Space Details:

Key	CSJM
Name	Cryptshare Java API Manual
Description	
Created by	hartwigr (Apr 18, 2016)

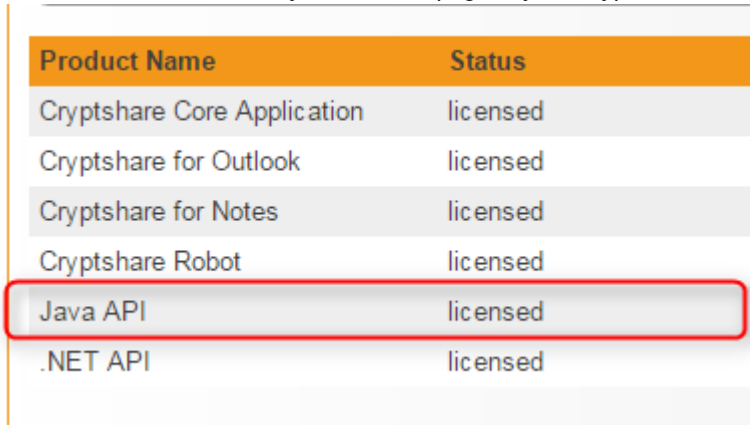
Available Pages:

Cryptshare Java API Manual : Prerequisites

Created by René Hartwig on May 23, 2018

Valid Cryptshare Server License

In order to be able to use the Cryptshare Java API with your Cryptshare Server instance, you will need to have a valid license. This license needs to be installed on the Cryptshare Server. You can find the status of your license in the section 'License Information' on the System-Status page of your Cryptshare Administration web site:



Product Name	Status
Cryptshare Core Application	licensed
Cryptshare for Outlook	licensed
Cryptshare for Notes	licensed
Cryptshare Robot	licensed
Java API	licensed
.NET API	licensed

WSDL Access

The Cryptshare Java API requires access to your Cryptshare Server, as it uses its web services to perform the required operations. The Cryptshare Server provides two web services, one that handles general service requests, and one that handles the file transfer:

- https://<your_cryptshare_server_url>/service**/serviceV2?wsdl**
- https://<your_cryptshare_server_url>/service**/transferV2?wsdl**

  Warning

  If one or both interfaces cannot be reached from the client side, you will not be able to use the Java API.

  HTTP and HTTPS Access

  If you have "Force secure connection" enabled for your Cryptshare Server, you have to use the HTTPS protocol to connect to your server using the Cryptshare Java API. Trying to access the server with HTTP in this case will result in an XML parsing exception when trying to read the WSDL file, since the server will then not actually return the WSDL file, but a HTTPS redirect message.

  So if you want to be able to access the Cryptshare Server using the HTTP protocol, you will have to disable the "Force secure connection for Cryptshare User Interface" setting on the "Connection Settings" page of your Cryptshare Administration Interface.

Compatible Java Runtime Environment

The Cryptshare Java API requires **Java Runtime Environment 1.6** or higher.

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
-

- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : Verification

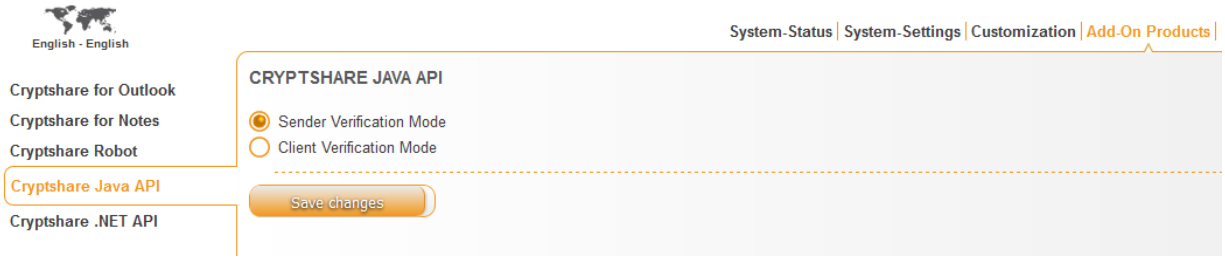
Created by René Hartwig on May 23, 2018

Verification Modes

Most of the services offered by the Cryptshare Server require a form of verification, before they can be used. The Cryptshare Server supports two types of verification: Sender Verification and Client Verification.

- If the Cryptshare Server is configured to use the Sender Verification mode, each email address that is used as the sender when using the Cryptshare services will need to be verified.
- With Client Verification, the whole client system on which the API application is running is verified on the Cryptshare Server. Then that **Client** will be able to access the server's services using any sender address, without any further verification requirements.

The verification mode can be configured on the Add-On Products tab of the Cryptshare Server Administration Interface:



Checking the Verification

You can use API methods to check what kind of verification mode is configured on the server, and whether a given sender email address is verified, or if the verification mode is set to Client Verification, whether the **Client** is verified.
As with all API operations, you first have to create the **Client** instance, as described under API Functions. Then you can call the **Client**'s **checkVerification()** method to check the verification status. It will return a **VerificationStatus** instance, containing the queried verification status.
If the method **isSenderVerification()** in the **VerificationStatus** instance returns *true*, then the verification mode on the server is configured to be Sender Verification, if it returns *false*, then the server is set to the Client Verification mode. The method **isVerified()** will tell you whether the sender address you specified is verified or not. If the verification mode on the server is set to Client Verification and the **Client** has not yet been verified, the method **isVerified()** will return *false*. If the **Client** has been verified, it will return *true* for all sender addresses. If the verification mode is set to Sender Verification, then the method **isVerified()** will only return *true*, if the sender address has actually been verified.

Example: Checking the Verification Status

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection,
"C:\\temp\\client.store");

// now you can call the checkVerification() method on the client
VerificationStatus result = client.checkVerification();

if (result.isSenderVerification()) {
```

```
    System.out.println("Verification mode is Sender Verification");
} else {
    System.out.println("Verification mode is Client Verification");
}

if (result.isVerified()) {
    System.out.println("Sender address is verified!");
} else {
    System.out.println("Sender address is NOT verified!");
}
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : Sender Verification

Created by René Hartwig on May 23, 2018

With Sender Verification mode, a verification has to be performed for each sender email address that is used. This makes sure that the sender address that is being used for a transfer through the API is valid and authorized for use by its owner. That's important, because the Cryptshare Policy set up on the server grants usage permissions and settings based on the email addresses used for a transfer.
To verify a specific sender email address, you need to send a verification request for that email address to the Cryptshare Server. The Cryptshare Server will then send a verification email containing a verification code to the specified sender email address.
This verification code will then need to be stored in the **Client**'s verification store for the corresponding sender email address. Once this verification process is complete, the **Client** will then subsequently read the verification code for the sender email address from its verification store and include that data in each request it sends to the Cryptshare Server. The Cryptshare Server then checks the email address / verification code combination against its own database entry, to make sure the sender email address is indeed verified, before performing any requested operations.

