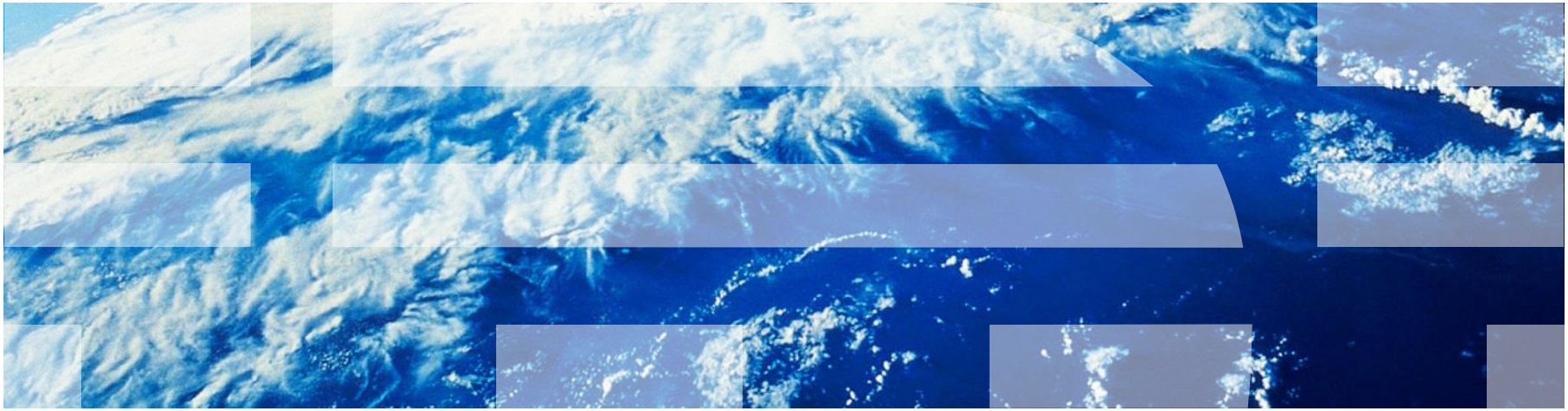
IBM Worklight Foundation V6.2.0 Getting Started

Client X.509 Certificate Authentication and User Enrollment



20 June 2014

Trademarks

- IBM, the IBM logo, ibm.com, WebSphere, and Worklight are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
- Other company products or service names may be trademarks or service marks of others.
- This document may not be reproduced in whole or in part without the prior written permission of IBM.

About IBM®

- See http://www.ibm.com/ibm/us/en/

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Overview (1 of 3)

- The X.509 User Certificate Authentication feature is a newly introduced user realm that establishes user identity with an X.509 client certificate.
- The user identity is established for a particular user on a specific device and application.
- This feature provides SSL client-side certificate authentication and user enrollment capabilities.
- SSL client-side certificate authentication consists of establishing a two-way SSL handshake between a Worklight® client and server, which in turn, enables the client and server both to present their identities and therefore establish mutual trust through the SSL/TLS protocol.

Overview (2 of 3)

- You can enroll new users to the Worklight Mobile Application Management system and your PKI of choice with the user enrollment capabilities.
- User enrollment also provisions the device with the necessary X.509 user credentials for subsequent use.
- A basic embedded PKI is provided with this feature that is meant to get you started quickly for educational and non-production environments only.
- For production environments, this feature makes it easy to integrate with your existing PKI.
  - You can use either the PKI Bridge Java™ interface or built-in Worklight adapters to delegate certificate management functions down to an external PKI system.

Overview (3 of 3)

- In this module, you learn how to enable and configure the User Certificate Authentication user realm.
- This module also shows you:
  - How to use the embedded PKI that is provided by Worklight
- For production environments, this feature makes it easy to integrate with your existing PKI.
  - You can use either the PKI Bridge Java interface or built-in Worklight adapters to delegate certificate management functions down to an external PKI system.

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Prerequisites

- You must have a general understanding of Worklight user realms and adapters. For more information about the relevant modules for form-based authentication and the HTTP-adapter, see http://ibm.biz/knowctr#SSZH4A_6.2.0/com.ibm.worklight.getstart.doc/start/c_gettingstarted.html

- It is assumed that you follow these instructions by using an application that currently supports form-based authentication.
  - The form-based authentication module uses non-validating login modules. These login modules are not recommended for production environments.
  - Use other user authentication realms, like WASLTPA in production.

