, "UTF-
        8"));
    URLConnection connection = new URL(urlStr + "?" + query).openConnection();
    connection.setRequestProperty("Accept-Charset", charset);

    return IOUtils.readStringFromStream(connection.getInputStream());
}
}

```

D. Shibboleth Token Provider Implementation

```
/*
 * Copyright 2002 National Student Clearinghouse
 *
 * This code is part of the Meteor system as defined and specified
 * by the National Student Clearinghouse and the Meteor Sponsors.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library 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 library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
package org.meteornetwork.meteor.provider.ui.token;

import javax.servlet.http.HttpServletRequest;

import org.meteornetwork.meteor.saml.Role;
import org.meteornetwork.meteor.saml.SecurityToken;
import org.meteornetwork.meteor.saml.SecurityTokenImpl;
import org.meteornetwork.meteor.saml.exception.SecurityTokenException;

public class ShibbolethTokenProvider implements TokenProvider {

    @Override
    public SecurityToken getSecurityToken(HttpServletRequest request) throws
        SecurityTokenException {

        SecurityToken token = new SecurityTokenImpl();
        token.setAssertionId(getAttribute(request, "Shib-Session-ID"));
        token.setIssuer(getAttribute(request, "Shib-Identity-Provider"));

        token.setUserHandle(getRequiredAttribute(request, "UserHandle"));
        token.setOrganizationId(getAttribute(request, "OrganizationId"));
        token.setOrganizationIdType(getAttribute(request, "OrganizationIdType"));
        token.setOrganizationType(getAttribute(request, "OrganizationType"));
        token.setAuthenticationProcessId(getRequiredAttribute(request,
            "AuthenticationProcessId"));
        token.setLevel(Integer.valueOf(getRequiredAttribute(request, "Level")));
        token.setRole(Role.valueOfName(getRequiredAttribute(request, "Role")));
        if (token.getRole() == null) {
            throw new SecurityTokenException("Could not create SecurityToken from
                Shibboleth attributes. Role " + getRequiredAttribute(request,
                    "Role") + " is not a valid Meteor role.");
        }
        token.setSsn(getAttribute(request, "SSN"));
        token.setLender(getAttribute(request, "Lender"));

        return token;
    }

    private String getAttribute(HttpServletRequest request, String attributeName) {
        return (String) request.getAttribute(attributeName);
    }

    private String getRequiredAttribute(HttpServletRequest request, String attributeName)
        throws SecurityTokenException {

        String attrValue = getAttribute(request, attributeName);
    }
}
```

```
        if (attrValue == null) {
            throw new SecurityTokenException("Could not create SecurityToken from
                Shibboleth attributes. " + attributeName + " is required.");
        }
        return attrValue;
    }
}
```