Brekeke SIP Server

Version 3

Authentication Plug-in
Developer's Guide

Brekeke Software, Inc.

Version

Brekeke SIP Server Version 3 Authentication Plug-in Developer's Guide

Disclaimer

Brekeke Software, Inc. reserves the right to change any information found in this document without any written notice to the user.

Trademark Acknowledgement

- *Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.*
- *Other logos and product and service names contained in this document are the properties of their respective owners.*

1. WECOME 4

2. REQUIREMENTS 5

2.1. User Directory 5

2.2. Plug-in 5

3. PROCESS FLOW 6

4. CLASSES AND INTERFACES 7

4.1. UserDir Interface 7

4.2. UserRecord Class 7

4.3. Envrnmt Class 8

1 getStr 8

2 getStr 8

3 getInt 9

4 getLong 9

4.4. Logging Class 9

1 print 9

2 println 9

3 println 10

4.5. LogLevel 10

1 LogLevel 11

5. METHODS 11

5.1. init 11

5.2. close 11

5.3. lookup 11

5.4. append 12

5.5. remove 12

5.6. remove 13

5.7. getCount() 13

6. PLUG-IN INSTALLATION 13

7. PLUG-IN SAMPLE CODE 14

1. Wecome

The Brekeke SIP Server authenticates SIP request methods, such as INVITE and REGISTER sent from SIP client user agents. The server queries the user directory database and validates the query through the default Authentication plug-in. The user directory is a user database that has a user name and password.

If you want to use another existing user directory service (a user database or authentication server) for authentication on the Brekeke SIP Server, it is possible by creating your own Authentication plug-ins.

In this document, we introduce the classes, interfaces, and methods necessary to develop Authentication plug-ins.

2. Requirements

2.1. User Directory

As a minimum requirement, the user directory must include a user name and password for each user. The Brekeke SIP Server only queries but not add or delete user data. Therefore, you must create and delete users with another tool.

2.2. Plug-in

The plug-in will be loaded as a Java class. It is recommended that you use Java version 1.6 or later for developing a plug-in. The class for the plug-in must be implemented by `UserDir` interface (refer to Section 4.1).

The methods that should be implemented in the plug-in class are listed below:

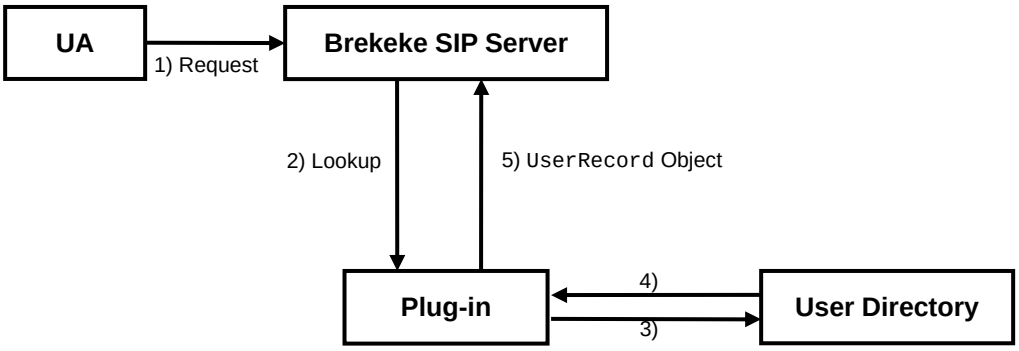
Method Name	Action
`init()`	Initialize the plug-in
`close()`	Close the plug-in
`lookup()`	Search a user
`append()`	Add a user
`remove()`	Delete a user
`getCount()`	Get a total number of users

Please refer to the section 5 Methods for additional details.

For compiling your program, include the following path in the CLASSPATH.

```
<SIP_server_install_directory>\webapps\sip\WEB-INF\lib\ondosip.jar
```

3. Process Flow



1) A SIP user agent sends a SIP request such as REGISTER or INVITE to Brekeke SIP Server.
2) Brekeke SIP Server passes the user name, SIP method name, destination SIP-URI and **`Proxy-Authorization`** header value or **`Authorization`** header value which are in the SIP request to the plug-in's **`lookup()`** method.
3) The plug-in queries the user directory for the record of the user name.
4) The user directory returns an appropriate record.
5) The plug-in puts the record to a **`UserRecord`** object (refer to Section 4.2, `UserRecord` Class) and returns the object to Brekeke SIP Server. If no record is found, null is returned.
6) Brekeke SIP Server compares the returned result with the authentication information in the SIP request if a **`UserRecord`** object is returned from the plug-in.

4. Classes and Interfaces

This section explains the classes and interfaces you need to develop plug-ins.

4.1. UserDir Interface

Package: **`com.brekeke.net.usrdir`**

The plug-in class must be implemented by **`UserDir`** interface.
Please refer to the section 5 Methods for additional details about the methods of **`UserDir`** interface.