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Understanding how user certificate authentication works (1 of 4)

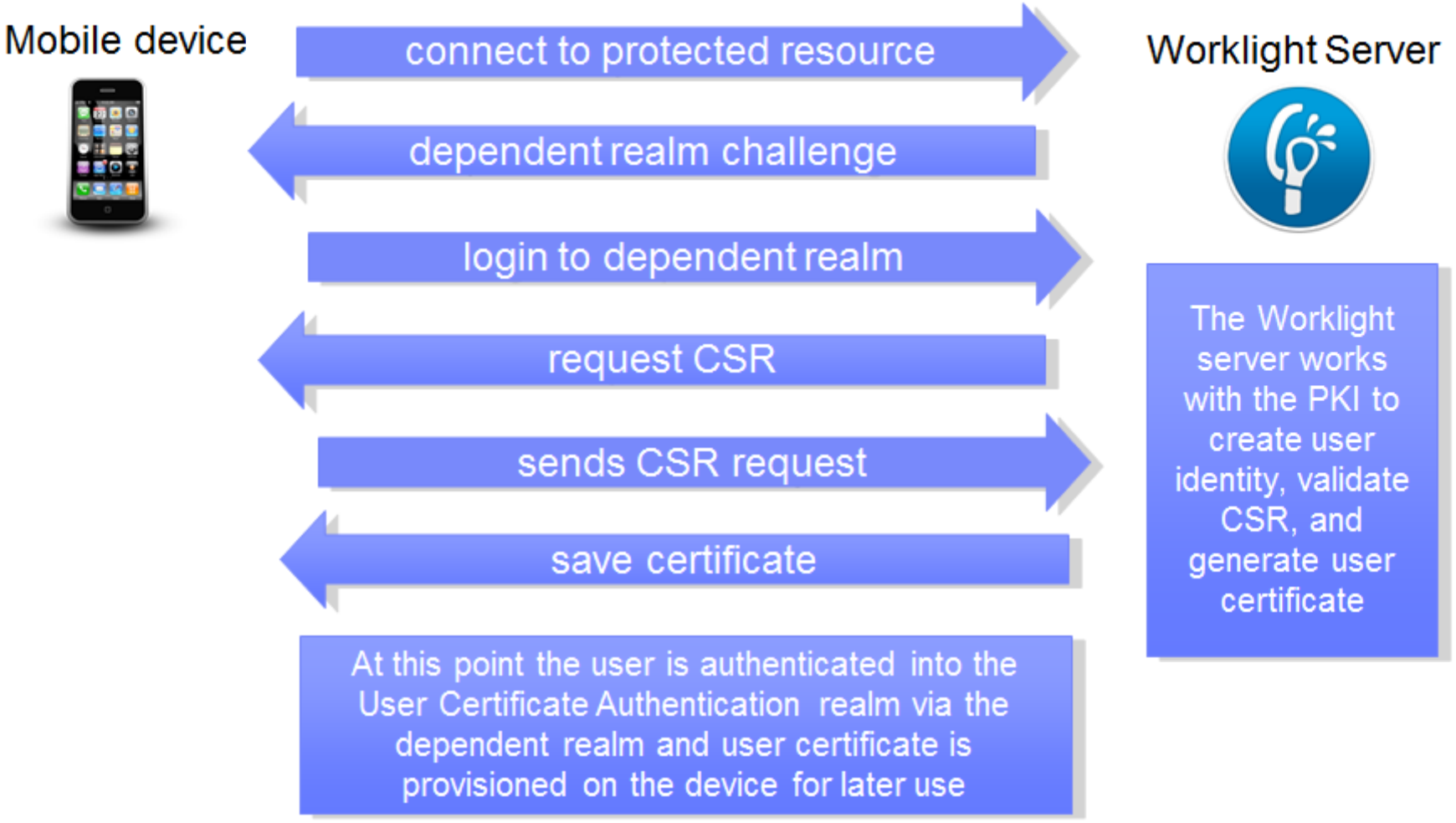
- User certificate authentication is the process in which an X.509 certificate is issued by an existing PKI through the Worklight Server to a specific user on a specific application and device.
- The relevant user information is obtained during the user enrollment process with the specified dependent user realm.
- The user enrollment process relies on a dependent user realm to help it establish the initial user identity to which the X.509 certificate is issued.
- Worklight then provisions the device with the X.509 client certificate for use in subsequent connections to the server.

Understanding how user certificate authentication works (2 of 4)

- The first time a user connects to the Worklight Server, the user must authenticate through the dependent realm to initiate the enrollment process. After a user is enrolled into the User Certificate Authentication realm, subsequent connections to the server occur through the two-way SSL/TLS handshake, where the client certificate is presented as the SSL client entity.