Example: Performing a Sender Verification

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// now you can call the checkVerification() method, to first check the
// current verification status of the specified sender email address
VerificationStatus result = client.checkVerification();

if (result.isSenderVerification()) {
    // Verification Mode is indeed set to Sender Verification
    if (!result.isVerified()) {
        // Sender email address is NOT verified, so request a new verification code
        client.requestVerification();

        // Cryptshare Server will now have sent an email containing the new
        // verification code to the email address "sender_email@server.com".
        // So now you could, for instance, prompt the user of your application
        // to enter that code on the command line, or if your application
        // has access to the user's emails, you could automatically parse the
        // email for the verification code.
        // Let's ask the user to enter the code:
        System.out.println("Please type in the verification code:");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String vericode = reader.readLine();
        reader.close();

        // now tell the client to store the verification code in the
        // verification store
        client.storeVerification(vericode.trim());
    }
}
```

Related Knowledge Base Articles

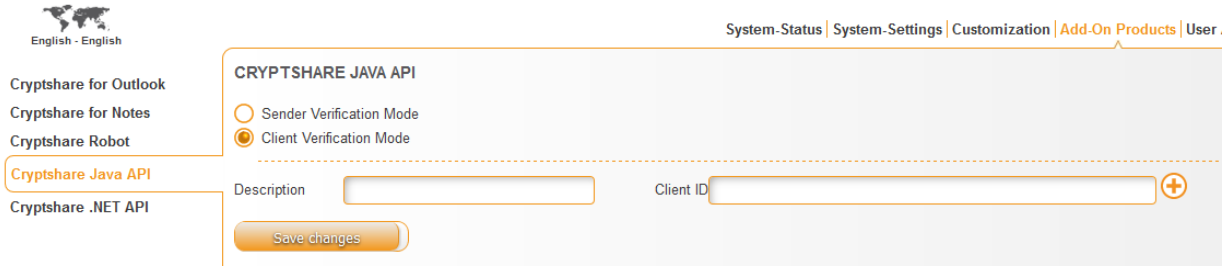
- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Cryptshare Java API Manual : Client Verification

Created by René Hartwig on Jul 05, 2018

If the Verification mode is set to Client Verification, any sender email address can be used to perform a transfer operation, without having to perform a verification process, as described under Sender Verification, for each individual email address. With Client Verification mode, the verification is only performed once for the **Client** instance as a whole, and then the **Client** instance can use any sender email address to perform its operations.
Please Note
Using Client Verification increases the risk of abuse, since anyone with access to the verified **Client** instance can perform Cryptshare transfers using any sender email address. It is therefore recommended that Client Verification mode only be used if Sender Verification mode does not meet your requirements.

To verify your **Client** instance, first make sure that the Verification mode configured on the Cryptshare Server is set to Client Verification. You can then call the requestVerification() method on the **Client** instance, which will return a new unique Client ID. You then have to register this Client ID on the Cryptshare Server administration page:



Enter a description for your **Client** instance and the new Client ID, then click the "+" button to add the new client verification entry. Don't forget to save the changes. You can now use the verified **Client** instance with any sender email address, without the need of any further verification.

Example: Performing a Client Verification

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
// Since we are only doing a client verification in this example, it
// does not really matter what sender email address you enter to create
// the client, as it will not be needed for any transfer operations.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// now you can call the checkVerification() method, to first check the
// current verification status
VerificationStatus result = client.checkVerification();

if (!result.isSenderVerification()) {
    // Verification Mode is indeed set to Client Verification
    if (!result.isVerified()) {
        // Client is NOT verified, so request a new verification
        String clientID = client.requestVerification();

        // When the requestVerification() method is called in Client Verification
        // mode, the client automatically creates a new Client ID and stores it
```

```
        // in its verification store. The generated Client ID will have also been
        // logged to the log file. This clientID has to then be registered on the
        // Cryptshare Server administration page.
    }
}
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Cryptshare Java API Manual : API Functions

Created by René Hartwig on May 23, 2018

This section describes what methods the Cryptshare Java API provides, and how to use them to perform the desired operations.
All methods are provided by the API's **Client** object, so in order to use any of the API's functionality, you will need to first create an instance of the **Client** class. The main purpose of the API is to provide you with a simple way to transfer files with the Cryptshare Server, so the creation of a **Client** instance always requires the sender's email address, the URL to the Cryptshare Server in the form of a **CryptshareConnection** instance and the verification data.

Creating the Client

The first step in creating the **Client** instance is to create a **CryptshareConnection**, initialized with your Cryptshare Server URL. The **CryptshareConnection** constructor requires a **WebServiceUri** object as its parameter, which is a class that takes your base server URL address and automatically creates the correct service URLs for the API to access the Cryptshare web services of your server. Once the **CrptshareConnection** instance is created, you can use it as a parameter to the **Client**'s constructor.
Besides the **CryptshareConnection** instance, the **Client**'s constructor also requires the sender's email address, as well as either the path to a verification store file, or the encrypted verification store bytes directly. The verification store contains the verification data for your client or sender addresses, depending on the Verification Mode configured on the Cryptshare Server. The verification store data is encrypted with your **Client Key** (see Client ID and Client Key). When specifying a store path, you can either write the full path of your verification store, including the file name, such as "*C:\temp\client.store*", or just the directory path, such as "*C:\temp*". If you only specify a directory path, the **Client** will look for a file named "*client.store*" in the specified directory. If the verification store file does not exist, it will be automatically created by the **Client**. If you pass in the encrypted store bytes directly to the **Client**'s constructor, no file will be created by the **Client**. The data bytes will simply be decrypted with your **Client Key** and only available in memory.
Here is an example of creating a **Client** instance by specifying a verification store file:

Example: Creating a Client with a Store Path

```
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");
```

And here is an example of creating a **Client** instance by specifying the encrypted verification store bytes directly:

Example: Creating a Client with Store Bytes

```
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Get the encrypted store bytes - the method getStoreBytes() is not part of the
// Cryptshare Java API. It is here only used as a placeholder for a method of
// your application, that provides the store bytes to use, for example by reading
```

```
// them in from a file or from a DB.
bytes[] storeBytes = getStoreBytes();

