

# Creating a Security Check

fork and edit tutorial (<https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/authentication-and-security/creating-a-security-check/index.md>) | report issue (<https://github.ibm.com/MFPSamples/DevCenter/issues/new>)

## Overview

A `SecurityCheck` is an object responsible for obtaining credentials from a client and validate them.

Security checks are defined inside **Java adapters**. Any Java adapter can theoretically define a `SecurityCheck`. An adapter can either be a *resource* adapter (meaning it serves resources/content to send to the client), a *SecurityCheck* adapter, or **both**.

**Prerequisites:** Familiarize yourself with the MobileFirst Platform Foundation authentication framework before continuing.

Read the Authentication concepts ([../authentication-concepts/](#)) tutorial.

Jump to:

- Defining a `SecurityCheck`
- `SecurityCheck` Implementation
- `SecurityCheck` Configuration
- Predefined Security Checks
- Tutorials to follow next

## Defining a SecurityCheck

In the Java adapter's `adapter.xml` file, add an XML element called `securityCheckDefinition`. For example:

```
<securityCheckDefinition name="sample" class="com.sample.sampleSecurityCheck">
  <property name="successExpirationSec" defaultValue="60"/>
  <property name="failureExpirationSec" defaultValue="60"/>
  <property name="maxAttempts" defaultValue="3"/>
</securityCheckDefinition>
```

- The `name` attribute will be the name of your `SecurityCheck`
- The `class` attribute specifies the implementation of the `SecurityCheck`
- Some `SecurityChecks` can be configured with a list of `property` elements.

## SecurityCheck Implementation

The class file of the `SecurityCheck` is where the security check's implementation takes place. The implementation should extend one of the provided base classes, below. The parent class you choose will determine the balance between customization and simplicity.

## SecurityCheck

`SecurityCheck` is a Java **interface**, defining the minimum required methods to represent the server-side state of a security check. Using this interface alone does not provide any implementation code and it is the sole responsibility of the implementor to handle each scenario.

## SecurityCheckWithExternalization

This abstract class implements a basic version of the `SecurityCheck` interface.

It provides, among other options: externalization as JSON, inactivity timeout, expiration countdown and more.

Subclassing this class leaves a lot of flexibility in your Security Check implementation.

Learn more in the `SecurityCheckWithExternalization` user documentation topic.

## SecurityCheckWithAttempts

This abstract class extends `SecurityCheckWithExternalization` and implements most of its methods to simplify usage. Two methods are required to be implemented: `validateCredentials` and `createChallenge`.

The `SecurityCheckWithAttempts` class is meant for simple flows to need to validate arbitrary credentials in order to grant access to a resource. Also provided is a built-in capability to block access after a set number of attempts.

Learn more in the `SecurityCheckWithAttempts` (`../security-check-with-attempts`) tutorial.

## SecurityCheckWithUserAuthentication

This abstract class extends `SecurityCheckWithAttempts` and therefore inherits all of its features.

In addition, the `SecurityCheckWithUserAuthentication` class provides the MobileFirst framework an `AuthenticatedUser` object which represents the logged-in user. Methods that are required to be implemented are `createUser`, `validateCredentials` and `createChallenge`.

Also provided is a built-in capability to optionally enable a "Remember Me" login behavior.

Learn more in the `SecurityCheckWithUserAuthentication` (`../security-check-with-user-authentication`) tutorial.

## SecurityCheck Configuration

Each `SecurityCheck` implementation class can use a `SecurityCheckConfiguration` class that defines properties available for that `SecurityCheck`. Each base `SecurityCheck` class comes with a matching `SecurityCheckConfiguration` class. You can create your own implementation that extends one of the base `SecurityCheckConfiguration` classes and use it for your custom `SecurityCheck`.

For example, `SecurityCheckWithUserAuthentication`'s `createConfiguration` method returns an instance of `SecurityCheckWithAuthenticationConfig`.

```
public abstract class SecurityCheckWithUserAuthentication extends SecurityCheckWithAttempts {
    @Override
    public SecurityCheckConfiguration createConfiguration(Properties properties) {
        return new SecurityCheckWithAuthenticationConfig(properties);
    }
}
```

`SecurityCheckWithAuthenticationConfig` enables a property called `rememberMeDurationSec` with a default of `0`.

```
public class SecurityCheckWithAuthenticationConfig extends SecurityCheckWithAttemptsConfig {

    public int rememberMeDurationSec;

    public SecurityCheckWithAuthenticationConfig(Properties properties) {
        super(properties);
        rememberMeDurationSec = getIntProperty("rememberMeDurationSec", properties, 0);
    }

}
```

These properties can be configured at several levels:

## adapter.xml

In the Java adapter's adapter.xml file, inside `<securityCheckDefinition>`, you can add one or more `<property>` elements.

The `<property>` element takes the following attributes:

- **name**: The name of the property, as defined in the configuration class.
- **defaultValue**: Overrides the default value defined in the configuration class.
- **displayName**: A friendly name to be displayed in the console.

Example:

```
<property name="maxAttempts" defaultValue="3" displayName="How many attempts are allowed"/>
```

## MobileFirst Operations Console - Adapter

In the MobileFirst Console, in the "Security Check" tab of your adapter, you will be able change the value of any property defined in the adapter.xml.

Note that **only** the properties defined in adapter.xml appear on this screen; properties defined in the configuration class won't appear here automatically.

MobileFirst Operations Console

Hello, admin

<

Home > mfp > PinCodeAttempts

Actions

PinCodeAttempts

PinCodeAttempts

Configurations

Resources

Security Check

Configuration Files

Security Checks

PinCodeAttempts

The valid PIN code \*

1234

Default Value: 1234

How many attempts are allowed \*

3

Default Value: 3

How long before the client can try again (seconds) \*

60

Default Value: 60

How long is a successful state valid for (seconds) \*

60

Default Value: 60

Save

Cancel

Restore Default Values

## MobileFirst Operations Console - Application

Property values can also be overridden at the application level. In your MobileFirst Console, in the "Security" tab of your application, under the "Security Check Configurations" section, you can modify the values defined in each Security Check available.



## Predefined Security Checks

Also available are these predefined security checks:

- Application Authenticity (../application-authenticity/)
- Direct Update (../using-the-mfpf-sdk/direct-update)
- LTPA

## Tutorials to follow next

Continue reading about security checks in the following tutorials:

- Implementing the SecurityCheckWithAttempts (../security-check-with-attempts/)

- [Implementing the SecurityCheckWithUserAuthentication \(../security-check-with-user-authentication/\)](#)

[Learn about additional MobileFirst Platform Foundation authentication and security features \(../\).](#)