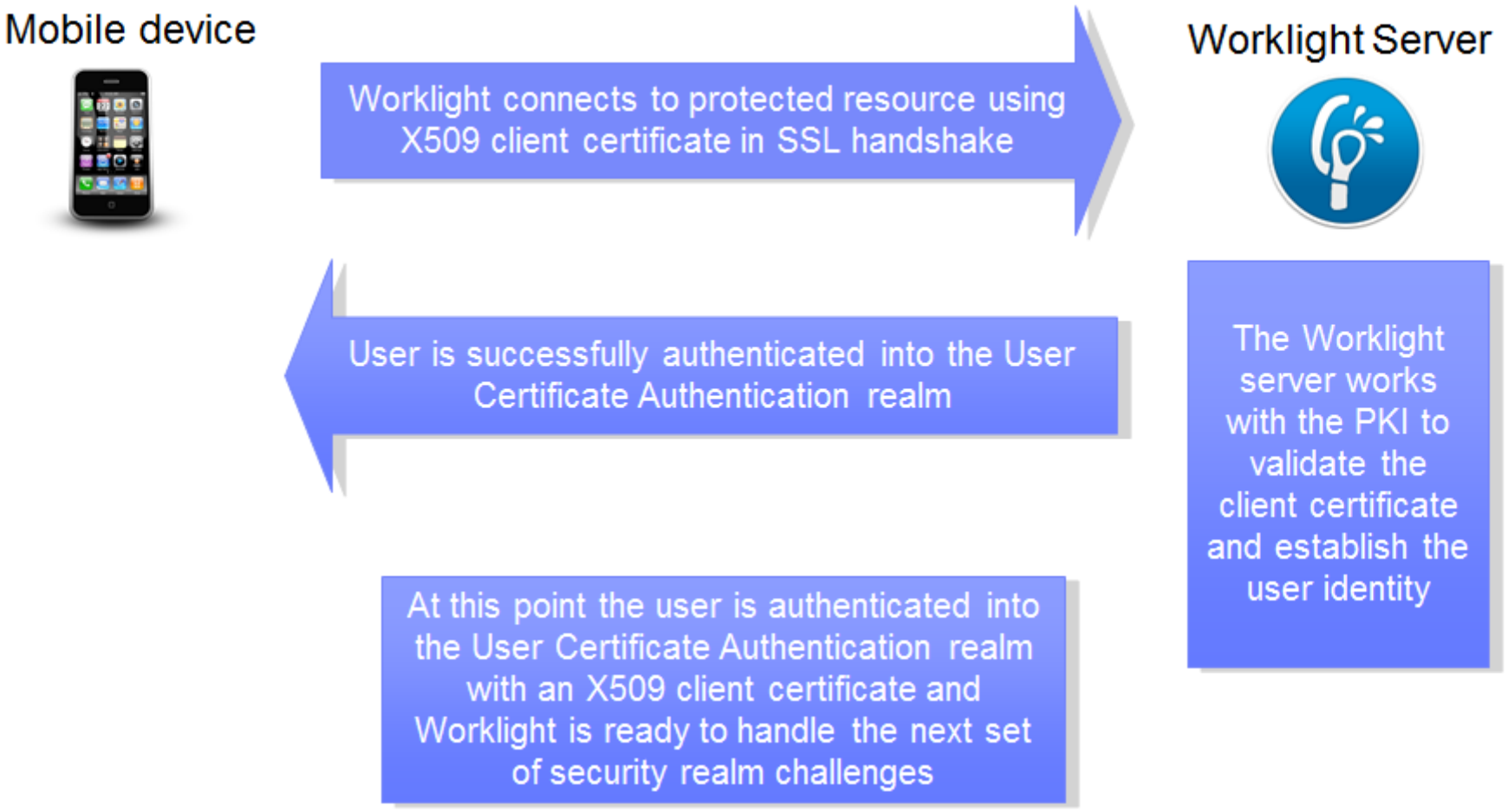
Understanding how user certificate authentication works (3 of 4)

User Enrollment Flow



Understanding how user certificate authentication works (4 of 4)

Client Certificate Authentication Flow



Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

X.509 certificate and certificate authorities (CAs)

- For security reasons, during testing, it is not recommended to use an established CA that uses an embedded PKI in your infrastructure.
- It is possible to create a self-signed CA that can sign both a server certificate and user certificates.
- This module uses the OpenSSL command-line utility.
- OpenSSL is included in most Linux distributions and in Mac OS X. Windows users can obtain an OpenSSL binary from the OpenSSL website.
- The commands that are shown in this module work on Linux and Mac OS X. For Windows, use the equivalent MS-DOS commands.

