

# Step Up Authentication

## Overview

Resources can be protected by several security checks. In this case, the MobileFirst Server will send all the relevant challenges simultaneously to the application.

A security check may be dependent on another security check, so it is important to be able to control when the challenges will be sent.

For example, in this tutorial we'll describe an application that has two resources protected by a username and password where the second resource also requires an additional PIN code.

**Prerequisite:** Read the `CredentialsValidationSecurityCheck` (`../credentials-validation`) and `UserAuthenticationSecurityCheck` (`../user-authentication`) tutorials before continuing.

Jump to:

- Referencing a Security Check
- State Machine
- The Authorize Method
- Challenge Handlers
- Sample Applications

## Referencing a Security Check

Create two security checks: `StepUpPinCode` and `StepUpUserLogin`. Their initial implementation is the same as described in the `Credentials Validation` (`../credentials-validation/security-check/`) and `User Authentication` (`../user-authentication/security-check/`) tutorials.

In this tutorial's example, `StepUpPinCode` **depends on** `StepUpUserLogin`. The user should only be asked to enter a PIN code if a successful login to `StepUpUserLogin` previously happened. To do so, `StepUpPinCode` needs to be able to **reference** the `StepUpUserLogin` class.

The MobileFirst Foundation framework provides an annotation to inject a reference. In your `StepUpPinCode` class, at the class-level, add:

```
@SecurityCheckReference
private transient StepUpUserLogin userLogin;
```

**❗ Important:** Both security check's implementations need to be bundled inside the same adapter.

To resolve this reference, the framework looks up for a security check with the appropriate class, and injects its reference into the dependent security check.

If there are more than one security checks of the same class, the annotation has an optional `name` parameter that can be used to specify the unique name of the referred check.

## State machine

All classes that extend `CredentialsValidationSecurityCheck` (which includes both `StepUpPinCode` and `StepUpUserLogin`) inherit a simple state machine. At any given moment, the security check can be in one of these states:

- `STATE_ATTEMPTING`: A challenge has been sent and the security check awaits to the client response. The attempt count is maintained during this state.
- `STATE_SUCCESS`: The credentials have been successfully validated.
- `STATE_BLOCKED`: The maximum number of attempts has been reached and the check is in locked state.

The current state can be obtained using the inherited `getState()` method.

In `StepUpUserLogin`, add a convenience method to check if the user is currently logged-in. This method is later used, below.

```
public boolean isLoggedIn(){
    return this.getState().equals(STATE_SUCCESS);
}
```

## The authorize Method

The `SecurityCheck` interface defines a method called `authorize`. This method is responsible for implementing the main logic of the security check, such as sending a challenge or validating the request. The class `CredentialsValidationSecurityCheck`, which `StepUpPinCode` extends, already includes an implementation for this method. However in our case we want to check the state of `StepUpUserLogin` before starting the default behavior of the `authorize` method.

To do so, **override** the `authorize` method:

```
@Override
public void authorize(Set<String> scope, Map<String, Object> credentials, HttpServletRequest request,
AuthorizationResponse response) {
    if(userLogin.isLoggedIn()){
        super.authorize(scope, credentials, request, response);
    }
}
```

This implementation checks the current state of the `StepUpUserLogin` reference:

- If the state is `STATE_SUCCESS` (meaning the user is logged in), we continue the normal flow of the security check.
- If `StepUpUserLogin` is in any other state, nothing is done (no challenge will be sent, no success nor failure).

Assuming the resource is protected by **both** `StepUpPinCode` and `StepUpUserLogin`, this flow makes sure that the user is logged in before we ask for the secondary credential (PIN code). The client will never receive both challenges at the same time, even though both security checks are activated.

Alternatively, if the resource is protected **only** by `StepUpPinCode` (meaning the framework will only activate this security check), you can change the `authorize` implementation to trigger `StepUpUserLogin` manually:

```

@Override
public void authorize(Set<String> scope, Map<String, Object> credentials, HttpServletRequest request,
AuthorizationResponse response) {
    if(userLogin.isLoggedIn()){
        //If StepUpUserLogin is successful, continue the normal processing of StepUpPinCode
        super.authorize(scope, credentials, request, response);
    } else {
        //In any other case, process StepUpUserLogin instead.
        userLogin.authorize(scope, credentials, request, response);
    }
}

```

## Retrieve current user

In `StepUpPinCode`, we are interested in knowing the current user's ID so we can lookup this user's PIN code in some database.

