MICRO FOCUS

ZENworks Mobile Workspace

SDK Server Guide

May 2017

Legal Notice

TABLE OF CONTENTS

1 Enterprise Authentication Integration 3

1.1 JAAS Login module 3

1.2 JAAS Context name 3

1.3 JAAS Callback handler 3

1.4 ZENworks Mobile Workspace Configuration 3

2 Enterprise Group Integration 5

2.1 Synchronization parameters interface 5

2.2 Synchronization manager interface 5

2.3 Synchronization manager example 7

1 ENTERPRISE AUTHENTICATION INTEGRATION

The ZENworks Mobile Workspace SDK provides APIs to delegate user authentication to any type of enterprise authentication server such as LDAP servers, RSA servers or RADIUS-based servers. However, for backward compatibility, LDAP AD authentication has been kept as the default authentication module.

1.1 JAAS Login Module

The authentication is based on JAAS standards which delegates authentication to the JEE application server, and allows users to use pre-defined modules or define their own custom module by implementing the javax.security.auth.spi.LoginModule interface.

1.2 JAAS Context Name

The "JAAS context name" parameter must be referenced in ZENworks Mobile Workspace configuration with the name of the JAAS context (also known as "realm") in which the custom LoginModule is configured. Typically, the LoginModule is provided and configured by the Java Entreprise (JEE) application server.

1.3 JAAS Callback handler

The JAAS Login module (either default or custom) is initialized by default with a javax.security.auth.callback.CallbackHandler that provides the username and password entered by the end-user as a javax.security.auth.callback.NameCallback and a javax.security.auth.callback.PasswordCallback, respectively to the LoginModule. If the configured LoginModule does not require other credentials, the default JAAS callback handler class name field can be left blank.
If the LoginModule requires additional configuration (for example, a RADIUS LoginModule requiring the server address and shared key parameters), a CallbackHandler with the following constructor arguments must be provided with the LoginModule and referenced in the ZENworks Mobile Workspace configuration: `public MyCustomCallbackHandler(String userName, char[] password);`
The custom CallbackHandler is responsible to load its additional parameter(s) from any source, such as configuration file, JNDI object, system property, etc.

1.4 ZENworks Mobile Workspace Configuration

The custom classes must be available on the classpath of the ZENworks Mobile Workspace application, and then the JAAS context name referencing the custom LoginModule and the custom CallbackHandler class name must be set in the corresponding domain console of the security server (Server->Domains):