// Create the Client instance with the sender's email address, the
// CryptshareConnection, and the encrypted store bytes.
Client client = new Client("sender_email@server.com", connection, storeBytes);
```

Once the **Client** instance is created, you can call its methods to perform the desired service operations, as described in the respective sections of this manual. Please note that most service operations require a verified **Client** or sender address. How to verify your **Client** or sender address is described in the section Verification.

Client ID and Client Key

The **Client ID** identifies a particular computer client used to run an application which uses the Cryptshare Java API. The **Client ID** contains data which uniquely identifies the computer instance, such as the computers MAC addresses. The **Client ID** is created when the application is started for the first time and stored in the verification store for subsequent uses. The **Client ID** is needed for the communication with the Cryptshare Server.
If the verification mode on the Cryptshare Server is configured to be Client Verification, this **Client ID** will have to be used as the verified **Client ID** on the server.
The **Client Key** is, like the **Client ID**, based on the unique computer MAC addresses and is used to encrypt the verification store data. So when the **Client** opens its verification store, it will create the **Client Key** and use it to decrypt the store data.
Warning
Since the **Client Key** is based on the computer's MAC addresses, exchanging the network adapters in your computer will result in a new and different **Client Key**. You would then not be able to access the verification store anymore, since it was encrypted with the old key, and any verification data you had previously saved in the store would be lost. The Cryptshare Server would see the **Client** again as unverified, and you would have to repeat the verification process again. To avoid that, you may want to consider changing the MAC address of the new network card to the old MAC address, if possible.

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop

- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : General Server Data

Created by René Hartwig, last modified on May 23, 2018

With the **Client**'s method **requestServerData()** you can request general server data from the Cryptshare Server. The method returns a **ServerData** object which contains the official base URL of the Cryptshare Server, the server's time zone, as well as the disk spaces in bytes that are available for future file transfers.

Example: Requesting General Server Data

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Request general server data
ServerData serverData = client.requestServerData();

System.out.println("Official Cryptshare Server URL: " + serverData.getServerUrl());
System.out.println("Server time zone: " + serverData.getServerTimeZone());
System.out.println("Available disk space for uploaded transfers (retention folder) in
bytes: "
                                        + serverData.getAvailableRetentionDiskSpace());

System.out.println("Available disk space for transfers during upload (temporary folder)
in bytes: "
                                        + serverData.getAvailableTempDiskSpace());
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer

- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : License Information

Created by René Hartwig, last modified on May 23, 2018

The license information of the current Cryptshare Server installation and the Add-On Product's own license information can be queried from the server using the **Client**'s **requestLicenseInfo()** method. The method returns a **LicenseInfo** object containing the license status of the Cryptshare Server and the Add-On Product, as well as the respective license and subscription expiration dates in the format "YYYY-MM-DD".

Example: Requesting the License Information

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Request the License Information
LicenseInfo licenseInfo = client.requestLicenseInfo();

System.out.println("Server license expiration date: " +
                        licenseInfo.getServerLicenseExpirationDate());
System.out.println("Server subscription expiration date: " +
                        licenseInfo.getServerSubscriptionExpirationDate());
System.out.println("Is server license valid? " + licenseInfo.isServerLicenseValid());
System.out.println("Product license expiration date: " +
                        licenseInfo.getProductLicenseExpirationDate());
System.out.println("Product subscription expiration date: " +
                        licenseInfo.getProductSubscriptionExpirationDate());
System.out.println("Is product license valid? " + licenseInfo.isProductLicenseValid());
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching

- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : Password Functions

Created by René Hartwig, last modified on May 23, 2018

When doing a Cryptshare Transfer, you can specify a password that will be required to access the transferred files. However, this password needs to fulfill the requirements of the Password Policy set up on the Cryptshare Server.
You can have the Cryptshare Server check a password to make sure that it fulfills the requirements of the Password Policy or you can have the Cryptshare Server generate a password for you.

Requesting a Server Generated Password

You can request a new password from the Cryptshare Server by calling the **Client**'s method **requestPassword(int)**. In order to use this method, the sender address or the **Client** instance will have to be verified (see Verification). The method requires the desired length of the new password as a parameter and returns a new password of the given length that is compatible with the Password Policy set up on the Cryptshare Server. You can then use this password as the password for a Cryptshare File Transfer operation. If the specified length is less than the required minimum length defined in the Password Policy, then the minimum length will be used.

Example: Requesting a Password

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Assuming that either the sender address or the client is already verified,
// we can now request a new password of length 8, for example.
String password = client.requestPassword(8);
```

Validating a Password

If you would like to choose your own password for a Cryptshare File Transfer, then you can have that password validated by the Cryptshare Server to make sure that it is compatible with the server's Password Policy. Otherwise, you will get an error when trying to initiate a Cryptshare File Transfer with an invalid password. The **Client**'s method **checkPassword(String)** takes the password to check as a parameter and returns a **PasswordPolicy** instance, containing the result of the Password Policy check for the given password.

Example: Requesting a Password

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
```

```
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Assuming that either the sender address or the client is already verified,
// we can now request to have our password, for instance 'password123' validated.
String passwordToCheck = "password123";
PasswordPolicy passwordPolicy = client.checkPassword(passwordToCheck);

// The security strength of the given password as a float in the range from
// 0.0f - 1.0f with 1.0f being most secure.
System.out.println("password security = " + passwordPolicy.getSecurity());

// The PasswordPolicy object also contains the results of the test for the
// given password
System.out.println("Is password too short?: " + passwordPolicy.isTooShort());
System.out.println("Is password too long?: " + passwordPolicy.isTooLong());
System.out.println("Does password not fulfil all requirements?: " +
                           passwordPolicy.isInsufficientCharacteristics());
System.out.println("Does password have insufficient number of digits?: " +
                           passwordPolicy.isInsufficientDigits());
System.out.println("Does password not have enough alphabetic characters?: " +
                           passwordPolicy.isInsufficientAlphabetical());
System.out.println("Does password not have sufficient special characters?: " +
                           passwordPolicy.isInsufficientSpecial());
System.out.println("Does password not have enough uppercase letters?: " +
                           passwordPolicy.isInsufficientUpper());
System.out.println("Does password not have enough lowercase letters?: " +
                           passwordPolicy.isInsufficientLower());
System.out.println("Does password contain illegal whitespace?: " +
                           passwordPolicy.isIllegalWhitespace());
System.out.println("Is the password an illegal word?: " +
                           passwordPolicy.isIllegalWord());
System.out.println("Does the password contain an illegal sequence?: " +
                           passwordPolicy.isIllegalSequence());
System.out.println("Does the password contain an illegal repetition?: " +
                           passwordPolicy.isIllegalRepetition());

// The PasswordPolicy object also contains the configured settings of the
// password policy to see how the policy is configured on the server
System.out.println("Minimum password length: " + passwordPolicy.getMinimumLength());
System.out.println("Maximum password length: " + passwordPolicy.getMaximumLength());
System.out.println("Must a password contain digits?: " +
                           passwordPolicy.isMustContainDigits());
System.out.println("Must a password contain characters?: " +
                           passwordPolicy.isMustContainChars());
System.out.println("Must a password contain special characters?: " +
                           passwordPolicy.isMustContainSpecialChars());
System.out.println("Must a password be upper/lower case?: " +
                           passwordPolicy.isMustBeUpperLowerCase());
System.out.println("Will a password that can be found in a dictionary be declined?: " +
                           passwordPolicy.isDictionaryDeclined());
System.out.println("Will a password containing a character repetition be declined?: " +
                           passwordPolicy.isCharRepetitionsDeclined());
System.out.println("Are whitespaces in a password allowed?: " +
                           passwordPolicy.isAllowWhitespaces());
System.out.println("Are alphabetical sequences in a password allowed?: " +
                           passwordPolicy.isAllowAlphabeticalSequence());
System.out.println("Are numeric sequences in a password allowed?: " +
                           passwordPolicy.isAllowNumericSequence());
System.out.println("Are QUERTY sequences in a password allowed?: " +
                           passwordPolicy.isAllowQwertySequence());
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Cryptshare Java API Manual : E-Mail Notifications

Created by René Hartwig, last modified on May 23, 2018

Disabling e-mail notifications on server-side

Usually the Cryptshare Server manages all e-mail communication between the participants of a transfer. However, when developing an own client application for Cryptshare communication it might be desirable to send these notifications from within this client.
Therefore Cryptshare allows it to disable the sender and the recipient notification for each transfer.

Disabling the sender notification for a transfer

```
Transfer transfer = new Transfer();
transfer.setNotifySender(false);
```

Disabling the recipient notification for a transfer

```
Transfer transfer = new Transfer();
transfer.setNotifyRecipients(false);
```

Requesting e-mail templates from the Cryptshare Server

Please also consider the documentation for Cryptshare Language Packages in the Cryptshare Server Documentation.

Developers of a Cryptshare Client Application might want to handle transfer notifications themselves, for instance for an e-mail integration such as Cryptshare for Outlook. In order to still have the same layout and especially the unique information per recipient the API offers the possibility to request the HTML code for the e-mail notification from the server.

```
client.requestMailTemplate("<template-name>", <replacements>, <language>, <mailFormat>);
```

<template-name> : The name of the desired e-mail template
<replacements> : Mapping of the placeholders in the template
<language> : The language of the template
<mailFormat> : The mail format which shall be used. This can either be '**HTML**' or '**plain**'

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching

- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

E-Mail Placeholders

E-Mail placeholders are used to fill in individual information into the templates. This way each e-mail recipient can retrieve an e-mail with unique content, such as the personal download link, name or e-mail address.

