TrustFactor Java SDK User Manual

Contents

Release Notes

This document is the user manual for the Java SDK.

Changelog - 7.2.2

- Changed `contractRevalidationDataSign` function input parameter from `String` to `HashMap<String, ?>`

Changelog - 7.2.1

- Adds a method to sign an arbitrary string
  - Method is called `contractRevalidationDataSign` on Client object
- Added new APIs:
  - `getContractDetailsByKey`
  - `getContractDetailsByUserID`

Changelog - 7.2.0

- Added new APIs to improve Trustfactor integration:
  - `getAssociationsContractsDevicesV2`
  - `getContractsWithDevices`
  - `getContractDevicesByKeyV2`
  - `getContractDevicesByUserIDV2`
  - `registerCodeV2`
- Added `DeviceIP` and `DeviceLocation` to:
  - `ContractRecover` callback
  - `Register` callback
  - `ShareContract` callback
  - `DeviceWithinContract` model
  - `AssociationContractDeviceV2` model
- SDK Client Registration methods changed names:
  - `getDeeplinkURL` changed to `generateRegistrationDeeplinkURL`
  - `generateQRCodeURL` changed to `generateRegistrationQRCodeURL`
  - `getQRCode` changed to `getRegistrationQRCode`

Changelog - 7.1.0

- Added `expiresAt` property to create transaction v3 response model

Changelog - 7.0.1

- Added `marketingName` field to `DeviceInfo` model
- Added `E_400090080` and `E_400090081` error codes on `Create Transaction` request

Changelog - 7.0.0

- Renamed `ApplicationClient` to `Client`
- Remove outdated transaction creation request
- Response `Timestamp` is now mandatory
- Added new `ClientBuilder` to create new `Client` and `CallbackHandlerBuilder` for `CallbackHandler`
- Added support for secondary endpoint and secondary endpoint pub (should use the builders for easier client initialization)

Changelog - 6.1.0

- Allow forward slash in the end of the endpoint url
- Fix Generic transaction `action` and `actionName` optional properties
- Added `GetTransactionStatus` API call

- Added support for `default` required authentication on transaction creation

Changelog - 6.0.1

- Detail the transaction deep link creation
- Fix TLS error on frontdoor connection

Changelog - 6.0.0

- Added *Risk Modules objects* for v3 transactions
- Changed `MessageEngine` to `messageEngine` in `NORMAL_TRANSACTION` and `ACTIONABLE_TRANSACTION` v3 transactions;
- Changed `MsgIdentifier` to `msgIdentifier` in `MessageEngine` model.
- Removed `toJsonString` method from all requests, now replaced with `toString` to return JSON string.
- Set `GetHeaders` method to public in `Client` to use in `callback` responses

Changelog - 5.0.2

- Fix http response memory leak

Changelog - 5.0.1

- Set keep alive strategy for the http client

Changelog - 5.0.0

- Removed transaction dry run requests and associated objects
- Replace `Device` object with `DeviceInfo` on `RemoveDeviceContract` callback
- Removed `ContractsKeys` property on `ShareContract` callback
- Removed `paramsOrder` from `GenericTransaction` v2 and added in the `GenericTransaction` v3.
- Use `Metadata` v3 in `Core` object instead of `Metadata` v2.

Changelog - 4.2.0

- Removed admin enums and associated classes

Changelog - 4.1.0

- Add action field to `GenericTransactionsV3`
- Remove setCorrelationID method from the client options

Changelog - 4.0.0

- Envelopes timestamp validation
- Deleted all backoffice requests and associated tests
- Improved http client to be reused
- Added timestamp on association creation
- Added support for request custom headers

Changelog - 3.4.0

- Support for contract_key field on transaction creation
- Added getTransactionDeeplinkURL

Glossary

A list of terms to frame the client in the TrustFactor Services.

Term	Description
`TrustFactor Agent`	TrustFactor Application installed in a mobile device (iOS or Android)
`Correlation ID`	Unique ID that identifies a request in TrustFactor services
`Callbacks`	Requests sent by the TrustFactor services to the Application to notify a state change
`App User ID or User ID`	ID that uniquely identifies the application user on the TrustFactor services
`Username`	The username is the "display name" shown on the TrustFactor Agent and Backoffice when users access their Profile inside an application
`Contract Key`	This key ensures integrity, authenticity, privacy and non-repudiation on contracts communications and identifies a user's contract
`Device Key`	This key ensures integrity, authenticity, privacy and non-repudiation on devices communications and identifies a user's device
`Endpoint Address`	URL to communicate with the TrustFactor services
`Endpoint Public Key`	Used to encrypt messages to the TrustFactor Application Endpoint
`TrustFactor Public Key`	Authenticates communications to your application on the TrustFactor services
`Application Private Key`	This is a secret value that the application should save, and it should not be shared with anyone else but the client

Models

Here you can find some helpful models used in the SDK.

Envelope

The model `Envelope` can be found on `com.securityside.trustfactor.util.crypto` namespace. Contains an encrypted message, the receiver and sender keys and the payload signature

Device Info

The model `DeviceInfo` can be found on `com.securityside.trustfactor.model.device` package.

Attribute	Description
`name`	Device's name (configured by the user on the Agent)
`manufacturer`	Device's manufacturer
`os`	Device's operating system ("Android"/"iOS")
`osVersion`	Device's operating system version

Meta Data

The `MetaData` model for transactions v3 can be found on `com.securityside.trustfactor.model.transaction.creation.v3.auxiliary` package. Contains transaction's metadata.

Attribute	Description
`timestamp`	Should contain the application timestamp when the transaction was created
`userAgent`	Should contain the user's UserAgent of the request that initiated the transaction creation flow
`platform`	User's operating system
`channel`	Application channel used. Useful for applications that have Web/Mobile channels
`sourceIP`	User's IP either IPv4 or IPv6

Core

The `Core` model for transactions v3 can be found on `com.securityside.trustfactor.model.transaction.creation.v3` package.

Attribute	Description
`metadata`	Transaction's metadata

Transaction's core contains basic information needed to create any type of transaction.

Message Engine

The `MessageEngine` model for transactions v3 can be found on `com.securityside.trustfactor.model.transaction.creation.v3.auxiliary` package.

Attribute	Description
`language`	Overrides the preferred language defined in the Operation
`message`	Specifies the message that should be displayed on user's agent. It will override the message template engine or any operation defined message. `msgIdentifier` should be null or empty if this field is filled.

Attribute	Description
`msgIdentifier`	Specifies the operation's defined template message to use

The `MessageEngine` object in the transactions is used to easily show messages to the user. There are two ways to do that, the simplest way is to fill the field `message` with the intended message. The other way is to fill the `msgIdentifier` with the identifier of a message created within the operation referenced on the `action` field of the transaction. This message identifier is retrieved through the backoffice Operations menu. `MessageEngine` can only be used with `NORMAL_TRANSACTION` and `ACTIONABLE_TRANSACTION` types, since they are the only ones that take a predefined operation on the backoffice.

SDK Constants

These constants can be found on `com.securityside.trustfactor.constants.enums` package.

Transaction Type

This constant can be found on `TransactionType` enumerator in `com.securityside.trustfactor.constants.enums` package.

Transaction Type	Description	CustomApps Only
`NORMAL_TRANSACTION`	Normal transactions do not have a decision process. One use case scenario is to replace SMS OTP tokens using TrustFactor's secure communication channel. In the agent, `NORMAL_TRANSACTION` only shows the message, and an 'OK' button and no 'Accept' / 'Reject' buttons.	No
`ACTIONABLE_TRANSACTION`	Transaction with a decision process that is based on an operation.	No
`GENERIC_TRANSACTION`	A very versatile transaction with a decision process that doesn't need an existing operation in the application and that is highly configurable.	No
`SIBS_3DS_TRANSACTION`	Transaction with a decision process for 3DS payments using SIBS API via MBWay SDK. The transaction is created without SIBS details in TrustFactor services and the transaction state will be `TRANSACTION_NON_DECIDABLE` in TrustFactor.	Yes
`SIBS_3DS_TRANSACTION_V2`	Transaction with a decision process for 3DS payments using SIBS API via MBWay SDK. The transaction is created with SIBS details in TrustFactor services and the transaction state can be defined with `Update Transaction status` request after the SIBS callback.	Yes

Transaction Status

This constant can be found on `TransactionStatus` enumerator in `com.securityside.trustfactor.constants.enums` package. **Final State** indicates that the transaction is in a final state

Transaction Type	Final State
`TRANSACTION_PENDING`	No
`TRANSACTION_ACCEPTED`	Yes
`TRANSACTION_DECLINED`	Yes
`TRANSACTION_EXPIRED`	Yes
`TRANSACTION_FAILED`	Yes
`TRANSACTION_NON_DECIDABLE`	Yes
`TRANSACTION_NO_ANSWER`	No

Note: `TRANSACTION_NO_ANSWER` is a `SIBS_3DS_TRANSACTION_V2` only transaction state.