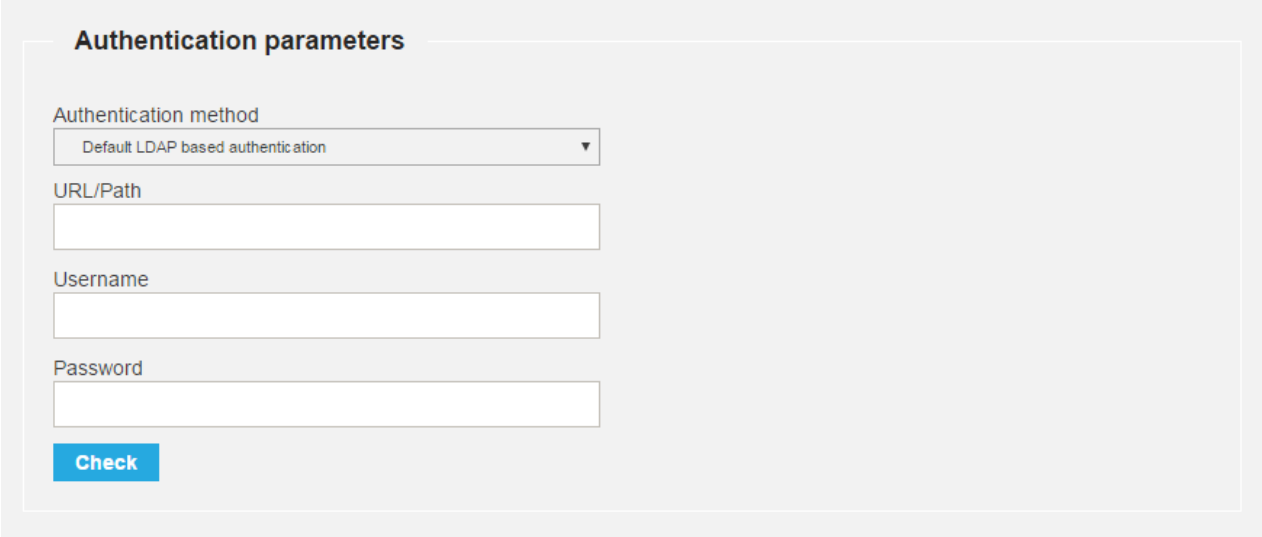

Figure 1: Authentication parameters

2 ENTERPRISE GROUP INTEGRATION

The ZENworks Mobile Workspace SDK provides API to synchronize security groups against any data sources such as LDAP servers, databases or flat files. However, for backward compatibility, LDAP (ActiveDirectory-compatible) synchronization has been kept as the default synchronization module. Group synchronization works with a proprietary interface provided by the ZENworks Mobile Workspace SDK, it does not rely on roles that the LoginModule may provide when authenticating users, although this interfaces uses Groups.

2.1 Synchronization Parameters Interface

Even if all parameters can be hard-coded in the synchronization manager or retrieved from a custom data source, ZENworks Mobile Workspace SDK provides an integrated and flexible way to retrieve them. The administration console allows IT managers to set several parameters that can be used by a custom synchronization manager:

- **URL/Path (String):** This parameter can be used to provide an URL or a path to the data source such as LDAP URL, file path or database connection URL. It can also be used to provide any additional parameters (i.e. using semi-colon: url;param1;param2)

- **Username (String):** This parameter is commonly used to provide a username needed to connect the synchronization data source.

- **Password (String):** This parameter is commonly used to provide a password needed to connect the synchronization data source.

Developers of custom adapters will be able to access these parameters through the *ch.sysmosoft.sense.common.synchronization.GroupSynchronizationParameter interface*.
ZENworks Mobile Workspace synchronization mechanism does not directly use Principal JAAS class (java.security.Principal). For the comfort purpose of the administration console, Principal class has been extended by SENSEPrincipal (ch.sysmosoft.sense.common.synchronization.SENSEPrincipal) which force developers to return the first name and the last name of a principal. This interface is discussed later as part of the synchronization manager.

2.2 Synchronization manager interface

This is the main class of the synchronization mechanism which will be dynamically loaded and used by the ZENworks Mobile Workspace Security server to synchronize groups and users.
Based on the *ch.sysmosoft.sense.common.synchronization.GroupSynchronizationManager* interface, four methods must be implemented to fulfill the contract:

- **void init(GroupSynchronizationParameter groupSynchronizationParameter):** As its name indicates, this method aims to initialize the synchronization manager with the given parameters. These parameters are those previously set through the administration console.

- **boolean checkParameters():** This method returns a boolean to indicate if the parameter set through the administration console is correct or not. It is up to the developer of the custom module to perform appropriate check(s) (parameter required, right URL/Path, etc). This method will be used by the administration console to delegate the check and let the IT manager know if the parameters are correct or not.

- **List<Group> getGroups():** Used in the group editing user interface, this method must return the list of JAAS groups (java.security.acl.Group) allowed to be synchronized by ZENworks Mobile Workspace. The name cannot be null or empty as it will be used to retrieve principals in that group. It means no member (principal) of this group will be used directly. Only principal(s) returned by the method List< SENSE Principal> getPrincipals(Group selectedGroup) will be synchronized.

- **List<SENSEPrincipal> getPrincipals(Group selectedGroup):** When a group is selected in the user interface, this method is called to retrieve members of the selected group. In the case of a new group creation, ZENworks Mobile Workspace synchronization mechanism will add all users to the security group. If you want to synchronize an existing group, ZENworks Mobile Workspace will compare both groups and synchronize the existing group against the reference group. The principal's name will be used as the username and displayed, as is, in the administration console. Users without user names or already members of different groups will not be synchronized.

Once the custom module has been developed, it must be referenced in the server classpath and the class implementing the GroupSynchronizationManager must be set in the corresponding domain console of the security server (Server->Domains)