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Create root CA, signing CA, and certificates (1 of 2)

- Create an empty directory and navigate to that path in your system's terminal.
- Create a basic OpenSSL configuration file that is named `openssl.cnf` (sample is provided on the next slide). Put this file in the directory that you created.
- If you want different policy requirements, see the OpenSSL configuration documentation for instructions on how to configure the various options.

Create root CA, signing CA, and certificates (2 of 2)

- `openSSL.cnf` sample file:

```
[ req ]
default_bits                    = 2048                          # size of keys
default_keyfile                 = key.pem                       # name of generated keys
default_md                      = sha1                          # message digest algorithm
string_mask                     = nombstr                       # permitted characters
distinguished_name              = req_distinguished_name

[ req_distinguished_name ]
0.organizationName              = Organization Name (company)
organizationalUnitName          = Organizational Unit Name (department, division)
emailAddress                    = Email Address
emailAddress_max                = 40
localityName                    = Locality Name (city, district)
stateOrProvinceName             = State or Province Name (full name)
countryName                     = Country Name (2-letter code)
countryName_min                 = 2
countryName_max                 = 2
commonName                      = Common Name (hostname, IP, or your name)
commonName_max                  = 64

[ policy_match ]
countryName                     = optional
stateOrProvinceName             = optional
localityName                    = optional
organizationName                = optional
organizationalUnitName          = optional
commonName                      = supplied
emailAddress                    = optional
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Create root CA (1 of 5)

- Append the following section to the `openssl.cnf` configuration file to set up the root CA requirements.

```
[ root_authority ]
basicConstraints       = CA:TRUE
subjectKeyIdentifier = hash

[ root_authority_ca_config ]
dir                    = ./rootca
certs                  = $dir/certs
new_certs_dir          = $dir/newcerts
database                       = $dir/index.txt
certificate            = $dir/root_ca.crt
private_key            = $dir/root_ca_key.pem
serial                 = $dir/serial
RANDFILE               = $dir/.rand
policy                 = policy_match
```

Create root CA (2 of 5)

- Using a terminal, create the folder structure and requirements for the root CA.

```
# Create a root CA certificate directory structure
mkdir rootca
mkdir rootca/certs rootca/crl rootca/newcerts
touch rootca/serial

export HEXOUT=0123456789ABCDEF
# Create a serial list of random numbers
for y in {1..2048}
do
export output="";
for i in {1..16}
do
      export randomnum=$((RANDOM%16))
      export output=$output${HEXOUT:$randomnum:1};
done
echo "$output" >> rootca/serial
done

touch rootca/index.txt
```

Create root CA (3 of 5)

- For Windows, create the folder structure and requirements for the root CA.

```
REM Create a root CA certificate directory structure
MKDIR rootca
MKDIR rootca\certs
MKDIR rootca\crl
MKDIR rootca\newcerts

REM Create a serial list of random numbers for the root CA
openssl rand -hex -out rootca\serial 8

REM Create index for root CA
COPY NUL rootca\index.txt
```

Create root CA (4 of 5)

- Using a terminal, generate an RSA key pair and then self-sign a root CA certificate. The password must remain secure, even for a test environment. For the following example, the password is `passRoot`.

```
export ROOT_CA_SUBJECT="Development Root CA"

# Create the RSA key pair
# The parameter, 2048, represents the key length
openssl genrsa -des3 -out rootca/root_ca_key.pem -passout
pass:passRoot 2048

# Sign a certificate with the key pair
openssl req -new -x509 -nodes -sha1 -days 365 -key
rootca/root_ca_key.pem -out rootca/root_ca.crt -config
openssl.cnf -subj "/CN=$ROOT_CA_SUBJECT" -extensions
root_authority -passin pass:passRoot
```

Create root CA (5 of 5)

- For Windows, generate an RSA key pair and then self-sign a root CA certificate. The password must remain secure, even for a test environment. For the following example, the password is `passRoot`.