Template Snippet with name and e-mail placeholders

```
[...]
<p>Dear Sir or Madam,</p>

<p>Confidential data has been sent to you by <a href="mailto:$email">$name</a>.
[...]
```

Possible placeholders in e-mail notifications

The following table shows available placeholders in e-mail templates. Placeholders tagged with `SENDER` are specifically required for the sender template. Placeholders tagged with `RECIPIENT` are specifically required for the recipient template.

Tag	Placeholder Key	Description	Additional Notes
SENDER RECIPIENT	`TemplatePlaceholder.BASEURL`	The Cryptshare Server URL	i.e. https://server.url.com
RECIPIENT	`TemplatePlaceholder.SUBJECT`	The message subject	
RECIPIENT	`TemplatePlaceholder.MESSAGE`	The body of the message	
SENDER RECIPIENT	`TemplatePlaceholder.DATE`	The expiration date of the transfer	
SENDER RECIPIENT	`TemplatePlaceholder.LIST`	The list of the files in the transfer	

RECIPIENT	`TemplatePlaceholder.LINK`	The download link for the transfer	
SENDER RECIPIENT	`TemplatePlaceholder.PASSWORDMODE`	The type of password mode used for the transfer	Possible values: • manual • generated • none
RECIPIENT	`TemplatePlaceholder.NAME`	The name of the sender	
RECIPIENT	`TemplatePlaceholder.PHONE`	The phone number of the sender	
RECIPIENT	`TemplatePlaceholder.EMAIL`	The e-mail address of the sender	
SENDER RECIPIENT	`TemplatePlaceholder.LIST_1`	List of the recipients addressed in 'To'	
SENDER RECIPIENT	`TemplatePlaceholder.LIST_2`	List of the recipients addressed in 'Cc'	

Please note that the above table only shows the default configuration for Cryptshare E-Mails. Any placeholder can also be used for a different purpose in own templates. Please read the respective information in the Cryptshare Server Manual for further details.

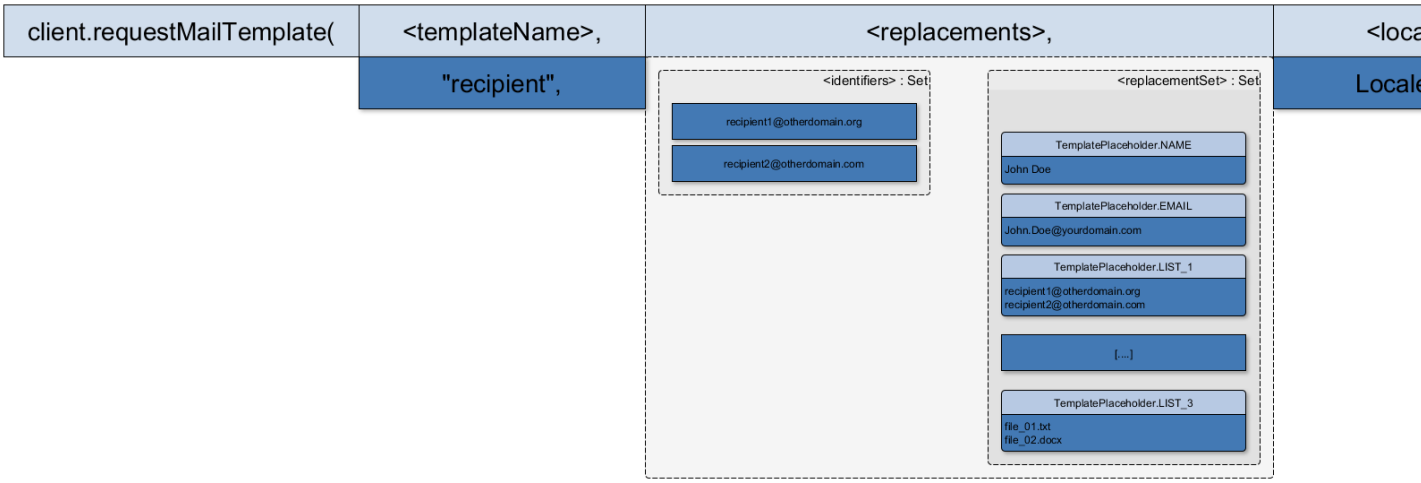
How to use e-mail placeholders

#client.requestMailTemplate() requires a very specific replacement-map in order to get the desired result. The parameter is a set holding instances of type *TemplateReplacement*. A *TemplateReplacement* object is basically a key-value wrapper where the key is a specific placeholder enum and the value can either be a list or a single value. This allows it, to retrieve customized templates per recipient with a single request.

Using placeholders

```
// <replacements>
Map<Set<String>, Set<TemplateReplacement>> replacements = new HashMap<Set<String>,
Set<TemplateReplacement>>();
// <replacementSet>
Set<TemplateReplacement> replacementSet = new HashSet<TemplateReplacement>();
replacementSet.add(new TemplateReplacement(TemplatePlaceholder.NAME, "John Adams"));
replacementSet.add(new TemplateReplacement(TemplatePlaceholder.EMAIL,
"john.adams@server.com"));

List<String> fileList = new ArrayList<String>();
fileList.add("file_01.txt");
fileList.add("file_02.docx");
replacementSet.add(new TemplateReplacement(TemplatePlaceholder.LIST_3, fileList));

List<String> recipientList = new ArrayList<String>();
recipientList.add("recipient1@otherdomain.org");
recipientList.add("recipienr2@otherdomain.com");
replacementSet.add(new TemplateReplacement(TemplatePlaceholder.LIST_1, recipientList));

// <identifiers>
Set<String> identifiers = new HashSet<String>();
identifiers.addAll(recipientList);
replacements.put(identifiers, replacementSet);

Map<List<String>, String> mailTemplates = client.requestMailTemplate("recipient",
```

```
replacements, new Locale("ja"), "html");
```



<templateName> : The name of the desired e-mail template
<replacements> : Mapping between <identifiers> and <replacementSet>
<identifiers> : Set of e-mail addresses which have a specific <replacementSet>
<replacementSet> : A set of placeholders for <identifiers>

The result object of an e-mail template request

When requesting an e-mail template using *#client.requestMailTemplate()* the returned mapping contains a unique template per identifier-set. This means with one request a filled in template can be returned per single recipient or for all recipients together or for a combination of both.

Example: multiple <identifier>-<replacementSet> combinations

```
// identifiers1: One single recipient, replacementSet1: Set of replacements
replacements.put(identifiers1, replacementSet1);
// identifiers2: Two recipients, replacementSet2: Set of replacements
replacements.put(identifiers2, replacementSet2);
Map<List<String>, String> mailTemplates = client.requestMailTemplate("recipient",
replacements, new Locale("ja"), "html");
```

The above request would return two templates, one for each set of identifiers put into the replacement map both having a different content, meaning the placeholders have been filled in with the contents of *replacementSet1*, respectively *replacementSet2*.

Cryptshare Java API Manual : Language Resources

Created by René Hartwig, last modified on May 23, 2018

The Cryptshare Server can provide the **Client** with the language resources contained in the Language Packages that have been installed on the server. These Language Packages can then be used for a user interface on client side, for instance.

Available Languages

You can check what languages are available on the server using the **Client**'s method **requestLanguagePacks()**. It will return a list of **LanguagePack** objects containing information about each installed language pack. You can then use the data from this list to request a specific language pack file for download.