SIBS 3DS v2 Final Status

This constant can be found on `SIBS3DSV2FinalStatus` enumerator in `com.securityside.trustfactor.constants.enums` package.

- `SIBS3DS_TRANSACTION_ACCEPTED`
- `SIBS3DS_TRANSACTION_DECLINED`
- `SIBS3DS_TRANSACTION_EXPIRED`

Authentication Mechanism

This constant can be found on `AuthenticationMechanism` enumerator in `com.securityside.trustfactor.constants.enums` package.

- `AUTHENTICATION_PASSWORD`
- `AUTHENTICATION_BIOMETRICS`
- `AUTHENTICATION_DEFAULT`

Risk Level

This constant can be found on `RiskLevel` enumerator in `com.securityside.trustfactor.constants.enums` package.

- `LOW`
- `MEDIUM`
- `HIGH`

Risk Module

This constant can be found on `RiskModule` enumerator in `com.securityside.trustfactor.constants.enums` package.

- `LRC`
- `IPREP`

Event Data

This constant can be found on `EventData` enumerator in `com.securityside.trustfactor.constants.enums` package.

- `MAP`
- `WEBVIEW`

First Steps

Before you begin

There are two types of TrustFactor Agent applications (mobile apps):

- **TrustFactor App** - maintained by SecuritySide for *Android* and *iOS*
- **CustomApps** - built and customized for customers, this option may have different features and allow for more API calls than the TrustFactor App.

CustomApps have their own dedicated cloud environment and so will have a different *endpointUrl* than the TrustFactor App. With their own cloud environment, they allow better control over the BackOffice and also extended functionality as is the case of embedding the SIBS MBWay SDK in order to perform 3D-Secure credit card payment authorizations through the TrustFactor Agents.

Throughout this document, if a particular API or functionality requires a CustomApp, it will be noted through a "CustomApps only" reference.

Client and Callback Handler

Before starting to integrate the TrustFactor SDK in their application, the developer should have the following necessary data:

- **Endpoint Address** - Ensure there are no network policies restricting an HTTP Connection to/from the endpoint
- **Endpoint Public Key** - Authenticates communications to the TrustFactor services
- **TrustFactor Public Key** - Authenticates communications to your application on the TrustFactor services
- **Application Private Key** - This is a secret value that the application should save, and it should not be shared with anyone else but the client

All of these settings can be set or changed through the Application Settings on the TrustFactor BackOffice. Please check the BackOffice documentation for more information.

To instantiate the `Client` there is a new `ClientBuilder` to help with the secondary endpoint and secondary endpoint pub. This new secondary properties are obligatory, but can be the same as the primary property maintaining the same behavior as the previous `Client`.

```
import com.securityside.trustfactor.client.Client;
import com.securityside.trustfactor.client.ClientBuilder;
import com.securityside.trustfactor.util.exception.CryptoException;

public class Main {
    public static void main(String[] args) {
        ClientBuilder clientBuilder = new ClientBuilder();
        clientBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw="));
        clientBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw="))
        clientBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        clientBuilder.SetAppPriv(new PrivateKey("redacted"));
        clientBuilder.SetDeeplinkBaseURL("https://open.trustfactor.securityside.com");
        Client appClient = clientBuilder.Build();
    }
}
```

The first step in order to use the SDK is to instantiate a `Client` object which acts as a `Client`. This is the central point which you can use in order to make API Calls to TrustFactor Services. This is done by calling `clientBuilder.Build()` method from `ClientBuilder`

In order to handle callbacks you will need to instantiate a `CallbackHandler` object, using the `CallbackHandlerBuilder`, the same way as `ClientBuilder`.

```java
import com.securityside.trustfactor.callback.CallbackHandler;
import com.securityside.trustfactor.callback.CallbackHandlerBuilder;
import com.securityside.trustfactor.util.exception.CryptoException;

public class Main {
    public static void main(String[] args) {
        CallbackHandlerBuilder callbackBuilder = new CallbackHandlerBuilder();
        callbackBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        callbackBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        callbackBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw=")
        callbackBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw=
        callbackBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        callbackBuilder.SetAppPriv(new PrivateKey("redacted"));
        CallbackHandler handler = callbackBuilder.Build();
    }
}
```

After the `Client` and `CallbackHandler` initialization, everything is ready to properly communicate with TrustFactor services.

Send Request

In order to call TrustFactor services we need to instantiate the intended request, and then use the `Client` (created earlier) to call the method for the request. Every request for the TrustFactor services has an associated method in the `Client`.

All the requests' responses extend an abstract class called `GenericResponse`, that have generic fields for all responses such as a dictionary with error codes, if any occurred, the request correlation id, and the HTTP status code.

Handle Callbacks

When the TrustFactor services need to communicate some change, they do so by sending a request to the application, from now on referred to as a callback. There are five different types of callbacks, with detailed explanations in the *Callbacks section* :

Callback	Description	Callback Handle Method
Registration	A user has registered with TrustFactor	`handleRegisterCallback`
Transaction Decision	A user has approved or rejected an authentication with TrustFactor	`handleTransactionCallback`
Device Removal	A user has removed a device associated with TrustFactor	`handleRemoveDeviceContractCallback`
Contract Sharing	A user has shared their profile with another TrustFactor device	`handleShareContractCallback`
Contract Recovery	A user has recovered their profile on a new TrustFactor app through a Recovery Code	`handleContractRecoverCallback`

There is a Production Mode setting on the backoffice when a callback fails. Please consult the Backoffice manual for further information about the Production Mode.

The application developer must expose five different routes, one for each callback type. The resulting URLs must be configured through the TrustFactor BackOffice (see the BackOffice documentation for more information).
After the five routes are exposed in an HTTP Server, to handle the callbacks using the SDK, one needs only to call the respective method from the `CallbackHandler` with the respective input.

When SIBS transactions v2 are active, there is an additional callback for SIBS transactions that functions in a slightly different way.
The application developer still needs to expose an HTTP route specific for the callback, this time calling the `handleGetSIBSRegisterTokenRequest` method from `CallbackHandler`. Furthermore, instead of just returning a 200 HTTP status code the developer must call `handleGetSIBSRegisterTokenResponse` method from `CallbackHandler` with the SIBS activation code. The `handleGetSIBSRegisterTokenResponse` method returns an `Envelope` object with the encrypted data that should be JSON serialized and sent in the body of the response.
An example of this callback can be seen *here*.

Please consult the BackOffice documentation for further information about SIBS transactions v2 activation.

Walkthrough

Register a Contract

To create a contract you need to instantiate a `RegisterCodeReq` object. You will need to insert the App User ID (which identifies the user in TrustFactor services), Username (that is shown to the user and listed in backoffice) and the duration of the code.

The recovery of transaction history depends on the contract's app user ID. That is, if an application wants the user to continue to have access to the transaction history after the registration of a new contract, then it has to reuse the same app user ID that it he had before.

```
import com.securityside.trustfactor.client.Client;
import com.securityside.trustfactor.client.ClientBuilder;
import com.securityside.trustfactor.client.request.registercode.RegisterCodeReq;
import com.securityside.trustfactor.util.exception.CryptoException;

public class Main {
    public static void main(String[] args) {
        ClientBuilder clientBuilder = new ClientBuilder();
        clientBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw="));
        clientBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw=")
        clientBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        clientBuilder.SetAppPriv(new PrivateKey("redacted"));
        clientBuilder.SetDeeplinkBaseURL("https://open.trustfactor.securityside.com");
        Client appClient = clientBuilder.Build();

        // Creates a register code request
        int duration = 60; // code duration of 60 seconds
        RegisterCodeReq request = RegisterCodeReq.builder()
                .appUniqueId("appUniqueId")
                .appUsername("appUsername")
                .codeDuration(duration)
                .build();
    }
}
```

After the request is created we need to send it to the TrustFactor services, and we do it by calling the `registerCode` method in the `Client`.

```
import com.securityside.trustfactor.client.Client;
import com.securityside.trustfactor.client.ClientBuilder;
import com.securityside.trustfactor.client.request.registercode.RegisterCodeReq;
import com.securityside.trustfactor.client.request.registercode.RegisterCodeRes;
import com.securityside.trustfactor.util.exception.CryptoException;

public class Main {
    public static void main(String[] args) {
        ClientBuilder clientBuilder = new ClientBuilder();
        clientBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw="));
        clientBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw=")
        clientBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        clientBuilder.SetAppPriv(new PrivateKey("redacted"));
        clientBuilder.SetDeeplinkBaseURL("https://open.trustfactor.securityside.com");
        Client appClient = clientBuilder.Build();
```

```
        // Creates a register code request
        int duration = 60; // code duration of 60 seconds
        RegisterCodeReq request = RegisterCodeReq.builder()
                .appUniqueId("appUniqueId")
                .appUsername("appUsername")
                .codeDuration(duration)
                .build();

        // Sends a register code request
        RegisterCodeRes response = appClient.registerCode(request);
    }
}
```

When we get the `RegisterCodeRes` response we have two possible flows for the user to complete the registration:

- **DeepLink URL**
- **QR Code**

The `DeepLink URL` is used when the user is in a mobile device and can't scan the QR Code from their own screen. This allows him to tap a button that redirects him to the mobile application and completes the registration. With the `QR Code` there are two possible options. Either with the `QR Code URL` or with the `QR Code`.