4.2. UserRecord Class

Package: **`com.brekeke.net.usrdir`**

`UserRecord` is the class that holds a user's information. This object is returned from a **`lookup()`** method when Brekeke SIP Server calls the method.

`UserRecord` class does not contain methods. It only contains the following member variables.

Type	Variable Name	Content
`String`	**`username`**	User name
`String`	**`password`**	Password in plain text
`boolean`	**`bAuthorized`**	A flag on whether authorized or not
`long`	**`uid`**	User ID (Optional)
`long`	**`gid`**	Group ID (Optional)
`String`	**`longname`**	Long user name (Optional)
`String`	**`email`**	E-mail address (Optional)
`String`	**`desc`**	Description of the user (Optional)
`long`	**`timeexpire`**	Expiration (Optional)
`long`	**`timemake`**	Date and time of when this record was created (Optional)
`Object`	**`ext`**	For extension (Optional)

All variables, except username, password, and bAuthorized, are optional. If Brekeke SIP Server needs to calculate an encrypted password from the plain password, and authorize by comparing the calculated password with a header field (**`Proxy-Authorization or Authorization`**) in the request from user agent, set **`bAuthorized = false`**. The default value of **`bAuthorized`** is false. If the authorization is done at the plug-in and Brekeke SIP Server does not need to do an authorization, set **`bAuthorized = true`**.

4.3. Envrnmt Class

Package: **`com.brekeke.common`**

`Envrnmt` is a class that holds environment properties. The **`Envrnmt`** object is passed to the plug-in as an argument of **`init()`** method.

Environment properties must be set in the Brekeke SIP Server Admin Tool's **[Configuration] -> [Advanced]** page in advance. Restart Brekeke SIP Server after you modify the property variables.

1 getStr

Method: **`public String getStr( String key )`**

Description: Search for the **`String`** type property with the specified key in the properties.
The method returns null if the key is not found.

Parameters: **`key`** – the property key.

Returns: the **`String`** value with the specified key value.

2 getStr

Method: **`public String getStr( String key, String defStr )`**

Description: Search for the String type property with the specified key.
The method returns the default value argument if the key is not found.

Parameters: **`key`** – the property key.
`defStr` – the default value.

Returns: The **`String`** value with the specified key value.

3 getInt

Method: `public int getInt( String key, int defNum )`

Description: Search for the int type property with the specified key.
The method returns the default value argument if the key is not found.

Parameters: **`key`** – the property key.
`defNum` – the default value.

Returns: The **`int`** value with the specified key value.

4 getLong

Method: `public long getLong( String key, long defNum )`

Description: Search for the **`long`** type property with the specified key.
The method returns the default value argument if the key is not found.

Parameters: **`key`** – the property key.
`defNum` – the default value.

Returns: The **`long`** value with the specified key value.

4.4. Logging Class

Package: **`com.brekeke.common`**

`Logging` is a class to output logs. This object is passed to the plug-in through **`init()`** method as an argument.

1 print

Method: `public void print( String str, LogLevel loglevel, int require )`

Description: If the current log level meets the given required log level, then the string is output.

Parameters: **`str`** - The String to be output
`loglevel` – Current log level
`require` – Required log level

2 println

Method: `public void println( LogLevel loglevel, int require )`

Description: If the current log level meets the given required log level, a new line character (**\n**) is output.

Parameters: **`loglevel`** – Current log level
`require` – Required log level

3 println

Method:

```
public void println( String str, LogLevel loglevel, int require )
```

Description: If the current log level meets the given required log level, then the string and a new line character (**\n**) is output.

Parameters: **`str`** – The String to be output

`loglevel` – Current log level

`require` – Required log level

4.5. LogLevel

Package: **`com.brekeke.common`**

`LogLevel` is a class that contains log level values.

This is used for logging. Level is set for console output (standard output) or file output.

Log level values are as follows:

Variable name	Value	Level
LOG_LEVEL_SILENT	0	No Logging
LOG_LEVEL_SYSTEM	1	System
LOG_LEVEL_ERROR	2	Error
LOG_LEVEL_WARNING	4	Warning
LOG_LEVEL_EXCEPTION	8	Exception
LOG_LEVEL_STATUS	16	Status
LOG_LEVEL_DETAIL	32	Detail
LOG_LEVEL_DEBUG	64	Debug
LOG_LEVEL_ALL	255	All

You can specify multiple log levels by logical add (OR). For example, if you want to output only System and Detail logs, the log level will be 34 (2 OR 32).

Log level = 2 OR 32= 34

1 LogLevel

Constructor: **`public LogLevel( int levelConsole, int levelFile )`**

Description: Creates a new LogLevel object.

Parameters: **`levelConsole`** - Log level for console output.

5. Methods

This section explains all methods that a plug-in must implement. These methods are declared in **`UserDir`** interface.

5.1. init

Method: **`public void init( Envrnmt env, Logging log ) throws Exception`**

Description: **`init()`** method is called when Brekeke SIP Server starts. Please include any kind of initialization regarding the plug-in such as connecting a database or setting log levels.