Example: Requesting a List of Available Languages

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Now we can request the list of installed languages
List<LanguagePack> languagePackList = client.requestLanguagePacks();

for (LanguagePack languagePack : languagePackList) {
    Locale locale = languagePack.getLocale();
    String version = languagePack.getLanguagePackVersion();
    long lastUpdate = languagePack.getLastUpdate();
    System.out.println("Language pack language = " + locale.getLanguage() +
            " with version = " + version + " last updated at " + lastUpdate);
}
```

Downloading a Language Pack File

Now that you know which Language Packages are available on the server, you can download a specific language pack file using the **Client**'s method **requestLanguagePackFile(String,Locale)**. The method requires the base name of the actual language pack file on the server as the first parameter and the desired Locale as the second parameter.

Language Pack Base Name

The base name of the actual language pack file on the server is just the file name, minus the language or country information. For example, if you have the language pack file "lang_en_GB.xml" installed on the server, the base name for that language pack file would be "lang.xml". So you would request this particular language pack file, using the following method call:

```
Locale locale = Locale.UK;
byte[] langFileBytes = client.requestLanguagePackFile("lang.xml", locale);
```

Here is a complete example of how to request a specific language pack file:

Example: Downloading a Language Pack File

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Get the list of available languages
List<LanguagePack> languagePackList = client.requestLanguagePacks();

// Now we can request the download of a specific language pack file, where
// the first parameter of the method is the base name of the physical language
// pack file on the server, and the second parameter is the desired Locale.
// For this example, we will just download the first language pack from the
// list of available languages.
 byte[] langFileBytes = client.requestLanguagePackFile("lang.xml",
                                        languagePackList.get(0).getLocale());

// save the file to the current directory on our local disk
if (langFileBytes != null) {
    FileOutputStream outStream = null;
    try {
        outStream = new FileOutputStream("lang.xml");
        outStream.write(langFileBytes);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    finally {
        try {
            outStream.close();
        }
        catch (Exception e) {}
    }
}
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
-

- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : Policy Rules

Created by René Hartwig, last modified on May 23, 2018

Cryptshare uses Policy Settings that can be defined on the server side to allow or deny usage of the system to certain senders and recipients and to control the transfer options pertaining to specific senders and recipients. You can request the policy rules configured for a specific sender/recipients combination by calling the **Client**'s method **requestPolicy(List<String>)**. The method takes a list of recipient email addresses and returns a **Policy** object containing the resulting policy rules. Please note that the Cryptshare Server only allows this operation for verified sender addresses or verified **Client** instances (see Verification).

Example: Querying the Policy Settings

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// The list of recipient email addresses which this sender wants to send
// files to.
List<String> recipients = new ArrayList<String>();
recipients.add("john.smith@abc.com");
recipients.add("jane.adams@xyz.com");
Policy policy = client.requestPolicy(recipients);

// The returned Policy object contains the rules defined for transfers
// between this sender ("sender_email@server.com") and the specified
// recipients ("john.smith@abc.com","jane.adams@xyz.com").
System.out.println("Will the sender be notified of downloads? " +
                policy.isDownloadNotification());
System.out.println("Will the file names be shown in the email message? " +
                policy.isShowFilenames());
System.out.println("Is sending of confidential message allowed?: " +
                policy.isAllowConfidentialMessage());
System.out.println("The storage duration of the transferred files in days: " +
                policy.getStorageDuration());
System.out.println("The maximum total size of the transfer in MB: " +
                policy.getTransferLimit());
System.out.println("Recipients not allowed for this sender by the policy" +
                policy.getFailedAddresses());
System.out.println("The allowed password modes for a transfer: " +
                policy.getPasswordMode());
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer

- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : Terms of Use

Created by René Hartwig, last modified on May 23, 2018

The Terms of Use that are defined under the Legal tab on the Cryptshare Server administration website can be downloaded by the **Client**, so that they can be presented to a user in a client side application. The Terms of Use can be downloaded using the **Client**'s method **requestTermsOfUse()**. The method returns a **Map<String,String>** containing the Terms of Use texts as the values, and their corresponding language codes as the keys.

Example: Requesting the Terms of Use

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Request the Terms of Use
Map<String,String> terms = client.requestTermsOfUse();

for (Entry<String,String> entry : terms.entrySet()) {
    System.out.println("Terms of use for language '" + entry.getKey() + "': " +
entry.getValue());
}
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in

- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : File Transfer

Created by René Hartwig, last modified on May 23, 2018

You can perform both synchronous, as well as asynchronous Cryptshare file transfers using the API. A transfer can contain one or more files that will be encrypted, uploaded to the Cryptshare Server, and then made available to the transfer's recipients for download. Depending on your transfer settings, the recipients and/or sender of the transfer will then be notified by email when the transfer is complete and the files are available for download.
Before you can perform a transfer, you have to configure the transfer settings, define the recipients and select the files you wish to include in the transfer (see Preparing a Transfer). Then you can either Perform a Synchronous Transfer or Perform an Asynchronous Transfer.

Querying Default Server Transfer Settings

The default transfer settings, such as the maximum storage duration of the files and the maximum allowed transfer size, are configured on the Cryptshare Server and apply to all transfers. You can query these settings using the Client's method requestTransferData(). This method returns a TransferSettings object, containing the maximum file storage duration and the maximum transfer size in MB. These settings are the default server settings for transfers. You can specify different values for specific sender/recipient combinations by changing the Policy Settings.

Example: Querying Transfer Settings

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Request the general transfer settings
TransferSettings settings = client.requestTransferData();

System.out.println("Maximum file storage duration = " +
transferSettings.getStorageDuration());
System.out.println("Maximum transfer size in MB = " +
transferSettings.getTransferLimit());
```

Preparing a Transfer

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer

- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook
- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Before you can perform a transfer, you have to create a **Transfer** object with the sender's contact details, the password mode, the message that is to be included in the transfer, the recipients, the languages of the notification emails, the storage duration, and, of course, the files that will be transferred.
If a recipient is not allowed for the transfer, based on the server's policy settings, an exception will be thrown when trying to perform the transfer. To avoid this, you can check the Policy Rules before attempting to perform a transfer.
The password mode used for the transfer must be one of the allowed password modes defined in the Policy Rules.
About the Password Modes
If the password mode is **PasswordMode.MANUAL**, then you have to explicitly set a password for the transfer. You can check the password to make sure that it fulfills the requirements of the Password Policy by using the functions described in Password Functions. The recipient will then have to contact the sender of the transfer to find out what the password is, before any files can be downloaded.
For password mode **PasswordMode.GENERATED**, the Client will request a server-generated password and automatically set it for the transfer. The password will be included in the transfer message, so that the recipients do not need to explicitly contact the sender to learn the password.
When using **PasswordMode.NONE**, you do not have to define a password. The server will automatically generate a password and include it in the download link, so that the recipient does not need to explicitly enter a password to download the files. This is obviously the least secure method, and should thus be used sparingly.
You can specify the languages to use for the default recipient and sender email notifications. The languages are specified with their language code, e.g. "en" or "en_GB". You can only specify languages for which there are language packs installed on the server. See Language Resources.
You can also set a custom message and subject for the notification emails. The **Transfer** object has setters for the **message** and **subject** fields that take either a String, or an InputStream as a parameter, so you can either directly set a text as the message or subject, or give the setter method an InputStream containing the text, and the text will be automatically read from the stream. The encoding of the stream has to be UTF-8, to ensure that the text can be read in correctly. Once the text is read from the InputStream, the stream will automatically be closed. Your message text can also include HTML markup. For the subject text, only plain text is supported, so using HTML tags in the subject line may lead to unexpected results.
If allowed by the Policy, you may also choose to include a confidential message to the recipients of the transfer. Setting the confidential message and subject is similar to setting the custom notification message on the **Transfer** object. The parameters for the setter methods of the **Transfer** object's **confidentialMessage** and **confidentialSubject** fields are either a String or an InputStream, so the same rules apply as described above for the custom message and subject.
When specifiying an expiration date for the transferred files, make sure that it is still within the maximum allowed storage duration as specified in the Policy Rules. The transferred files will only be available for download until the specified

expiration date.
Once the **Transfer** object has been initialized as described in the example here, it can be used for performing the transfer either synchronously or asynchronously as described in the sections Perform a Synchronous Transfer and Perform an Asynchronous Transfer below.