```
REM Create the RSA key pair
REM The parameter, 2048, represents the key length
openssl genrsa -des3 -out rootca\root_ca_key.pem -passout
pass:passRoot 2048

REM Sign a certificate with the key pair
openssl req -new -x509 -nodes -sha1 -days 365 -key
rootca\root_ca_key.pem -out rootca\root_ca.crt -config
openssl.cnf -subj "/CN=Development Root CA" -extensions
root_authority -passin pass:passRoot
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Create signing CA (1 of 5)

- Edit the `openssl.cnf` file that you created earlier and append the following configuration to set up the signing CA configuration options:

```
[ signing_authority ]
basicConstraints                = CA:TRUE,pathlen:0
subjectKeyIdentifier            = hash
authorityKeyIdentifier  = keyid:always, issuer:always

[ signing_authority_ca_config ]
dir                             = ./signingca
certs                           = $dir/certs
new_certs_dir                   = $dir/newcerts
database                        = $dir/index.txt
certificate                     = $dir/signing_ca.crt
private_key                     = $dir/signing_ca_key.pem
serial                  = $dir/serial
RANDFILE                        = $dir/.rand
policy                  = policy_match
email_in_dn                     = false
```

Create signing CA (2 of 5)

- Using the terminal, create the folder structure and requirements for the signing CA. Run these commands from the base directory.

```
# Create a signing CA certificate directory structure
mkdir signingca
mkdir signingca/certs signingca/crl signingca/newcerts
touch signingca/serial

export HEXOUT=0123456789ABCDEF
# Create a serial list of random numbers
for y in {1..2048}
do
export output="";
for i in {1..16}
do
      export randomnum=$((RANDOM%16))
      export output=$output${HEXOUT:$randomnum:1};
done
echo "$output" >> signingca/serial
done

touch signingca/index.txt
```

Create signing CA (3 of 5)

- For Windows, create the folder structure and requirements for the signing CA. Run these commands from the base directory.

```
REM Create a signing CA certificate directory structure
MKDIR signingca
MKDIR signingca\certs
MKDIR signingca\crl
MKDIR signingca\newcerts

REM Create a serial list of random numbers for the signing CA
openssl rand -hex -out signingca\serial 8

REM Create index for signing CA
COPY NUL signingca\index.txt
```

Create signing CA (4 of 5)

- Using the terminal, generate an RSA key pair, and then sign a signing CA CSR with the root CA. For this example, the password is `passSigning`. Run these commands from the base directory.

```
export SIGNING_CA_SUBJECT="Development Signing CA"

openssl genrsa -des3 -out signingca/signing_ca_key.pem -passout
pass:passSigning 2048

openssl req -new -key signingca/signing_ca_key.pem -out
signingca/signing_ca.csr -config openssl.cnf -subj
"/CN=$SIGNING_CA_SUBJECT" -passin pass:passSigning

openssl ca -in signingca/signing_ca.csr -out signingca/signing_ca.crt -
keyfile rootca/root_ca_key.pem -cert rootca/root_ca.crt -config
openssl.cnf -name root_authority_ca_config -extensions signing_authority
-md sha512 -days 365 -passin pass:passRoot
```

Create signing CA (5 of 5)

- For Windows, generate an RSA key pair, and then sign a signing CA CSR with the root CA. For this example, the password is `passSigning`. Run these commands from the base directory.

```
openssl genrsa -des3 -out signingca\signing_ca_key.pem -passout
pass:passSigning 2048

openssl req -new -key signingca\signing_ca_key.pem -out
signingca\signing_ca.csr -config openssl.cnf -subj "/CN=Development
Signing CA" -passin pass:passSigning

openssl ca -in signingca\signing_ca.csr -out signingca\signing_ca.crt -
keyfile rootca\root_ca_key.pem -cert rootca\root_ca.crt -config
openssl.cnf -name root_authority_ca_config -extensions signing_authority
-md sha512 -days 365 -passin pass:passRoot
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Create server certificate (1 of 3)

- Edit the `openssl.cnf` file that you created earlier and append the following configuration to set up the server certificate configuration options.

```
[ server_identity ]
basicConstraints        = CA:FALSE
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always;issuer:always
```

Create server certificate (2 of 3)

- Using the terminal, generate an RSA key pair and sign the new certificate with the signing CA. This certificate is your server identity certificate. The example uses `passServer` as the password. Run these commands from the base directory.

```
# Use the full hostname of your Worklight Server. SSL will break if the full
hostname is not provided, or if an IP address is used as the hostname.
export SERVER_FULL_HOSTNAME=dev.yourcompany.com
mkdir server

# Create the RSA key pair and generate a CSR
openssl genrsa -des3 -out server/server_key.pem -passout pass:passServer 2048
openssl req -new -key server/server_key.pem -out server/server.csr -config
openssl.cnf -subj "/CN=$SERVER_FULL_HOSTNAME" -passin pass:passServer

# Sign the CSR with the signing CA
openssl ca -in server/server.csr -out server/server.crt -keyfile
signingca/signing_ca_key.pem -cert signingca/signing_ca.crt -config openssl.cnf -
name signing_authority_ca_config -extensions server_identity -md sha512 -days 365 -
passin pass:passSigning
```

Create server certificate (3 of 3)

- For Windows, generate an RSA key pair and sign the new certificate with the signing CA. This certificate is your server identity certificate. The example uses `passServer` as the password. Run these commands from the base directory.