- `QR Code URL` - Made for a custom HTTP client or to obtain the token parameter (which is in the URL) and generate the QR Code
- `QR Code` - Returns a Base64 encoded image with a QR Code that the user should scan in order to complete the registration

Note that for the SDK to obtain the Base64 encoded QR Code it needs to call the TrustFactor Services. Consequently, if the application has a tool to generate the QR Codes it can do so, while saving some network traffic.

Depending on the platform (web or mobile) the user is on, you may want to adjust their registration flow accordingly.

```
import com.securityside.trustfactor.client.Client;
import com.securityside.trustfactor.client.ClientBuilder;
import com.securityside.trustfactor.client.request.registercode.RegisterCodeReq;
import com.securityside.trustfactor.client.request.registercode.RegisterCodeRes;
import com.securityside.trustfactor.util.exception.CryptoException;

public class Main {
    public static void main(String[] args) {
        ClientBuilder clientBuilder = new ClientBuilder();
        clientBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw="));
        clientBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw=")
        clientBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        clientBuilder.SetAppPriv(new PrivateKey("redacted"));
        clientBuilder.SetDeeplinkBaseURL("https://open.trustfactor.securityside.com");
        Client appClient = clientBuilder.Build();

        // Creates a register code request
        int duration = 60; // code duration of 60 seconds
        RegisterCodeReq request = RegisterCodeReq.builder()
                .appUniqueId("appUniqueId")
                .appUsername("appUsername")
                .codeDuration(duration)
                .build();

        // Sends a register code request
```

```
        RegisterCodeRes response = appClient.registerCode(request);

        // Handles the response
        int size = 400;  // Size in pixels of the qr code
        String deeplinkUrl = appClient.getDeeplinkURL(response);
        String qrCodeUrl = appClient.generateQRCodeURL(response, size);
        String qrCode = appClient.getQRCode(response, size);
    }
}
```

Finally, when the user finishes the contract registration the TrustFactor services send a callback to notify the application that the user finished the registration. The callback contains information such as the device key, contract key, app user id and *device info*.

```
import com.securityside.trustfactor.callback.CallbackHandler;
import com.securityside.trustfactor.callback.CallbackHandlerBuilder;
import com.securityside.trustfactor.callback.message.Register;
import com.securityside.trustfactor.util.exception.CryptoException;

public class Main {
    public static void main(String[] args) {
        CallbackHandlerBuilder callbackBuilder = new CallbackHandlerBuilder();
        callbackBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        callbackBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        callbackBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw=")
        callbackBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw=
        callbackBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        callbackBuilder.SetAppPriv(new PrivateKey("redacted"));
        CallbackHandler handler = callbackBuilder.Build();

        String data = ""; // received in the request body, HTTP server logic not shown here
        Register register = handler.handleRegisterCallback(data);
    }
}
```

Create a Transaction

To create a transaction you first need to instantiate the wanted object such as `GenericTransaction` for a generic transaction. Keep in mind that there may be several versions of the `GenericTransaction` class, therefore always use the most recent one to get the latest features.

Old method versions are kept for compatibility reasons and may be deprecated in future SDK versions.

```
import com.securityside.trustfactor.client.Client;
import com.securityside.trustfactor.client.ClientBuilder;
import com.securityside.trustfactor.model.transaction.creation.v3.Core;
import com.securityside.trustfactor.model.transaction.creation.v3.GenericTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.Metadata;
import com.securityside.trustfactor.util.exception.CryptoException;

import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        ClientBuilder clientBuilder = new ClientBuilder();
        clientBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw="));
        clientBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw="))
        clientBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        clientBuilder.SetAppPriv(new PrivateKey("redacted"));
        clientBuilder.SetDeeplinkBaseURL("https://open.trustfactor.securityside.com");
        Client appClient = clientBuilder.Build();

        // Creates a generic transaction
        GenericTransaction transaction = GenericTransaction.builder()
                .message("Test")
                .actionName("Action")
                .transactionDuration(30)
                .core(Core.builder()
                        .metadata(Metadata.builder()
                                .timestamp(1619705292L)
                                .userAgent("TrustFactor Java SDK")
                                .channel("Mobile")
                                .sourceIp("1.2.3.4")
                                .build())
                        .build())
                .customHeaders(new HashMap<String, String>() {{
                    put("X-Custom-Header", "Header value");
                }})
                .build();
    }
}
```

After creating the request object you will need to call an `Client` method, in this case you will need to use either `createTransactionV3WithUserID` or `createTransactionV3WithKey` to create the transaction, respectively with the User ID or the Contract Key. For this example we opted for the `createTransactionV3WithUserID`.

```
import com.securityside.trustfactor.client.Client;
import com.securityside.trustfactor.client.ClientBuilder;
import com.securityside.trustfactor.client.request.transaction.CreateTransactionV3WithUserIDRes;
import com.securityside.trustfactor.model.transaction.creation.v3.Core;
import com.securityside.trustfactor.model.transaction.creation.v3.GenericTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.Metadata;
```

```java
import com.securityside.trustfactor.util.exception.CryptoException;

import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        ClientBuilder clientBuilder = new ClientBuilder();
        clientBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw="));
        clientBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw="));
        clientBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        clientBuilder.SetAppPriv(new PrivateKey("redacted"));
        clientBuilder.SetDeeplinkBaseURL("https://open.trustfactor.securityside.com");
        Client appClient = clientBuilder.Build();

        // Creates a generic transaction
        GenericTransaction transaction = GenericTransaction.builder()
                .message("Test")
                .actionName("Action")
                .transactionDuration(30)
                .core(Core.builder()
                        .metadata(Metadata.builder()
                                .timestamp(1619705292L)
                                .userAgent("TrustFactor Java SDK")
                                .channel("Mobile")
                                .sourceIp("1.2.3.4")
                                .build())
                        .build())
                .customHeaders(new HashMap<String, String>() {{
                    put("X-Custom-Header", "Header value");
                }})
                .build();

        // Send the create generic transaction
        CreateTransactionV3WithUserIDRes response = appClient
                .createTransactionV3WithUserID("john.doe", transaction);
    }
}
```

When the generic transaction request is completed a `CreateTransactionV3WithUserIDRes` response is created, inherited from a generic response class `GenericResponse`. The response contains the transaction Unique ID, the Contract Key as well as the fields from the `GenericResponse` presented earlier.

Apart from deciding a transaction using a push notification, a transaction deep link url can also be used on mobile. The `getTransactionDeeplinkURL` can be used when the user is in a mobile device and does not receive a notification for the transaction. This allows him to tap a button that redirects him to the mobile application and decide the transaction.

```java
import com.securityside.trustfactor.client.Client;
import com.securityside.trustfactor.client.ClientBuilder;
import com.securityside.trustfactor.client.request.transaction.CreateTransactionV3WithUserIDRes;
import com.securityside.trustfactor.model.transaction.creation.v3.Core;
import com.securityside.trustfactor.model.transaction.creation.v3.GenericTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.Metadata;
import com.securityside.trustfactor.util.exception.CryptoException;

import java.util.HashMap;
```

```java
public class Main {
    public static void main(String[] args) {
        ClientBuilder clientBuilder = new ClientBuilder();
        clientBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        clientBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw="));
        clientBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw="
        clientBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        clientBuilder.SetAppPriv(new PrivateKey("redacted"));
        clientBuilder.SetDeeplinkBaseURL("https://open.trustfactor.securityside.com");
        Client appClient = clientBuilder.Build();

        // Creates a generic transaction
        GenericTransaction transaction = GenericTransaction.builder()
                .message("Test")
                .actionName("Action")
                .transactionDuration(30)
                .core(Core.builder()
                        .metadata(Metadata.builder()
                                .timestamp(1619705292L)
                                .userAgent("TrustFactor Java SDK")
                                .channel("Mobile")
                                .sourceIp("1.2.3.4")
                                .build())
                        .build())
                .customHeaders(new HashMap<String, String>() {{
                    put("X-Custom-Header", "Header value");
                }})
                .build();

        // Send the create generic transaction
        CreateTransactionV3WithUserIDRes response = appClient
                .createTransactionV3WithUserID("john.doe", transaction);

        String transactionURL = appClient.getTransactionDeeplinkURL(response);
    }
}
```