Example: Preparing a Transfer Object

```
// Create a new Transfer object
Transfer transfer = new Transfer();

// Set the name of the sender
transfer.setSenderName("John Adams");

// Set the sender's phone number
transfer.setSenderPhone("234 5467");

// Set the message text of the mail that is to be included in the transfer.
// The Transfer object's setMessage(..) method takes either an
// actual text string containing the email message, or an InputStream
// which contains the email message text. If specifying an InputStream, the
// stream needs to be UTF-8 encoded, to ensure that the characters can be
// read in and displayed correctly in the email message.
try {
    // To illustrate, we'll use the message text from an input stream.
    // Your application may get this input stream from another process, for
    // instance, but in this example, we will just read it in from a file
    FileInputStream inputStream = new FileInputStream("C:\\temp\\message.txt");
    // Set the input stream as the message. The method will read in
    // the text from the stream and automatically close the stream when it's done.
    transfer.setMessage(inputStream);
}
catch (Exception ex) {
    // there was an error reading the text file, so show an error message
    System.err.println("Error reading the message file!");
}

// Set the subject text of the mail that is to be included in the transfer.
transfer.setSubject("Subject of the Transfer");

// Define the recipients
List<String> recipients = new ArrayList<String>();
recipients.addRecipient("jane.doe@abc.com");
recipients.addRecipient("jack.smith@xyz.com");

// Get the policy rule from the server for the given recipients, so we can
// check to make sure they are allowed
Policy policy = client.requestPolicy(recipients);

// Check to make sure the recipients are allowed and only add them
// to the transfer, if they are
if (policy.getFailedAddresses() != null && !policy.getFailedAddresses().isEmpty()) {
    for (String recipient : recipients) {
        if (!policy.getFailedAddresses().contains(recipient)) {
            transfer.addRecipient(recipient);
        } else {
            System.out.println("The recipient is invalid: " + recipient);
        }
    }
} else {
    // all recipients are valid
    transfer.addRecipients(recipients);
}

// If allowed by the policy, we can also send a confidential message to the
```

```
// recipients
if (policy.isAllowConfidentialMessage()) {
    transfer.setConfidentialSubject("Subject of the confidential message");
    transfer.setConfidentialMessage("This is the text of the confidential message.");
}

// Only continue if we have at least one valid recipient for the transfer
if (transfer.getRecipients() == null || transfer.getRecipients().isEmpty()) {

    throw new Exception("No valid recipients defined, aborting transfer.");
}
// Set the password mode that will be used for this transfer.
// Must be one of the password modes allowed by the policy
// We will just pick the most secure mode allowed by the policy
String allowedPasswordModes = policy.getPasswordMode().toString();
if (allowedPasswordModes.contains(PasswordMode.GENERATED)) {
    passwordMode = PasswordMode.GENERATED;
} else if (allowedPasswordModes.contains(PasswordMode.MANUAL)) {
    passwordMode = PasswordMode.MANUAL;
} else {
    passwordMode = PasswordMode.NONE;
}
```

Performing a Transfer

Synchronous

Once you have the **Transfer** object initialized as described in the example Preparing a Transfer above, you can use it to perform a synchronous transfer. A synchronous transfer is performed by calling the **Client**'s method **performTransfer(Transfer, TransferUploadListener)**. The method requires a **Transfer** object containing all the necessary transfer data, as well as a **TransferUploadListener** that will be informed about the upload progress and any errors encountered during the upload. As this is a synchronous operation, the method will block until all files in the transfer have been uploaded and the transfer is completed.

Example: Performing a Synchronous Transfer

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Prepare the transfer object as described in the example above
Transfer transfer = ...

// Perform a synchronous transfer with a new anonymous TransferUploadListener
instance.
// Method will block until the transfer is completed.
client.performTransfer(transfer, new TransferUploadListener() {
    @Override
    public void handleUploadProgressChanged(ProgressChangedEvent evt) {
        // This method is called repeatedly with the current upload data
        // Our upload listener will just output the current progress to the console
        double percent = (((double) evt.getBytesUploaded() / evt.getBytesTotal()) *
100.0);
        System.out.println("Transfer progress ... " + ((int)percent) + "%");
    }

    @Override
    public void handleUploadCompleted(UploadCompletedEvent evt) {
        // this method is called when all files of the transfer have been uploaded
        System.out.println("Upload completed!");
    }

    @Override
    public void handleUploadInterrupted(UploadInterruptedEvent evt) {
        // this method is called when an exception occurs during the file upload
        System.out.println("An exception occurred during the upload: " +
evt.getException());
```

```
    long streamSize = inputFile.length();
    // Create the TransferFile object for this stream. You also need to specify
    // a name for the file that will be created from the data of this stream,
including
    // a file extension, so that the recipient will be able to open it with the
```

```
    }

    @Override
    public void handleTransferCanceled() {
        // this method is called when the transfer has been cancelled
        // using the cancelTransfer() method
        System.out.println("The transfer has been canceled!");
    }
});
```

Asynchronous

To perform a transfer asynchronously, initialize the **Transfer** object as described in the example Preparing a Transfer above, and call the **Client**'s method **beginTransfer(Transfer, TransferUploadListener)**, giving it the initialized **Transfer** object containing all the necessary transfer data, as well as a **TransferUploadListener** that will be informed about the upload progress, the completion of the upload, and any errors encountered during the upload. The method will return immediately after starting a new Thread in the background which will perform the actual transfer. Once the upload completes in the background, the **TransferUploadListener**'s **handleUploadCompleted(UploadCompletedEvent)** method will be called.

Cancelling a Transfer

If you wish to cancel an ongoing transfer, you may do so by calling the **Client**'s method **cancelTransfer()**. This method stops the current file upload process and cancels the transfer. A transfer can only be cancelled while it is still in the process of uploading the files. Once all files of a transfer have been uploaded to the server, the transfer will be complete and cannot be cancelled using the API. A transfer that has already been completed from the **Client**'s point of view, can only be cancelled on server side by the Cryptshare system administrator.

Example: Performing an Asynchronous Transfer

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the sender's email address,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("sender_email@server.com", connection, "C:\temp");

// Prepare the transfer object as described in the example above
Transfer transfer = ...

// Perform an asynchronous transfer with a new anonymous TransferUploadListener
instance
// Method will return immediately
client.beginTransfer(transfer, new TransferUploadListener() {
    @Override
    public void handleUploadProgressChanged(ProgressChangedEvent evt) {
        // This method is called repeatedly with the current upload data
        // Our upload listener will just output the current progress to the console
        double percent = (((double) evt.getBytesUploaded() / evt.getBytesTotal()) *
100.0);
        System.out.println("Transfer progress ... " + ((int)percent) + "%");
    }

    @Override
    public void handleUploadCompleted(UploadCompletedEvent evt) {
        // this method is called when all files of the transfer have been uploaded
```

```
        System.out.println("Upload completed!");
    }

    @Override
    public void handleUploadInterrupted(UploadInterruptedEvent evt) {
        // this method is called when an exception occurs during the file upload
        System.out.println("An exception occurred during the upload: " +
evt.getException());
    }

    @Override
    public void handleTransferCanceled() {
        // this method is called when the transfer has been cancelled
        // using the cancelTransfer() method
        System.out.println("The transfer has been canceled!");
    }
});
```

Cryptshare Java API Manual : Transfer Polling

Created by René Hartwig, last modified on May 23, 2018

Transfer Polling

You can use the Java API Client's method **requestActiveTransfers()** to obtain a list of all active transfers that have been sent to the client's email address. The client has to be verified for this method to succeed. The method will return a map, with the transfer meta IDs as keys, and the corresponding download urls (without the password) as values. A download url will have the format "https://cryptshare.server.com/download1.php?id=33d03d8d6b", where the id parameter is the transfer meta id.