```
REM Use the full hostname of your Worklight Server. SSL will break if the full
REM hostname is not provided, or if an IP address is used as the hostname.
SET HOSTNAME=dev.yourcompany.com
mkdir server

REM Create the RSA key pair and generate a CSR
openssl genrsa -des3 -out server\server_key.pem -passout pass:passServer 2048
openssl req -new -key server\server_key.pem -out server\server.csr -config
openssl.cnf -subj "/CN=%HOSTNAME%" -passin pass:passServer

REM Sign the CSR with the signing CA
openssl ca -in server\server.csr -out server\server.crt -keyfile
signingca\signing_ca_key.pem -cert signingca\signing_ca.crt -config openssl.cnf -
name signing_authority_ca_config -extensions server_identity -md sha512 -days 365 -
passin pass:passSigning
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Create certificate chain for the server certificate (1 of 2)

- Using a terminal, send a full certificate chain all the way up to the trust anchor (root CA) for iOS and Android environments. You can concatenate the certificate files to the trust anchor (root CA).

```
# Create a chain for the signing CA
cat signingca/signing_ca.crt rootca/root_ca.crt >
signing_ca_chain.crt

# Create a chain for the server certificate
cat server/server.crt signingca/signing_ca.crt
rootca/root_ca.crt > server_chain.crt
```

Create certificate chain for the server certificate (2 of 2)

- For Windows, send a full certificate chain all the way up to the trust anchor (root CA) for iOS and Android environments. You can concatenate the certificate files to the trust anchor (root CA).

```
REM Create a chain for the signing CA
COPY rootca\root_ca.crt+signingca\signing_ca.crt
signing_ca_chain.crt

REM Create a chain for the server certificate
COPY
rootca\root_ca.crt+signingca\signing_ca.crt+server\server.crt
server_chain.crt
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Export a PKCS12 file for the signing CA

- Export the private key and certificate for the signing CA into a `.p12` keystore file so that the embedded PKI can sign the user certificates with the signing CA.

```
openssl pkcs12 -export -in signingca/signing_ca.crt -inkey
signingca/signing_ca_key.pem -out signingca/signing_ca.p12 -
passin pass:passSigning -passout pass:passSigningP12
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Export a PKCS12 file for the server certificate

- Export the private key and certificate for the server into a `.p12` keystore file so that the server can send the client a valid server certificate.

```
openssl pkcs12 -export -in server_chain.crt -inkey
server/server_key.pem -out server/server.p12 -passout
pass:passServerP12 -passin pass:passServer
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Configure IBM WebSphere® Application Server Liberty profile (Liberty) (1 of 3)

- Enable the `ssl-1.0` and `appSecurity-2.0` features in the `server.xml` file:

```
<featureManager>

     <feature>ssl-1.0</feature>
     <feature>appSecurity-2.0</feature>

</featureManager>
```

Configure IBM WebSphere Application Server Liberty profile (Liberty) (2 of 3)

- Liberty requires setting up the keystore and truststore to establish trust for the generated client certificates. For more information, see http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=/com.ibm.websphere.wlp.nd.doc/ae/rwlp_ssl.html.
  - Set up your server's keystore to use the `.p12` file that was generated earlier (`server.p12`).
  - Set up your truststore to use the `.p12` file that was generated earlier (`signing_ca.p12`).
  - Configure your server's HTTP endpoint and allow (but not require) client-side certificates. This configuration is available by using the `clientAuthenticationSupported="true"` property in the Liberty SSL element.
- Note: The full example is shown on the next slide.

Configure IBM WebSphere Application Server Liberty profile (Liberty) (3 of 3)

- The following example shows the updated SSL configuration:

```
<!-- default SSL configuration is defaultSSLSettings -->
<sslDefault sslRef="defaultSSLSettings" />
<ssl clientAuthenticationSupported="true" id="defaultSSLSettings"
keyStoreRef="defaultKeyStore" trustStoreRef="defaultTrustStore"
/>
<keyStore id="defaultKeyStore" location="server.p12"
password="passServerP12" type="PKCS12" />
<keyStore id="defaultTrustStore" location="signing_ca.p12"
password="passSigningP12" type="PKCS12" />
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Configure authenticationConfig.xml (1 of 3)