When the user decides the transaction or when it expires, TrustFactor services will send a callback to the application notifying its final status.

```java
import com.securityside.trustfactor.callback.CallbackHandler;
import com.securityside.trustfactor.callback.CallbackHandlerBuilder;
import com.securityside.trustfactor.callback.message.Transaction;
import com.securityside.trustfactor.util.exception.CryptoException;

public class Main {
    public static void main(String[] args) {
        CallbackHandlerBuilder callbackBuilder = new CallbackHandlerBuilder();
        callbackBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        callbackBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        callbackBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw="
        callbackBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cLOCDAuM8KYXTj3hoaYHw=
        callbackBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        callbackBuilder.SetAppPriv(new PrivateKey("redacted"));
        CallbackHandler handler = callbackBuilder.Build();
```

```
        String data = ""; // received in the request body, HTTP server logic not shown here
        Transaction transaction = handler.handleTransactionCallback(data);
    }
}
```

Handle SIBS callback

When the application receives this callback it means that SIBS transactions v2 are active, and the user needs an activation code to communicate with the MBWay SDK.

```
import com.securityside.trustfactor.callback.CallbackHandler;
import com.securityside.trustfactor.callback.CallbackHandlerBuilder;
import com.securityside.trustfactor.model.sibs.GetSIBSRegisterTokenReq;
import com.securityside.trustfactor.util.crypto.Envelope;
import com.securityside.trustfactor.util.exception.CryptoException;

public class Main {
    public static void main(String[] args) {
        CallbackHandlerBuilder callbackBuilder = new CallbackHandlerBuilder();
        callbackBuilder.SetPrimaryEndpoint("https://applications.trustfactor.app");
        callbackBuilder.SetSecondaryEndpoint("https://applications.trustfactor.app");
        callbackBuilder.SetPrimaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw=")
        callbackBuilder.SetSecondaryEndpointPub(new PublicKey("WMIskJyIJo36PO93Qju351cL0CDAuM8KYXTj3hoaYHw=
        callbackBuilder.SetTFPub(new PublicKey("craE8nZ9eCjUOFwArwTqfdNuruUrDtCNIQhYBt4vyYQ="));
        callbackBuilder.SetAppPriv(new PrivateKey("redacted"));
        CallbackHandler handler = callbackBuilder.Build();

        String data = ""; // received in the request body, HTTP server logic not shown here
        GetSIBSRegisterTokenReq regCodeRequest = handler.handleGetSIBSRegisterTokenRequest(data);

        // Requests SIBS activation code

        String seeAtvCodSdk = "see_atv_cod_sdk"; // registration token for SIBS
        Envelope envelope = handler.handleGetSIBSRegisterTokenResponse(seeAtvCodSdk);
    }
}
```

API Description

API Calls

The response may contain `ErrorCodes` when the TrustFactor service call is not successful.

`ErrorCodes` are errors with unique codes standardized by TrustFactor services to be easier to manage errors outside TrustFactor services and give more information about why the TrustFactor services couldn't handle the request.

`ErrorCodes` are thrown when a server side validation blocks a request.

`ErrorCodes` contain a unique code (e.g. `E_400020001`), a message and optionally, depending on the error code, extra arguments to the error. For example, when a contract already exists an `E_400020001` error code is thrown and has an argument with the App User ID, that serves as additional information to situate the error.

Also, to notice that every request that supports User ID as a similar request that supports Contract Key. That way the application has a more diverse choice.

Create Register Code

Creates a registration token for a user with an expiration time between 0 seconds and 600 seconds, otherwise, the request will return an error.
You should create and display a QR Code and/or a Deep Link URL considering user's device using the response object.
To get the QR Code you need to use the `getQRCode` method from the Application Client, for the QR Code URL you will need to use the `generateQRCodeURL` method and finally for the Deep Link URL you need to use `getDeeplinkURL` method.

- `RegisterCodeReq` (Request Object)
- `RegisterCodeRes` (Response Object)

Possible Response Error Codes:

- `E_400020001`
- `E_400020005`
- `E_400020006`
- `E_400020007`

Reset Contract Device

Removes the specified device from this contract.
If it is the last associated device, the user will no longer have TrustFactor associated to the account. From that moment on, it will no longer be possible to create new transactions for that user until he associates a new device through a registration flow.

- With Key
  - `ResetContractDeviceReq` (Request Object)
  - `ResetContractDeviceRes` (Response Object)
- With User ID
  - `ResetContractDeviceByUserIDReq` (Request Object)
  - `ResetContractDeviceByUserIDRes` (Response Object)

Possible Response Error Codes:

- `E_500010002`
- `E_500030003`
- `E_400030010`
- `E_400020002`
- `E_400040001`
- `E_400020001`

Get Contract Authentication Mechanisms

Returns all the authentications the user has configured on his agents. Might be useful to block certain actions (and don't create transactions), if the user doesn't have the required authentication mechanisms associated.

- With Key
  - `GetContractAuthenticationMechanismsReq` (Request Object)
  - `GetContractAuthenticationMechanismsRes` (Response Object)
- With User ID
  - `GetContractAuthenticationMechanismsByUserIDReq` (Request Object)
  - `GetContractAuthenticationMechanismsByUserIDRes` (Response Object)

Possible Response Error Codes:

- `E_400020002`

Set SIBS Details

Sets the contract SIBS details used for SIBS transactions.

- With Key
  - `SetSIBSDetailsByKeyReq` (Request Object)
  - `SetSIBSDetailsByKeyRes` (Response Object)
- With User ID
  - `SetSIBSDetailsByUserIDReq` (Request Object)
  - `SetSIBSDetailsByUserIDRes` (Response Object)

Possible Response Error Codes:

- `E_400090010`
- `E_400020002`
- `E_500010002`

Create Transaction

This API creates a new transaction for a user. In order to call this method you need a registered user, and a transaction object. For more information about how to create transactions, please check *Transactions in depth* chapter. The transaction object should have one of the following types:

Transaction Type	CustomApps Only
`NORMAL_TRANSACTION`	No
`ACTIONABLE_TRANSACTION`	No
`GENERIC_TRANSACTION`	No
`SIBS_3DS_TRANSACTION`	Yes
`SIBS_3DS_TRANSACTION_V2`	Yes

`NORMAL_TRANSACTION` and `ACTIONABLE_TRANSACTION` types reference an operation (see the BackOffice Documentation for more information on Operations) that should already be defined in Backoffice via `action` field.

- With Key

  - `CreateTransactionV3WithKeyReq` (Request Object)
  - `CreateTransactionV3WithKeyRes` (Response Object)
- With ID
  - `CreateTransactionV3WithUserIDReq` (Request Object)
  - `CreateTransactionV3WithUserIDRes` (Response Object)

Possible Response Error Codes:

- `E_500010002`
- `E_400020002`
- `E_400090057`
- `E_500040001`
- `E_500050003`
- `E_400090056`
- `E_400090050`
- `E_500050001`
- `E_400090007`
- `E_400090051`
- `E_400090052`
- `E_400090016`
- `E_400090017`
- `E_400090018`
- `E_400090044`
- `E_400090045`
- `E_400090026`
- `E_400090027`
- `E_400090028`
- `E_400090029`
- `E_400090015`
- `E_400090077`
- `E_400090039`
- `E_400090040`
- `E_400090041`
- `E_400090042`
- `E_400090072`
- `E_400090073`
- `E_400090074`
- `E_400090014`
- `E_400090038`
- `E_400090013`
- `E_400090071`
- `E_400090061`
- `E_400090062`
- `E_400090063`
- `E_400090064`
- `E_400090065`
- `E_400090066`
- `E_400090067`
- `E_400090068`
- `E_400090069`
- `E_400090070`
- `E_400090076`
- `E_400090019`
- `E_400090020`
- `E_400090023`
- `E_400090022`
- `E_400090024`

- E_400090046
- E_400090031
- E_400090032
- E_400090033
- E_400090035
- E_400090034
- E_400090036
- E_400090037
- E_400090047
- E_400090030
- E_400090079
- E_400090080
- E_400090081

Update Transaction Status – CustomApps with SIBS MBWay SDK only

Sets the status of a SIBS3DSV2 transaction.

- `UpdateTransactionStatusReq` (Request Object)
- `UpdateTransactionStatusRes` (Response Object)

The possible states the transaction can be updated to are:

- `TRANSACTION_ACCEPTED`
- `TRANSACTION_DECLINED`
- `TRANSACTION_EXPIRED`

If the requests tries to update the transaction to the same state the response returns a 204 HTTP code.

The possible states the transaction can be updated from are:

- `TRANSACTION_PENDING`
- `TRANSACTION_NO_ANSWER`

These transaction states can be found on `com.securityside.trustfactor.constants.enums` package.

