javax.security.auth.callback

# Interface CallbackHandler

---

```
public interface CallbackHandler
```

An application implements a `CallbackHandler` and passes it to underlying security services so that they may interact with the application to retrieve specific authentication data, such as usernames and passwords, or to display certain information, such as error and warning messages.

CallbackHandlers are implemented in an application-dependent fashion. For example, implementations for an application with a graphical user interface (GUI) may pop up windows to prompt for requested information or to display error messages. An implementation may also choose to obtain requested information from an alternate source without asking the end user.

Underlying security services make requests for different types of information by passing individual Callbacks to the `CallbackHandler`. The `CallbackHandler` implementation decides how to retrieve and display information depending on the Callbacks passed to it. For example, if the underlying service needs a username and password to authenticate a user, it uses a `NameCallback` and `PasswordCallback`. The `CallbackHandler` can then choose to prompt for a username and password serially, or to prompt for both in a single window.

A default `CallbackHandler` class implementation may be specified in the *auth.login.defaultCallbackHandler* security property. The security property can be set in the Java security properties file located in the file named <JAVA_HOME>/lib/security/java.security. <JAVA_HOME> refers to the value of the java.home system property, and specifies the directory where the JRE is installed.

If the security property is set to the fully qualified name of a `CallbackHandler` implementation class, then a `LoginContext` will load the specified `CallbackHandler` and pass it to the underlying LoginModules. The `LoginContext` only loads the default handler if it was not provided one.

All default handler implementations must provide a public zero-argument constructor.

## Method Summary

Methods

| Modifier and Type | Method and Description |
|---|---|
| `void` | `handle(Callback[] callbacks)`<br>Retrieve or display the information requested in the provided Callbacks. |

## Method Detail

### handle

```
void handle(Callback[] callbacks)
            throws IOException,
                   UnsupportedCallbackException
```

Retrieve or display the information requested in the provided Callbacks.

The `handle` method implementation checks the instance(s) of the `Callback` object(s) passed in to retrieve or display the requested information. The following example is provided to help demonstrate what an `handle` method implementation might look like. This example code is for guidance only. Many details, including proper error handling, are left out for simplicity.

```
public void handle(Callback[] callbacks)
throws IOException, UnsupportedCallbackException {

  for (int i = 0; i < callbacks.length; i++) {
     if (callbacks[i] instanceof TextOutputCallback) {

         // display the message according to the specified type
         TextOutputCallback toc = (TextOutputCallback)callbacks[i];
         switch (toc.getMessageType()) {
         case TextOutputCallback.INFORMATION:
             System.out.println(toc.getMessage());
             break;
         case TextOutputCallback.ERROR:
             System.out.println("ERROR: " + toc.getMessage());
             break;
         case TextOutputCallback.WARNING:
             System.out.println("WARNING: " + toc.getMessage());
             break;
         default:
             throw new IOException("Unsupported message type: " +
                                 toc.getMessageType());
         }

     } else if (callbacks[i] instanceof NameCallback) {

         // prompt the user for a username
         NameCallback nc = (NameCallback)callbacks[i];

         // ignore the provided defaultName
         System.err.print(nc.getPrompt());
         System.err.flush();
         nc.setName((new BufferedReader
                 (new InputStreamReader(System.in))).readLine());

     } else if (callbacks[i] instanceof PasswordCallback) {

         // prompt the user for sensitive information
         PasswordCallback pc = (PasswordCallback)callbacks[i];
         System.err.print(pc.getPrompt());
         System.err.flush();
         pc.setPassword(readPassword(System.in));

     } else {
         throw new UnsupportedCallbackException
                 (callbacks[i], "Unrecognized Callback");
     }
  }
}

// Reads user password from given input stream.
private char[] readPassword(InputStream in) throws IOException {
   // insert code to read a user password from the input stream
}
```

**Parameters:**

`callbacks` - an array of `Callback` objects provided by an underlying security service which contains the information requested to be retrieved or displayed.

**Throws:**

`IOException` - if an input or output error occurs.

`UnsupportedCallbackException` - if the implementation of this method does not support one or more of the Callbacks specified in the `callbacks` parameter.

Overview Package Class Use Tree Deprecated Index Help

*Java™ Platform Standard Ed. 7*

**Prev Class** **Next Class** Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method