- Uncomment the User Certificate Login Module section of the `authenticationConfig.xml` file.

```
<!-- Login Module for User Certificate Authentication -->
<loginModule name="WLUserCertificateLoginModule">
  <className>com.worklight.core.auth.ext.UserCertificateLoginModule</className>
</loginModule>
```

- Uncomment the `wl_userCertificateAuthRealm` section

```
<-- For User Certificate Authentication -->
<realm name="wl_userCertificateAuthRealm" loginModule="WLUserCertificateLoginModule">
  <className>com.worklight.core.auth.ext.UserCertificateAuthenticator</className>
  <parameter name="dependent-user-auth-realm" value="SampleAppRealm" />
  <parameter "name=pki-bridge-class"
value="com.worklight.core.auth.ext.UserCertificateEmbeddedPKI" />
```

Configure authenticationConfig.xml (2 of 3)

- Update the value of the `embedded-pki-bridge-ca-p12-file-path` element to the full path of your signing CA `.p12` file.
- Update the value of the `embedded-pki-bridge-ca-p12-password` element to the password (`passSigningP12`) that was used to create the `.p12` file.
- Update the value of the `dependent-user-auth-realm` element to the dependent realm you want to use (`SampleAppRealm`).
- The realm name (`wl_userCertificateAuthRealm`) cannot be changed.
- Note: The full example is shown on the next slide.

Configure authenticationConfig.xml (3 of 3)

- The following example shows the updates that were listed in the previous slide.

```
<!-- For User Certificate Authentication -->
<realm name="wl_userCertificateAuthRealm" loginModule="WLUserCertificateLoginModule">
  <className>com.worklight.core.auth.ext.UserCertificateAuthenticator</className>
  <parameter name="dependent-user-auth-realm" value="SampleAppRealm" />
  <parameter name="pki-bridge-class"
value="com.worklight.core.auth.ext.UserCertificateEmbeddedPKI" />
  <parameter name="embedded-pki-bridge-ca-p12-file-path"
value="YOUR_BASE_DIRECTORY/signing_ca.p12" />
  <parameter name="embedded-pki-bridge-ca-p12-password" value="passSigningP12" />
</realm>
```

- Define a security test that uses `wl_userCertificateAuthRealm`.

```
<customSecurityTest name="customx509Tests">
  <test realm="wl_antiXSRFRealm" step="1" />
  <test realm="wl_authenticityRealm" step="1" />
  <test realm="wl_directUpdateRealm" mode="perSession" step="1" />
  <test realm="wl_userCertificateAuthRealm" isInternalUserID="true" step="1" />
  <test realm="wl_deviceNoProvisioningRealm" isInternalUserID="true" step="2" />
</customSecurityTest>
```

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Configure application-descriptor.xml

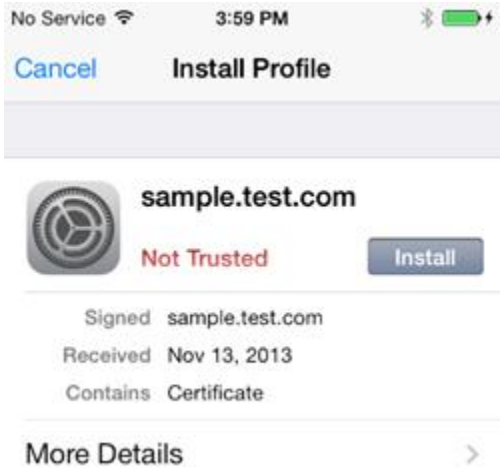
- Ensure that you added iOS or Android environments to your Worklight application.
- Protect your application or environment with your custom security test.
  - `<android securityTest="customx509Tests" version="1.0">`
  - `<iPhone bundleId="com.SampleApp" securityTest="customx509Tests" version="1.0">`
- Build and deploy your application and adapters to the Worklight Server.

Agenda

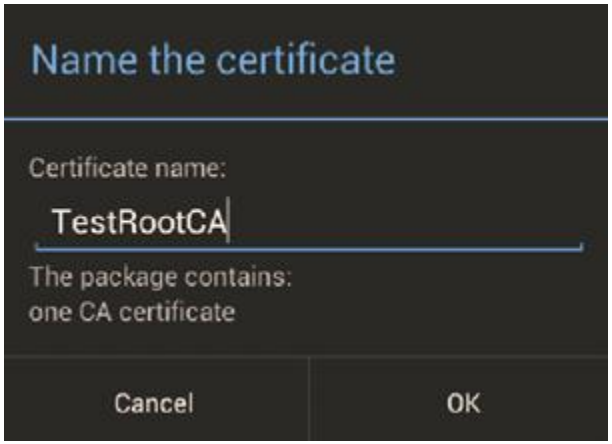
- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Install root CA on iOS and Android