Possible Response Error Codes:

- E_400080011
- E_400010004
- E_400080010
- E_400080012
- E_400090053
- E_500010002

Get Transaction Status

Retrieves the status of a transaction.

- `GetTransactionStatusReq` (Request Object)
- `GetTransactionStatusRes` (Response Object)

Possible Response Error Codes:

- E_400090053
- E_400090054

Callbacks

Every callback has `Envelope` class as the request body.

All callbacks come with the following headers:

- `X-Request-Correlation-ID` - identifies the Request ID on TrustFactor services (useful for error tracing – you should use this value when asking for support from the SecuritySide Team)
- `X-Application-Key` - contains the TrustFactor Public Key and might be used to block unauthorized access to endpoints

Register

This callback is triggered when a user successfully registers a new contract.
Contains an `X-Registration-ID` header with the App User ID that might be useful for quick implementations.
Contains the following data:

- Device Key
- Contract Key
- *Device Info*
- App User ID

Share Contract

This callback is triggered when a user shares a contract from one agent to another.
Contains the following data:

- Receiver Device Key
- Sender Device Key
- Shared contract key
- Shared app user ID
- *Receiver Device Info*

Contract Recover

This callback is triggered when a user recovers a contract via TrustFactor identity backup.
Contains the following data:

- Device Key
- *Device Info*
- Contract Key
- App User ID

Remove Device Contract

This callback is triggered when an association is removed (either via client application/backoffice or agent side).
Contains the following data:

- Removed Device Key
- Device Key that removed the other (optional, only happens when an agent is removed using another agent)
- Contract Key
- App User ID
- *Removed Device Info*

Transaction

Contains an `X-Transaction-ID` header with the Transaction ID, might be useful for middlewares.
This callback is triggered when a transaction is decided. Below is a list of possible transaction states:

- `TRANSACTION_ACCEPTED`
- `TRANSACTION_DECLINED`
- `TRANSACTION_EXPIRED`

Contains the following data:

- *Status* (final transaction status)
- Device Key (optional, the field is filled when transaction is accepted or declined)
- Transaction ID
- App User ID
- Contract Key
- *Authentications* - Authentications used for the agent transaction decision. Should be validated in order to confirm all the required authentications of the transaction were used on this decision in case it was accepted (a malicious application emulating the agent might decide the transaction ignoring the required authentications)

Get SIBS Register Token – CustomApps with SIBS MBWay SDK only

This callback is triggered when the agent requires SIBS registration token in order to communicate with MBWay SDK. The callback should respond with an activation code for the user.

Transactions in depth

In this chapter we will go in depth about how to create transactions, starting from which types you can create, which transactions types requires an operation and which parameters the transaction requires or may have.For further information about risk modules and how to create them, please visit *Risk Modules In Depth* chapter.

For some transaction types, specifically `NORMAL_TRANSACTION` and `ACTIONABLE_TRANSACTION`, an operation created in backoffice is needed. So for the sake of the examples assume there is a `transfer` operation created on the backoffice, identified as Attachment #1 on *Attachments* chapter.

Normal Transaction

Attribute	Description	Optional
`action`	Overrides `Operation` defined preferred	No
`core`	Please check `Core` in the *Models* chapter for further information	No
`messageEngine`	Please check `Message Engine` in the *Models* chapter for further information	No

```java
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.Metadata;
import com.securityside.trustfactor.model.transaction.creation.v3.Core;
import com.securityside.trustfactor.model.transaction.creation.v3.NormalTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.MessageEngine;

public class Main {
    public static void main(String[] args) {
        NormalTransaction normalTransaction = NormalTransaction.builder()
                .action("transfer")
                .core(Core.builder()
                        .metadata(Metadata.builder()
                                .timestamp(1619705292L)
                                .userAgent("TrustFactor Java SDK")
                                .channel("Mobile")
                                .sourceIp("1.2.3.4")
                                .build())
                        .build())
                .messageEngine(MessageEngine.builder()
                        .message("This is a message to show to the user")
                        .build())
                .build();
    }
}
```

Actionable Transaction

Attribute	Description	Optional
`action`	The action of an operation defined in the backoffice	No
`messageEngine`	Please check `Message Engine` in the *Models* chapter for further information	No
`core`	Please check `Core` in the *Models* chapter for further information	No

Attribute	Description	Optional
`customHeaders`	Custom headers are sent along with transaction decision callback. One possible use case scenario would be setting an ID, so the application could correlate the transaction decision request received from TrustFactor services with the one that led to the transaction creation.	Yes
`requiredAuthentications`	Authentications required to approve the transaction. The possible values come from `Authentication Mechanism` enum.	Yes
`overrideRiskCalculation`	You can optionally define the risk of the transaction and / or the parameters as well as the message that is displayed in the risk of the parameters if you activate this flag.	Yes
`risk`	Sets the transaction risk when the flag `OverrideRiskCalculation` is active. It is possible, but not obligatory when you fill `Risk` field, to fill `Risk` and `RiskMessage` fields from Params. The values from `Risk` and `Risk` and `RiskMessage` fields from Params should be null when the flag `OverrideRiskCalculation` is not active. The values that are possible for `Risk` field of the transaction are in the `Risk Level` enum.	Yes
`params`	The `Params` are used to send information in the transaction, by setting the values of the operation parameters, obeying the operation template. Each key of the `Params` dictionary corresponds to a key of the params of the operation defined in the `Key` field. Every parameter of the operation should be defined in the `Params` field of the transaction if the flag `Optional` is turned off, otherwise the parameter doesn't need to be filled. That way we can guarantee that a transaction has the mandatory parameters defined in the operation filled.	Yes
`riskModules`	Additional parameters for the risk modules calculation. For example the origin coordinates of the transaction to be used by `lrc` risk module. The risk modules calculated are defined in the operation. In the example of `transfer` operation, both `ip_reputation` and `lrc` are to be calculated, but they are not mandatory, meaning they can fail, but the transaction will proceed anyway, ignoring the risk score that would have been returned from the risk module.	Yes

```
import com.securityside.trustfactor.constants.enums.AuthenticationMechanism;
import com.securityside.trustfactor.constants.enums.RiskLevel;
import com.securityside.trustfactor.constants.enums.RiskModule;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.MessageEngine;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.Metadata;
import com.securityside.trustfactor.model.transaction.creation.v3.ActionableTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.Core;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.Param;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.IRiskModuleInput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.RiskModuleInputLRC;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.TransactionAgentInput;

import java.util.Arrays;
import java.util.HashMap;

public class Main {
```

```java
    public static void main(String[] args) {
        ActionableTransaction transaction = ActionableTransaction.builder()
                .action("transfer")
                .messageEngine(MessageEngine.builder()
                        .message("This is a message to show to the user")
                        .build())
                .core(Core.builder()
                        .metadata(Metadata.builder()
                                .timestamp(1619705292L)
                                .userAgent("TrustFactor Java SDK")
                                .channel("Mobile")
                                .sourceIp("1.2.3.4")
                                .build())
                        .build())
                .customHeaders(new HashMap<String, String>()
                {{
                    put("X-Custom-Header", "Header value");
                }})
                .requiredAuthentications(Arrays.asList(
                        AuthenticationMechanism.AUTHENTICATION_BIOMETRICS,
                        AuthenticationMechanism.AUTHENTICATION_PASSWORD)
                )
                .overrideRiskCalculation(true)
                .risk(RiskLevel.LOW)
                .params(new HashMap<String, Param>()
                {{
                    put("source_account", Param.builder()
                            .value("leandro.pinto")
                            .build());
                    put("destination_account", Param.builder()
                            .value("hilario.coelho")
                            .build());
                    put("amount", Param.builder()
                            .value(100)
                            .aux(new HashMap<String, String>(){{
                                put("currency", "EUR");
                            }})
                            .build());
                }})
                .riskModules(TransactionAgentInput.builder()
                        .input(new HashMap<RiskModule, IRiskModuleInput>() {{
                            put(RiskModule.LRC, RiskModuleInputLRC.builder()
                                    .lat(1)
                                    .lon(2)
                                    .accuracy(20)
                                    .build());
                        }})
                        .build())
                .build();
    }
}
```