Parameters: **`env`** - Envrnmt object

`log` - Logging object

5.2. close

Method: **`public void close() throws Exception`**

Description: **`close()`** method is called while Brekeke SIP Server is shutting down. Please include any kind of endings regarding the plug-in such as disconnection with database.

5.3. lookup

Method:

`public UserRecord lookup( String username, String method, String destination, String authinfo ) throws Exception`

Description:

`lookup` method is called when Brekeke SIP Server needs to do authentication. User name, SIP method name of the request, destination SIP-URI and **`Proxy-Authorization`** header value or **`Authorization`** header value are passed as argument.

User name is supposed to be used as a search key for a user directory. SIP method name and destination URI can be used for the information to make a decision. Put the search result of the user into a **`UserRecord`** object as a return value.

You must set **username** variable in **UserRecord** object. If the authorization is done here, and Brekeke SIP Server does not need to do an authorization, you do not need to set **password** variable however you must set **bAuthorized = true**. Otherwise, a plain password must be set in **password** variable. If the password in the user directory is encrypted, it must be decrypted.

Return value is a **UserRecord** object. If you want to fail the authentication regardless of the search result, return null.

If an exception occurs, Brekeke SIP Server fails the authentication and does not execute the request.

Parameters: **username** - User name of the user who sent the request
method - SIP method name of the request
destination - Destination SIP-URI of the request
authinfo - **Proxy-Authorization header value (INVITE) or Authorization header value (REGISTER)**

Returns: **UserRecord** object
null for authorization failure

5.4. append

Method: **public boolean append(UserRecord record) throws Exception**

Description: Brekeke SIP Server does not call this append method. This is supposed to be called from other tools. When this method is called, add the record to the user directory.

Parameters: **record** – UserRecord object

Returns: true if successful
false if not successful

5.5. remove

Method: **public boolean remove(UserRecord record) throws Exception**

Description: Brekeke SIP Server does not call this remove method. This is supposed to be called from other tools. When this method is called, remove the record from the user directory.

Parameters: **record** – UserRecord object

Returns: true if successful
false if not successful

5.6. remove

Method: `public boolean remove( String username ) throws Exception`

Description: Brekeke SIP Server does not call this remove method. This is supposed to be called from other tools. When this method is called, remove the record of given user name from the user directory.

Parameters: **`username`** – user name

Returns: true if successful
false if not successful

5.7. getCount()

Method: `public int getCount() throws Exception`

Description: When this method is called, return the number of users in the user directory.

Returns: the number of users in the user directory

6. Plug-in Installation

Copy the developed plug-in to some appropriate directory, and add the directory in Java class path (CLASSPATH).
Also, set the plug-in name to the property variable "**`net.usrdir.plugins`**" in the Brekeke SIP Server Admin Tool's **[Configuration]->[Advanced]** page.

Suppose the plugin-in **`SampleUserDir.class`** is in the directory **`/plugins/userdir`** and the package name of the plug-in class is **`com.domain.proxy.plugins`**

Add "**`/plugins/userdir`**" to the environment variable **`CLASSPATH.`**

For example:

```
CLASSPATH=/usr/java/jdk/lib:/plugins:/plugins/userdir
```

Set the plug-in to the property variable **`net.usrdir.plugins`** in the **[Configuration]->[Advanced]** page.

For example:

```
net.usrdir.plugins = com.domain.proxy.plugins.SampleUserDir
```

7. Plug-in Sample Code

Here is a sample code that explains how to implement init and lookup method.

```
package com.domain.proxy.plugins ;

import com.brekeke.common.* ;
import com.brekeke.net.usrdir.* ;

public class SampleUserDir implements UserDir
{
  Envrnmt  env = null ;
  Logging  log = null ;
  LogLevel  loglevel = null ;


  // init
  public void init( Envrnmt env, Logging log ) throws Exception
  {
    this.env = env ;
  this.log = log ;

    // Log level setting
    // It is obtained from the variable “net.userdir.loglevel.console”
    loglevel = new LogLevel( 0,
      env.getInt( "net.userdir.loglevel.file", LogLevel.LOG_LEVEL_EXCEPTION ) ) ;

     log.println( "SampleUserDir: start", loglevel, LogLevel.LOG_LEVEL_SYSTEM ) ;


    //
    // Initialization of database connection, etc.
    //
  }
```

```
  // lookup
  public UserRecord lookup( String username, String method, String destination,
    String authinfo ) throws Exception
  {

    //
    // database inquiry for a user information by username key.
    //


    if ( the user info not found ) {
      return ( null ) ;
    }

    if ( the user doesn´t have the authority to execute the method ) {
      return ( null ) ;
    }

    if ( the user doesn´t have the authority to access the destination SIP-URI ) {
      return ( null ) ;
    }


    UserRecord record = new UserRecord() ;
    record.username = username ;
    record.password = the plain password obtained from the database

    return ( record ) ;
  }


}
```