Example: Request active transfers

```
// First create the Client instance
// Create a WebServiceUri for your Cryptshare Server
WebServiceUri serviceUri = new WebServiceUri("https://cryptshare.server.com");

// Create a CryptshareConnection instance for your WebServiceUri
CryptshareConnection connection = new CryptshareConnection(serviceUri);

// Create the Client instance with the email address of the recipient, for which you
want to get the transfers,
// the CryptshareConnection, and the path to the verification store.
Client client = new Client("John.Doe@server.com", connection, "C:\temp");

// Request the active transfers for this client
// Returns a Map containing all transfer ids and download urls for all active transfers
that were sent to this client's
// email address (in this example: "John.Doe@server.com").
// This method will throw an exception if the client/email address has not been
verified.
Map<String, String> transferIdMap = client.requestActiveTransfers();

for (Entry<String, String> entry : transferIdMap.entrySet()) {
    System.out.println("Entry in transfer id map: metaId = " + entry.getKey() + " url =
" + entry.getValue());
}
```

Related Knowledge Base Articles

- KeyNotFoundException after confirming the transfer dialog of a Cryptshare transfer
- Error message "System Error. Code: 18." when performing Cryptshare transfer
- Error message "The path is not of a legal form." when performing Cryptshare transfer
- Performance issues with Outlook when using Cryptshare for Office 365 & Outlook
- Cryptshare for Outlook does not recognize the incoming verification e-mail
- Exception with message "LoadServerSettingsOnStartupAsync" is thrown when launching Outlook

- Office application crashes shortly after launching
- Outlook crashes when switching to a certain folder
- Error message "MAPI_E_INVALID_PARAMETER" when user performs a transfer
- Error message "Some components of Cryptshare for Outlook V2 don't support [...]" when launching Outlook with the Add-in
- Several "Get List Of All User Account Exception" errors in the log file
- AccessViolationException when adding attachments via drag and drop
- Shared mailbox is not recognized by the Add-in
- User receives "System.MissingMethodException" exception when launching an Office application
- The Cryptshare Server becomes unresponsive under high load.

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : API Java Doc

Created by René Hartwig on May 23, 2018

Befine Solutions AG - Schwarzwaldstr. 151 - 79102 Freiburg - GERMANY
Technical Support: Phone +49 761 389 13 100 - E-Mail: support@cryptshare.com

Cryptshare Java API Manual : Example Project

Created by Simon Erhardt, last modified by René Hartwig on Oct 30, 2018

General

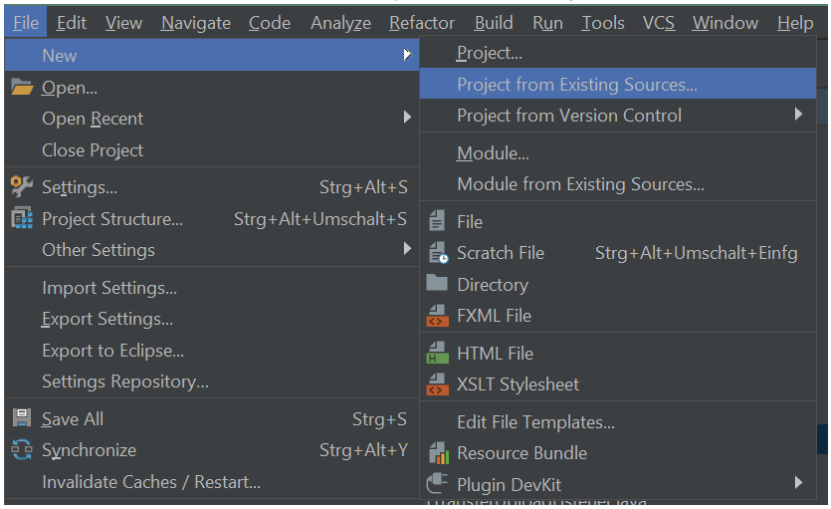
This page contains a simple Maven Java Project showing how the Java API for Cryptshare can be used.
Java API Version 3.0.0
Please note, that the Maven setup is configured to use the API version 3.0.0. Make sure you have the corresponding API version installed in the used repository.

How to import the project to your favorite IDE

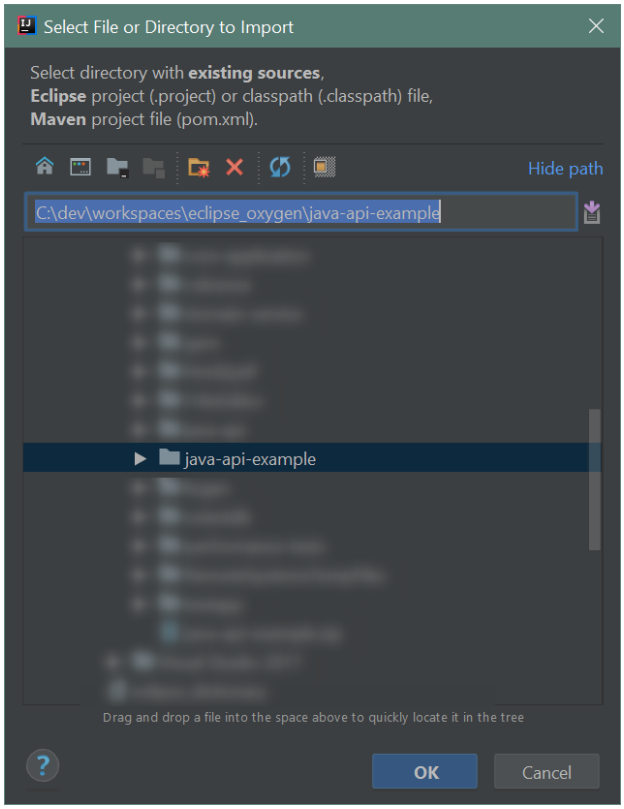
IntelliJ

Click here to show the instructions for importing this project into IntelliJ

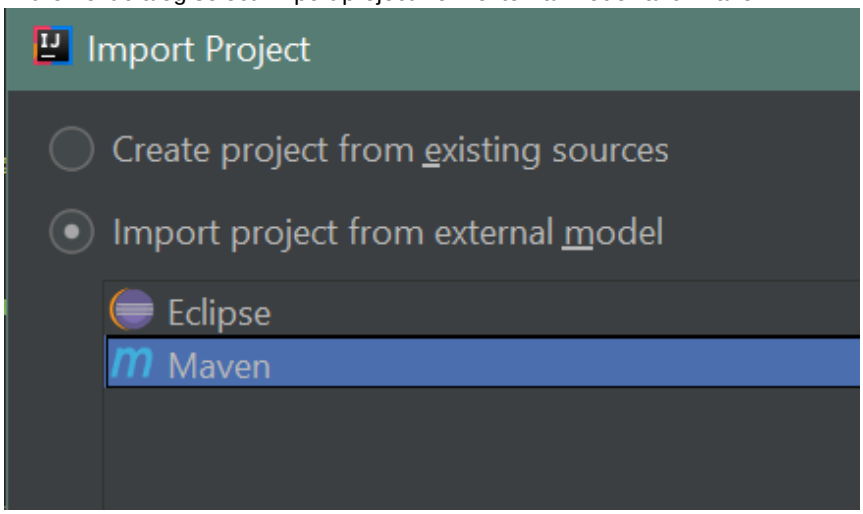
1. Download the project zip from this page and unpack it to your workspace.
2. In IntelliJ select 'File' → 'New' → 'Project from Existing Sources'