Generic Transaction

Attribute	Description	Optional
`message`	Transaction message to be shown to the user	No
`action`	This field is only used for filtering in the backoffice, not to be compared with `Action` field from `NormalTransaction` or `ActionableTransaction`. Ideally this field should be versioned, so when a change in the schema is made the version is bumped to keep track of the different parameters.	No
`transactionDuration`	The duration of the transaction limited between 30 seconds and 150 seconds.	No
`core`	Please check `Core` in the *Models* chapter for further information	No
`actionName`	The action name for this transaction that the user will see in his Agent.	Yes
`customHeaders`	Custom headers are sent along with transaction decision callback. One possible use case scenario would be setting an ID, so the application could correlate the transaction decision request received from TrustFactor services with the one that led to the transaction creation.	Yes
`requiredAuthentications`	Authentications required to approve the transaction. The possible values come from `Authentication Mechanism` enum.	Yes
`risk`	Sets the transaction risk. The values that are possible are in the `Risk Level` enum. When filled the transaction will assume the defined risk, and the params risk should be filled as well if applicable.	Yes
`params`	The `Params` are used to send information in the transaction, but does not obey an operation template. Meaning that the transaction can send whatever they want.	Yes
`riskModules`	Additional parameters for the risk modules calculation. For example the origin coordinates of the transaction to be used by `lrc` risk module.	Yes
`riskModulesAdded`	Additional risk modules that are shown to the user. New risk modules can be added to the output shown to the user	Yes
`riskModulesToCalculate`	Risk modules to calculate and specify if they are mandatory, because there is not an operation to decide which risk modules to calculate	Yes

```
import com.securityside.trustfactor.constants.enums.AuthenticationMechanism;
import com.securityside.trustfactor.constants.enums.RiskLevel;
import com.securityside.trustfactor.constants.enums.RiskModule;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.Metadata;
import com.securityside.trustfactor.model.transaction.creation.v3.Core;
import com.securityside.trustfactor.model.transaction.creation.v3.GenericTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .Param;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.TransactionRiskModuleToUse;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.TransactionRiskModulesToUse;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.IRiskModuleInput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.RiskModuleInputLRC;
```

```java
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.TransactionAgentInput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.TransactionAgentOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.TransactionRiskModuleOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.ModuleOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Coordinates;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Map;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Pinpoint;

import java.util.Arrays;
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        GenericTransaction transaction = GenericTransaction.builder()
                .message("Test")
                .actionName("Action")
                .action("test action")
                .transactionDuration(30)
                .core(Core.builder()
                        .metadata(Metadata.builder()
                                .timestamp(1619705292L)
                                .userAgent("TrustFactor Java SDK")
                                .channel("Mobile")
                                .sourceIp("1.2.3.4")
                                .build())
                        .build())
                .customHeaders(new HashMap<String, String>() {{
                    put("X-Custom-Header", "Header value");
                }})
                .requiredAuthentications(Arrays.asList(
                        AuthenticationMechanism.AUTHENTICATION_BIOMETRICS,
                        AuthenticationMechanism.AUTHENTICATION_PASSWORD)
                )
                .risk(RiskLevel.LOW)
                .params(new HashMap<String, Param>() {{
                    put("source_account", Param.builder()
                            .value("leandro.pinto")
                            .build());
                    put("destination_account", Param.builder()
                            .value("hilario.coelho")
                            .build());
                    put("amount", Param.builder()
                            .value(100)
                            .aux(new HashMap<String, String>() {{
                                put("currency", "EUR");
                            }})
                            .build());
                }})
                .paramsOrder(new HashMap<String, Long>() {{
```

```
                    put("source_account", 0L);
                    put("destination_account", 1L);
                    put("amount", 2L);
                }})
                .riskModules(TransactionAgentInput.builder()
                        .input(new HashMap<RiskModule, IRiskModuleInput>() {{
                            put(RiskModule.LRC, RiskModuleInputLRC.builder()
                                    .lat(1)
                                    .lon(2)
                                    .accuracy(20)
                                    .build());
                        }})
                        .build())
                .riskModulesAdded(TransactionAgentOutput.builder()
                        .output(new HashMap<String, TransactionRiskModuleOutput>() {{
                            put("new_risk_module", TransactionRiskModuleOutput.builder()
                                    .agentModuleOutput(ModuleOutput.builder()
                                            .risk(RiskLevel.HIGH)
                                            .message("Some message to notify the risk level")
                                            .onTap(Map.builder()
                                                    .title("Origin Location")
                                                    .coordinates(Coordinates.builder()
                                                            .lat(41.25F)
                                                            .lon(-8.5437F)
                                                            .accuracy(50F)
                                                            .build())
                                                    .pinpoint(Pinpoint.builder()
                                                            .country("Portugal")
                                                            .city("Porto")
                                                            .build())
                                                    .build().CreateMapOutput())
                                            .build())
                                    .build());
                        }})
                        .build())
                .riskModulesToCalculate(TransactionRiskModulesToUse.builder()
                        .transactionRiskModulesToUse(
                                new HashMap<RiskModule, TransactionRiskModuleToUse>() {{
                                    put(RiskModule.LRC, TransactionRiskModuleToUse.builder()
                                            .mandatory(true)
                                            .build());
                                    put(RiskModule.IPREP, TransactionRiskModuleToUse.builder()
                                            .mandatory(true)
                                            .build());
                                }}
                        )
                        .build())
                .build();
    }
}
```

SIBS 3DS Transaction

Attribute	Description	Optional
`message`	Transaction message to be shown to the user	No

Attribute	Description	Optional
`actionName`	The action name for this transaction that the user will see in his Agent.	No
`transactionDuration`	The duration of the transaction limited between 30 seconds and 150 seconds.	No
`core`	Please check `Core` in the *Models* chapter for further information	No
`typeData`	Information needed for the SIBS transaction. In this version it is not possible to input the SIBS details in the transaction, so the agent retrieves that information from SIBS. The drawback is that it is only possible to get the details one time with SIBS that will be with the first agent that registered in the MBWay SDK and consult them. If for instance you have two agents on your contract you cannot see the details on the second one (the one who didn't perform the transaction)	No
`requiredAuthentications`	Authentications required to approve the transaction. The possible values come from `Authentication Mechanism` enum.	Yes
`risk`	Sets the transaction risk. The values that are possible are in the `Risk Level` enum. When filled the transaction will assume the defined risk, and the params risk should be filled as well if applicable.	Yes
`riskModules`	Additional parameters for the risk modules calculation. For example the origin coordinates of the transaction to be used by `lrc` risk module.	Yes
`riskModulesAdded`	Additional risk modules that are shown to the user. New risk modules can be added to the output shown to the user	Yes
`riskModulesToCalculate`	Risk modules to calculate and specify if they are mandatory, because there is not an operation to decide which risk modules to calculate	Yes

```
import com.securityside.trustfactor.constants.enums.AuthenticationMechanism;
import com.securityside.trustfactor.constants.enums.RiskLevel;
import com.securityside.trustfactor.constants.enums.RiskModule;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.*;
import com.securityside.trustfactor.model.transaction.creation.v3.Core;
import com.securityside.trustfactor.model.transaction.creation.v3.SIBS3DSTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.TransactionRiskModuleToUse;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.TransactionRiskModulesToUse;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.IRiskModuleInput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.RiskModuleInputLRC;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.TransactionAgentInput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.TransactionAgentOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.TransactionRiskModuleOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.ModuleOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
```

```
        .riskmodules.output.agent.eventdata.Coordinates;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Map;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Pinpoint;

import java.util.Arrays;
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        SIBS3DSTransaction transaction = SIBS3DSTransaction.builder()
                .message("Test")
                .actionName("Action")
                .transactionDuration(30)
                .core(Core.builder()
                        .metadata(Metadata.builder()
                                .timestamp(1619705292L)
                                .userAgent("TrustFactor Java SDK")
                                .channel("Mobile")
                                .sourceIp("1.2.3.4")
                                .build())
                        .build())
                .requiredAuthentications(Arrays.asList(
                        AuthenticationMechanism.AUTHENTICATION_BIOMETRICS,
                        AuthenticationMechanism.AUTHENTICATION_PASSWORD)
                )
                .risk(RiskLevel.LOW)
                .riskModules(TransactionAgentInput.builder()
                        .input(new HashMap<RiskModule, IRiskModuleInput>() {{
                            put(RiskModule.LRC, RiskModuleInputLRC.builder()
                                    .lat(1)
                                    .lon(2)
                                    .accuracy(20)
                                    .build());
                        }})
                        .build())
                .riskModulesAdded(TransactionAgentOutput.builder()
                        .output(new HashMap<String, TransactionRiskModuleOutput>() {{
                            put("new_risk_module", TransactionRiskModuleOutput.builder()
                                    .agentModuleOutput(ModuleOutput.builder()
                                            .risk(RiskLevel.HIGH)
                                            .message("Some message to notify the risk level")
                                            .onTap(Map.builder()
                                                    .title("Origin Location")
                                                    .coordinates(Coordinates.builder()
                                                            .lat(41.25F)
                                                            .lon(-8.5437F)
                                                            .accuracy(50F)
                                                            .build())
                                                    .pinpoint(Pinpoint.builder()
                                                            .country("Portugal")
                                                            .city("Porto")
                                                            .build())
                                                    .build().CreateMapOutput())
                                            .build())
```

```
                                .build());
                        }})
                        .build())
                .riskModulesToCalculate(TransactionRiskModulesToUse.builder()
                        .transactionRiskModulesToUse(
                                new HashMap<RiskModule, TransactionRiskModuleToUse>() {{
                                    put(RiskModule.LRC, TransactionRiskModuleToUse.builder()
                                            .mandatory(true)
                                            .build());
                                    put(RiskModule.IPREP, TransactionRiskModuleToUse.builder()
                                            .mandatory(true)
                                            .build());
                                }}
                        )
                        .build())
                .typeData(SIBS3DSTransactionTypeData.builder()
                        .amount(100)
                        .currency("EUR")
                        .mbWaySDKRegistrationData(MBWaySDKRegistrationData.builder().build())
                        .serviceOperationPlayerCode("")
                        .build())
                .build();
    }
}
```