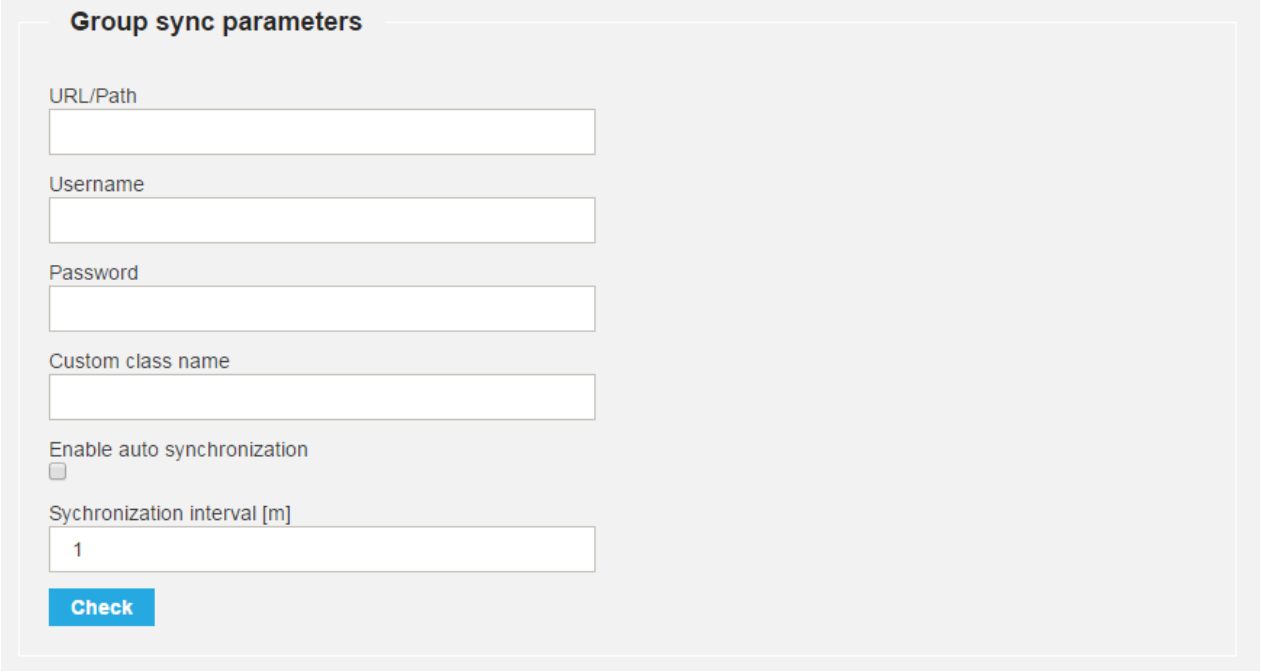

Figure 2: Group synchronization parameters

2.3 Synchronization manager example

The first step of implementing a custom synchronization manager is to define objects that implement JAAS standards.
The following class defines a simple group which must be initialized with a name. Keep in mind that the name is the only attribute used by the ZENworks Mobile Workspace synchronization mechanism. Members will not be retrieved using the members() method as they will be retrieved later through the getPrincipals() method of the synchronization manager.

```
public class SimpleGroup implements Group {

    private String name;

    public SimpleGroup(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
    public boolean addMember(Principal user) {
        return false;
    }
    public boolean isMember(Principal member) {
        return false;
    }
    public Enumeration<? extends Principal> members() {
        return null;
    }
    public boolean removeMember(Principal user) {
        return false;
    }

}
```

Code 1: Group synchronization

The following class defines a simple user based on ZENworks Mobile Workspace principle interface. The name is important as it will be used for identification and authentication when establishing a ZENworks Mobile Workspace security session.

```
public class SimpleUser implements SENSEPrincipal {

    private String name;
    private String firstName;
    private String lastName;

    public SimpleUser(String name, String firstName, String lastName) {
        this.name = name;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public String getName() {
        return name;
    }

     }
```

Code 2: Principal

Once JAAS objects have been defined, a custom synchronization manager can be created. The following is a rough example of the manager:

1. Keeps a reference on the parameters.
2. Checks if all parameters have been set (even if not needed later)
3. Returns a static group list.
4. Returns a static user list if the group is the one returned by the getGroups() method.

```
public class MyCustomSynchronizationManager
            implements GroupSynchronizationManager {

    private static final String GROUP_NAME = "My test group";
    private GroupSynchronizationParameter param;

    public void init(GroupSynchronizationParameter param) {
        this.param = param;
    }

    public boolean checkParameters() {
        return (param.getPassword() != null && !param.getPassword().isEmpty())
            && (param.getUrl() != null && !param.getUrl().isEmpty())
            && (param.getUsername() != null && !param.getUsername().isEmpty());
    }

    public List<Group> getGroups() {
        return Arrays.asList(new SimpleGroup(GROUP_NAME));
    }

    public List<SENSEPrincipal> getPrincipals(Group group) {
        if (GROUP_NAME.equals(group.getName())) {
            return Arrays.asList(new SimpleUser("test_login","Test","Test"));
        } else {
            return Collections.emptyList();
        }
    }

}
```

Code 3: Group synchronization