In `StepUpUserLogin`, add the following method to obtain the current user from the **authorization context**:

```

public AuthenticatedUser getUser(){
    return authorizationContext.getActiveUser();
}

```

In `StepUpPinCode` you can then use `userLogin.getUser()` to get the current user from the `StepUpUserLogin` security check, and check the valid PIN code for this specific user:

```

@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
    //Get the correct PIN code from the database
    User user = userManager.getUser(userLogin.getUser().getId());

    if(credentials!=null && credentials.containsKey(PINCODE_FIELD)){
        String pinCode = credentials.get(PINCODE_FIELD).toString();

        if(pinCode.equals(user.getPinCode())){
            errorMsg = null;
            return true;
        }
        else{
            errorMsg = "Wrong credentials. Hint: " + user.getPinCode();
        }
    }
    return false;
}

```

## Challenge Handlers

In the client-side, there are no special APIs to handle multiple steps. Rather, each challenge handler should handle its own challenge. In this example you need to register two separate challenge handlers: One to handle challenges from `StepUpUserLogin` and one to handle challenges from `StepUpUserLogin`.

# Sample Applications

## Security check

The `StepUpUserLogin` and `StepUpPinCode` security checks are available in the SecurityChecks project under the StepUp Maven project. Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/SecurityCheckAdapters/tree/release80>) the Security Checks Maven project.

## Applications

Sample applications are available for iOS (Swift), Android, Windows 8.1/10, Cordova and Web.

- Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/StepUpCordova/tree/release80>) the Cordova project.
- Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/StepUpSwift/tree/release80>) the iOS Swift project.
- Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/StepUpAndroid/tree/release80>) the Android project.
- Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/StepUpWin8/tree/release80>) the Windows 8.1 project.
- Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/StepUpWin10/tree/release80>) the Windows 10 project.
- Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/StepUpWeb/tree/release80>) the Web app project.

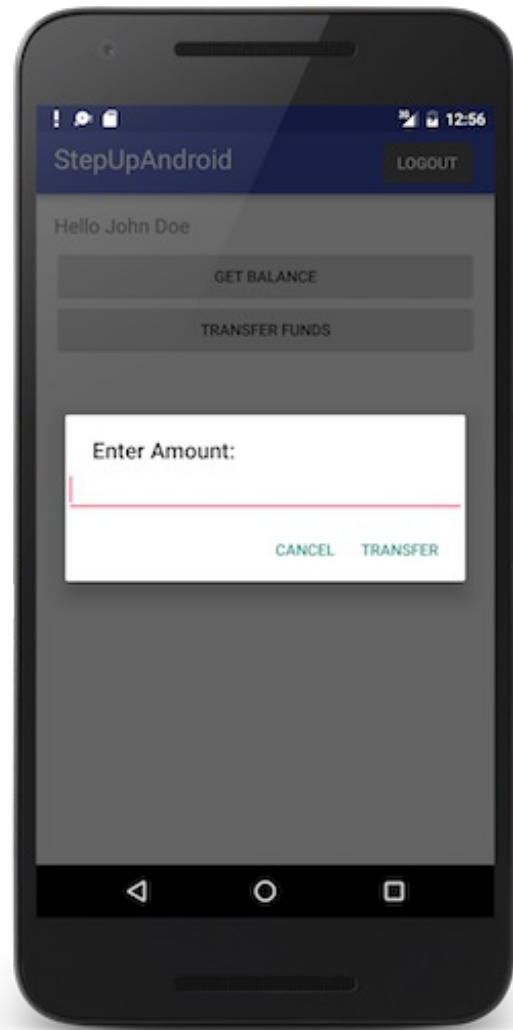
## Sample usage

**Users list:** See the `UserManager.java` file for a list of valid users.

## Deploy adapters

- Use either Maven, MobileFirst CLI or your IDE of choice to build and deploy (`../../adapters/creating-adapters/`) the available **StepUp** and **ResourceAdapter** adapters.

## Register applications



## Cordova

From a **Command-line** window, navigate to the project's root folder and:

- Add a platform by running the `cordova platform add` command.
- Registering the application: `mfpdev app register`.

## Native

From a **Command-line** window, navigate to the application's root folder and run the command: `mfpdev app register`.

## Web

Make sure you have Node.js installed.

Register the application in the MobileFirst Operations Console.

## Scope mapping

In the **MobileFirst Operations Console → StepUpWeb → Security**, map the following scopes:

- `accessRestricted` to `StepUpUserLogin`
- `transferPrivilege` to both `StepUpUserLogin` and `StepUpPinCode`

Alternatively, from the **Command-line**, navigate to the project's root folder and run the command: `mfpdev app push`.

### Note for web sample:

Start the reverse proxy by running the commands: `npm install` followed by: `npm start`.

In a browser, load the URL `http://localhost:9081/sampleapp` (`http://localhost:9081/sampleapp`).