SIBS 3DS V2 Transaction

Attribute	Description	Optional
`message`	Transaction message to be shown to the user	No
`actionName`	The action name for this transaction that the user will see in his Agent.	No
`transactionDuration`	The duration of the transaction limited between 30 seconds and 150 seconds.	No
`core`	Please check `Core` in the *Models* chapter for further information	No
`typeData`	Information needed for the SIBS transaction. In this version of the SIBS transaction you need to input the SIBS details to the transaction. As the SIBS details stay with the TrustFactor transaction you can consult those details anytime with any agent you have registered in your contract, since it is not dependent of SIBS anymore.	No
`requiredAuthentications`	Authentications required to approve the transaction. The possible values come from `Authentication Mechanism` enum.	Yes
`Risk`	Sets the transaction risk. The values that are possible are in the `Risk Level` enum. When filled the transaction will assume the defined risk, and the params risk should be filled as well if applicable.	Yes
`riskModules`	Additional parameters for the risk modules calculation. For example the origin coordinates of the transaction to be used by `lrc` risk module.	Yes
`riskModulesAdded`	Additional risk modules that are shown to the user. New risk modules can be added to the output shown to the user	Yes

Attribute	Description	Optional
`riskModulesToCalculate`	Risk modules to calculate and specify if they are mandatory, because there is not an operation to decide which risk modules to calculate	Yes

```java
import com.securityside.trustfactor.constants.enums.AuthenticationMechanism;
import com.securityside.trustfactor.constants.enums.RiskLevel;
import com.securityside.trustfactor.constants.enums.RiskModule;
import com.securityside.trustfactor.model.transaction.creation.v3.SIBS3DSTransactionV2;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary.*;
import com.securityside.trustfactor.model.transaction.creation.v3.Core;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.TransactionRiskModuleToUse;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.TransactionRiskModulesToUse;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.IRiskModuleInput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.RiskModuleInputLRC;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.TransactionAgentInput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.TransactionAgentOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.TransactionRiskModuleOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.ModuleOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Coordinates;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Map;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Pinpoint;

import java.util.Arrays;
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        SIBS3DSTransactionV2 transaction = SIBS3DSTransactionV2.builder()
                .message("Test")
                .actionName("Action")
                .transactionDuration(30)
                .core(Core.builder()
                        .metadata(Metadata.builder()
                                .timestamp(1619705292L)
                                .userAgent("TrustFactor Java SDK")
                                .channel("Mobile")
                                .sourceIp("1.2.3.4")
                                .build())
                        .build())
                .requiredAuthentications(Arrays.asList(
                        AuthenticationMechanism.AUTHENTICATION_BIOMETRICS,
                        AuthenticationMechanism.AUTHENTICATION_PASSWORD)
                )
                .risk(RiskLevel.LOW)
```

```
                .riskModules(TransactionAgentInput.builder()
                        .input(new HashMap<RiskModule, IRiskModuleInput>() {{
                            put(RiskModule.LRC, RiskModuleInputLRC.builder()
                                    .lat(1)
                                    .lon(2)
                                    .accuracy(20)
                                    .build());
                        }})
                        .build())
                .riskModulesAdded(TransactionAgentOutput.builder()
                        .output(new HashMap<String, TransactionRiskModuleOutput>() {{
                            put("new_risk_module", TransactionRiskModuleOutput.builder()
                                    .agentModuleOutput(ModuleOutput.builder()
                                            .risk(RiskLevel.HIGH)
                                            .message("Some message to notify the risk level")
                                            .onTap(Map.builder()
                                                    .title("Origin Location")
                                                    .coordinates(Coordinates.builder()
                                                            .lat(41.25F)
                                                            .lon(-8.5437F)
                                                            .accuracy(50F)
                                                            .build())
                                                    .pinpoint(Pinpoint.builder()
                                                            .country("Portugal")
                                                            .city("Porto")
                                                            .build())
                                                    .build().CreateMapOutput())
                                            .build())
                                    .build());
                        }})
                        .build())
                .riskModulesToCalculate(TransactionRiskModulesToUse.builder()
                        .transactionRiskModulesToUse(
                                new HashMap<RiskModule, TransactionRiskModuleToUse>() {{
                                    put(RiskModule.LRC, TransactionRiskModuleToUse.builder()
                                            .mandatory(true)
                                            .build());
                                    put(RiskModule.IPREP, TransactionRiskModuleToUse.builder()
                                            .mandatory(true)
                                            .build());
                                }}
                        )
                        .build())
                .typeData(SIBS3DSV2TransactionTypeData.builder()
                        .amount(100)
                        .currency("EUR")
                        .merchantName("")
                        .merchantLocation("")
                        .maskedPan("")
                        .serviceOperationPlayerCode("")
                        .mbWaySDKRegistrationDataV2(MBWaySDKRegistrationDataV2.builder().build())
                        .build())
                .build();
    }
}
```

Risk Modules in depth

Currently, there are two Risk Modules in the TrustFactor Services:

- `LRC`
- `IP Reputation`

LRC

With the current location of the user's Agents and the location of the transaction creation, a distance calculation is performed to find the nearest Agent to the transaction creation location. That way if the Agent is not within the margin of error the risk module returns a high risk result. This margin of error is a distance derived from the sum of the accuracy of both locations, the closest Agent and the transaction creation.

There are two ways we can get the location from the transaction. One way is to send the precise location of the transaction creation on the `riskModules` parameter with the `LRC` key and `RiskModuleInputLRC` object. The other way we take the transaction metadata IP to extrapolate a location from it. If the precise location is not sent the TrustFactor services will fallback to the transaction metadata IP for the location.

For the `RiskModuleInputLRC` the possible inputs are:

Parameter	Mandatory	Data Type	Data Format
`lat`	Yes	`float`	`Decimal Degrees`
`lon`	Yes	`float`	`Decimal Degrees`
`accuracy`	Yes	`float`	`Kilometers`

Keep in mind that the coordinates format used in the TrustFactor services is the `Decimal Degrees` (e.g. `lat = 41.25, lon = -8.5437`), and the accuracy is expressed in kilometers (e.g. `accuracy = 50`).

IP Reputation

Taking in account the transaction metadata IP we use threat intelligence sources to check if the source IP of the transaction has been flagged as the origin of suspicious activity. If the IP is flagged as suspicious, the risk module will return a high risk result.

Risk Modules parameters

We can divide the risk modules parameters in three parts:

- `riskModules` - Used for additional parameters for the TrustFactor risk modules calculation
- `riskModulesToCalculate` - Defines which risk modules should be used for transaction risk calculation and whether they are mandatory
- `riskModulesAdded` - Additional risk modules information that application might send that is directly mapped to mobiles agents UI

Risk Modules

This parameter's purpose is to send additional information that might be used during Risk Module processing, such as giving a precise location for the transaction origin.

The additional parameters are defined through the `TransactionAgentInput` object that contains an HashMap. This HashMap has `RiskModule` enum as its key and `IRiskModuleInput` interface as its value. The use of an interface as the value of the dictionary allows the creation of new inputs without the need of breaking changes for the already existing ones and support several classes generically.

The current possible classes for the `IRiskModuleInput` interface are:

Class	Corresponding Key
RiskModuleInputLRC	RiskModule.LRC
RiskModuleInputIPRep	RiskModule.IPREP

This risk module is supported in these transactions types:

- Actionable
- Generic
- SIBS 3DS
- SIBS 3DS V2

In the code snippet below we have an example of a generic transaction with only the risk modules parameter filled.

```
import com.securityside.trustfactor.constants.enums.RiskModule;
import com.securityside.trustfactor.model.transaction.creation.v3.GenericTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.IRiskModuleInput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.RiskModuleInputIPRep;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.RiskModuleInputLRC;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.input.TransactionAgentInput;

import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        GenericTransaction transaction = GenericTransaction.builder()
                .riskModules(TransactionAgentInput.builder()
                        .input(new HashMap<RiskModule, IRiskModuleInput>() {{
                            put(RiskModule.LRC, RiskModuleInputLRC.builder()
                                    .lat(41.25F)
                                    .lon(-8.5437F)
                                    .accuracy(50F)
                                    .build());
                            put(RiskModule.IPREP, RiskModuleInputIPRep.builder()
                                    .build());
                        }})
                        .build())
                .build();
    }
}
```

Risk Modules To Calculate

This parameter's purpose is to define which TrustFactor Risk Modules should be calculated and whether they are mandatory. A mandatory risk module means that if the transaction fails in the calculation of a Risk Module, the transaction will not be successful. This module only appears where there is no operation defined, as an operation has this functionality in it.