3. In the next dialog select the example-project directory.



4. In the next dialog select 'Import project from external model' and 'Maven'.

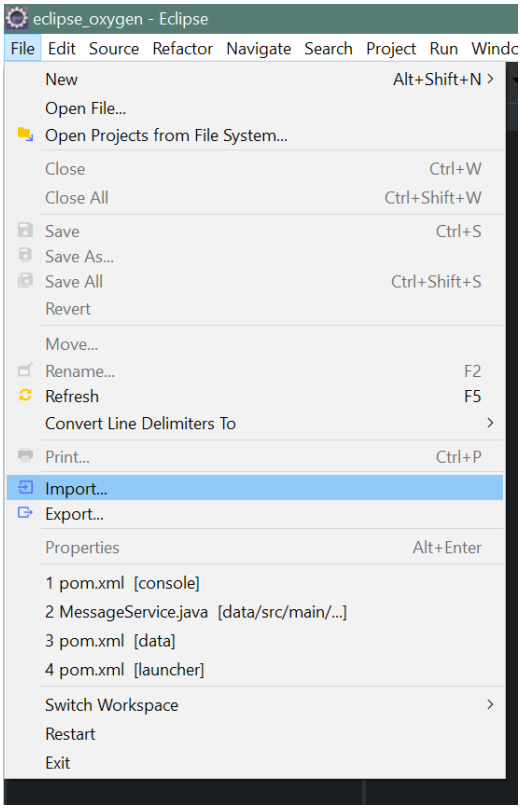


5. Follow the remaining steps of the wizard by clicking 'Next'.

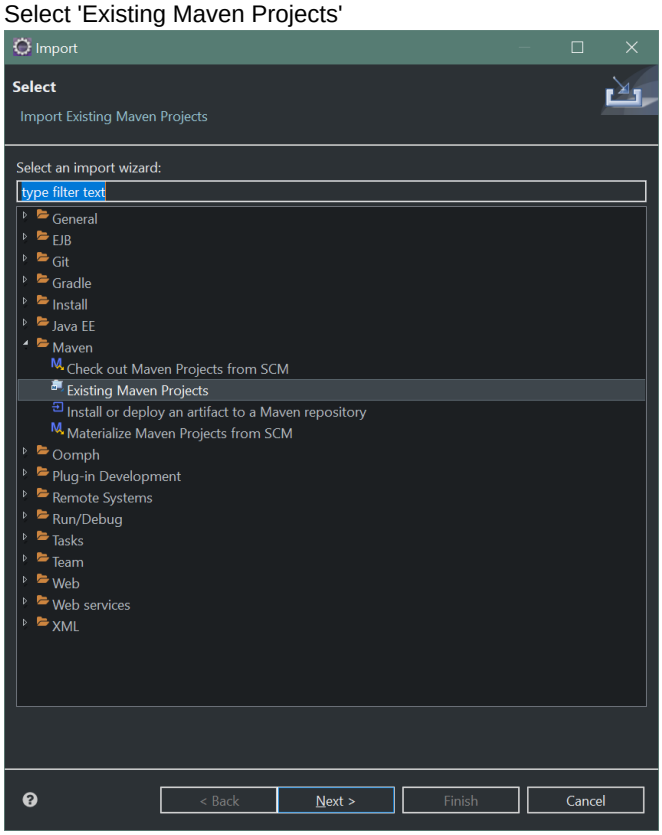
Eclipse

Click here to show instructions for importing this project into Eclipse

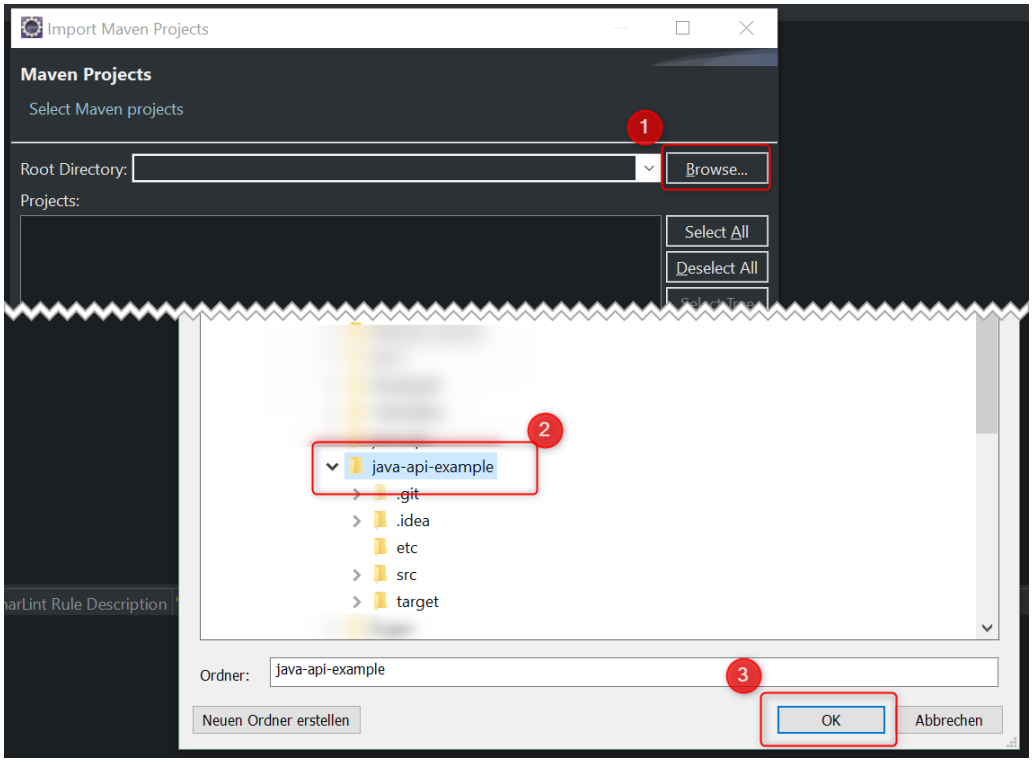
1. Download the project zip from this page and unpack it to your workspace.
2. In Eclipse select 'File' → 'Import'



3. Select 'Existing Maven Projects'



4. Click on 'Browse', select the project directory and click 'OK'



5. Click on 'Finish'

File	Modified
ZIP Archive java-api-example.zip	Aug 03, 2018 by Simon Erhardt

Cryptshare Java API Manual : Compatibility

Created by Dirk Riesterer on Oct 30, 2018

On this page you will find an overview with which Cryptshare server versions the Cryptshare Java API is compatible to.

Cryptshare Server Version	Java API V2	Java API v3.0
3.12.0	✔	✔
3.12.1	✔	✔
4.0.0	✔	✔
4.1.0	✔	✔
4.1.1	✔	✔
4.1.2	✔	✔
4.1.3	✔	✔

Related Knowledge Base Articles

Content by label
There is no content with the specified labels