- You must install the root CA that you generated in the previous steps onto your client devices for your devices to trust your Worklight Server over SSL. Email or host the `root_ca.crt` file, and then open the file on your device. The iOS and Android devices ask for approval when you manually attempt to install certificates.



iOS



Android

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Install application and test (1 of 2)

1. Deploy your application to the Worklight Server.
   - Run as → Run on <*server name*>
2. Update the deploy target for HTTPS.
   - Run as → Build Setting and Deploy Target…
   - Mark the check box **Build the application to work with a different Worklight Server**.
   - Enter the server HTTPS address.
     `https://<host>:<https port #>`
   - Enter the application context path.
     `/<path>`

Install application and test (2 of 2)

3. Build the application with the updated deploy target.
   - Run as → Build All Environments
4. Run the application on the specified environments.

Agenda

- Overview
- Prerequisites
- Understanding how user certificate authentication works
- X.509 certificate and certificate authorities (CAs)
  - Create root CA, signing CA, and certificates
  - Create root CA
  - Create signing CA
  - Create server certificate
  - Create certificate chain for the server certificate
  - Export a PKCS12 file for the signing CA
  - Export a PKCS12 file for the server certificate
- Configure IBM WebSphere Application Server Liberty profile (Liberty)
- Configure authenticationConfig.xml
- Configure application-descriptor.xml
- Install root CA on iOS and Android
- Install application and test
- Examining the result

Examining the result

- To confirm a successful configuration, ensure that you see a log-in form the first time that you try to access a protected resource. If `WL.Client.connect()` is uncommented in the `main.js` file, the log-in form is displayed when the application starts. Otherwise, `WL.Client.connect()` must be invoked before you call an adapter procedure to see a log-in form after the adapter is called.
- After you log in through the dependent realm, a successful response from the adapter invocation indicates that the user was successfully enrolled.
- On subsequent connections to the server, you are no longer asked to log in and the adapter calls continue to return successfully.

For more information

- For more information about the *User Certificate Authentication feature*, see the IBM Worklight Foundation user documentation at:
  - http://ibm.biz/knowctr#SSZH4A_6.2.0/com.ibm.worklight.monitor.doc/monitor/c_user_CA.html

Notices

- Permission for the use of these publications is granted subject to these terms and conditions.
- This information was developed for products and services offered in the U.S.A.
- IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
- IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
  - IBM Director of Licensing
    IBM Corporation
    North Castle Drive
    Armonk, NY 10504-1785
    U.S.A.
- For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
  - Intellectual Property Licensing
    Legal and Intellectual Property Law
    IBM Japan Ltd.
    1623-14, Shimotsuruma, Yamato-shi
    Kanagawa 242-8502 Japan
- **The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
- This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.
- Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
- IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
- Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
  - IBM Corporation
    Dept F6, Bldg 1
    294 Route 100
    Somers NY 10589-3216
    USA
- Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
- The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.
- Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

- This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
- Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
  - © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Privacy Policy Considerations

- IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.
- Depending upon the configurations deployed, this Software Offering may use session cookies that collect session information (generated by the application server). These cookies contain no personally identifiable information and are required for session management. Additionally, persistent cookies may be randomly generated to recognize and manage anonymous users. These cookies also contain no personally identifiable information and are required.
- If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent. For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the sections entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Support and comments

- For the entire IBM Worklight documentation set, training material and online forums where you can post questions, see the IBM website at:
  - http://www.ibm.com/mobile-docs
- **Support**
  - Software Subscription and Support (also referred to as Software Maintenance) is included with licenses purchased through Passport Advantage and Passport Advantage Express. For additional information about the International Passport Advantage Agreement and the IBM International Passport Advantage Express Agreement, visit the Passport Advantage website at:
    - http://www.ibm.com/software/passportadvantage
  - If you have a Software Subscription and Support in effect, IBM provides you assistance for your routine, short duration installation and usage (how-to) questions, and code-related questions. For additional details, consult your IBM Software Support Handbook at:
    - http://www.ibm.com/support/handbook
- **Comments**
  - We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.
  - For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.
  - When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state.
  - Thank you for your support.
  - Submit your comments in the IBM Worklight Developer Edition support community at:
    - https://www.ibm.com/developerworks/mobile/worklight/connect.html
  - If you would like a response from IBM, please provide the following information:
    - Name
    - Address
    - Company or Organization
    - Phone No.
    - Email address

Thank You