To define the risk modules to be calculated it is needed to instantiate a `TransactionRiskModulesToUse` object, that consists of a HashMap with `RiskModule` as key and `TransactionRiskModuleToUse` as the value. We can set that a risk module as mandatory by simply setting the `mandatory` parameter as true. If a risk module is absent from `riskModulesToCalculate` parameter it will not be calculated.

This risk module appears on the following transactions:

- Generic
- SIBS 3DS
- SIBS 3DS V2

In the code snippet below we have an example of a generic transaction with only the risk modules to calculate parameter filled.

```
import com.securityside.trustfactor.constants.enums.RiskModule;
import com.securityside.trustfactor.model.transaction.creation.v3.GenericTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.TransactionRiskModuleToUse;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.TransactionRiskModulesToUse;

import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        GenericTransaction transaction = GenericTransaction.builder()
                .riskModulesToCalculate(TransactionRiskModulesToUse.builder()
                        .transactionRiskModulesToUse(
                                new HashMap<RiskModule, TransactionRiskModuleToUse>(){{
                                    put(RiskModule.LRC, TransactionRiskModuleToUse.builder()
                                            .mandatory(true)
                                            .build());
                                    put(RiskModule.IPREP, TransactionRiskModuleToUse.builder()
                                            .mandatory(true)
                                            .build());
                                }}
                        )
                        .build())
                .build();
    }
}
```

Risk Modules Added

This module serves to add new Risk Modules that is directly mapped to mobiles agents UI. These risk modules can be used by the customer freely if they wish. For example if they want to alert in a more general way to something risky in the transaction.

To add additional risk modules to show to the user, one should instantiate a `TransactionAgentOutput` object that consists of an HashMap of string as keys and `TransactionRiskModuleOutput` as value. `TransactionRiskModuleOutput` contains only one parameter `agentModuleOutput` used in the Agents. `ModuleOutput` contains the information to be shown to the user, such as `risk`, `message` or a trigger with `RiskModuleOutputData` object.

`RiskModuleOutputData` class is a placeholder and should not be instantiated directly, instead it should be created a `Map` or `Webview` object and call the respective create output method. For the case of `Map` class call `CreateMapOutput` method and `CreateWebviewOutput` method for the `Webview` class.

- `Map` class has information about a location, for example the origin location of the transaction.
- `Webview` on the other hand has the information for a webview on the Agent device.

The `onTap` parameter is accessed when the user presses the Risk module on the Agent.

This risk module appears on the following transactions:

- Generic
- SIBS 3DS
- SIBS 3DS V2

In the code snippet below we have an example of a generic transaction with only the risk modules added parameter filled.

```
import com.securityside.trustfactor.constants.enums.RiskLevel;
import com.securityside.trustfactor.model.transaction.creation.v3.GenericTransaction;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.TransactionAgentOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.TransactionRiskModuleOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.ModuleOutput;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Coordinates;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Map;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Pinpoint;
import com.securityside.trustfactor.model.transaction.creation.v3.auxiliary
        .riskmodules.output.agent.eventdata.Webview;

import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        GenericTransaction transaction = GenericTransaction.builder()
                .riskModulesAdded(TransactionAgentOutput.builder()
                        .output(new HashMap<String, TransactionRiskModuleOutput>(){{
                            put("new_risk_module", TransactionRiskModuleOutput.builder()
                                    .agentModuleOutput(ModuleOutput.builder()
                                            .risk(RiskLevel.HIGH)
                                            .message("Some message to notify the risk level")
                                            .onTap(Map.builder()
                                                    .title("Origin Location")
                                                    .coordinates(Coordinates.builder()
                                                            .lat(41.25F)
                                                            .lon(-8.5437F)
                                                            .accuracy(50F)
                                                            .build())
                                                    .pinpoint(Pinpoint.builder()
                                                            .country("Portugal")
                                                            .city("Porto")
                                                            .build())
                                                    .build().CreateMapOutput())
                                            .build())
                                    .build());
                        }})
                        .build())
                .build();
    }
}
```

Guidelines/Tips

- App User ID is the unique identifier for a user in a TrustFactor application. It is recommended **not to re-use your internal user id**, so application users cannot be mapped from TrustFactor to your app without your internal mapping table.
- Keep track of the correlation id for each request, so it will be easier to debug when needed. It is highly recommended sending the correlation id for support purposes linked to the requests and the date.
- Every request timeout can be adjusted. The HTTP Client timeout cannot exceed 30 seconds due to infrastructure restrictions.
- In requests like `Reset Contract Device` where the application will receive a callback, it should be possible in 30 seconds for the application to send the request, TrustFactor services process it and the application respond to the callback so the application should respond to it in less than 30 seconds as defined in the backoffice.
- Avoid keeping track of contract keys since they are prone to desynchronize between Application/TrustFactor services.
- Custom headers on transaction creation can help keeping track of the decision process as the transaction callback brings all the custom headers. There are some blocked headers which are listed at the Attachment #2 in the *Attachments* chapter.
- Update to the latest version of each API as soon as possible. New features may only be available on newer API versions so it is highly recommended to keep the SDKs and methods up-to-date.
- When the user completes the registration flow, a screen with the registered device details should be presented to the user, so they can confirm that it was their device that registered. As the QR Code is presented in a screen anyone can read the same QR Code and register in the users account, so this serves as an additional validation by the user that the correct device was enrolled. The user should be given the opportunity to remove the recently registered device, but only for a short period of time, in case an attacker succeeds to register theirs instead of the legitimate user's device.
- The application should implement user notifications (like email / SMS message) when they receive the following callbacks, to inform users that changes have occurred with their TrustFactor enrollment for your app, for a given app user ID:
  - Share Contract
  - Contract Recover
  - Remove Device Contract
- Make Info/Debug logs of all requests/responses and callbacks from TrustFactor.

Attachments

In this section there will be information not deemed obligatory in the main document.

Attachment #1

Example operation


```
{
  "action": "transfer",
  "type": "actionable",
  "active": true,
  "version": 24,
  "default_language": "EN-US",
  "dactions": {
    "EN-US": "Transfer Edited",
    "PT-PT": "Transferência"
  },
  "dparams": [
    {
      "key": "destination_account",
      "data_type": "string",
      "description": "Destination Account",
      "languages": {
        "EN-US": "Destination Account",
        "PT-PT": "Conta Destino"
      },
      "optional": false
    },
    {
      "key": "amount",
      "data_type": "money",
      "description": "Amount",
      "languages": {
        "EN-US": "Amount",
        "PT-PT": "Montante"
      },
      "risk": {
        "low": {
          "type": "money-v2",
          "rule_logic": "vcae47dc7b473496cac7dee950beee972",
          "conditions": {
            "vcae47dc7b473496cac7dee950beee972": {
              "alias": "teste",
              "type": "lt",
              "value": {
                "EUR": 500,
                "USD": 1000
              },
              "aux_params": {
                "strict_check": true
              }
            }
          }
        },
        "medium": {
          "weight": 0.5,
          "type": "money-v2",
```

```
        "rule_logic": "v9be363bf165747acaa1e5d794d941eb0",
        "conditions": {
          "v9be363bf165747acaa1e5d794d941eb0": {
            "alias": "limite_inferior",
            "type": "gte",
            "value": {
              "EUR": 500,
              "USD": 1000
            },
            "aux_params": {
              "strict_check": true
            }
          }
        },
        "languages": {
          "EN-US": "Considerable high amount to transfer",
          "PT-PT": "Montante para transferir consideravelmente alto"
        }
      },
      "high": {
        "weight": 1,
        "type": "money-v2",
        "rule_logic": "ve47efc391c1248c9aa1b63fd81d75dba",
        "conditions": {
          "ve47efc391c1248c9aa1b63fd81d75dba": {
            "alias": "teste",
            "type": "gte",
            "value": {
              "EUR": 1000,
              "USD": 1500
            },
            "aux_params": {
              "strict_check": true
            }
          }
        },
        "languages": {
          "EN-US": "Large amount to transfer",
          "PT-PT": "Grande montante de transferência"
        }
      }
    },
    "optional": false
  },
  {
    "key": "source_account",
    "data_type": "string",
    "description": "Source Account",
    "languages": {
      "EN-US": "Source Account",
      "PT-PT": "Conta Remetente"
    },
    "optional": false
  }
],
"messages": {
```

```
    "EN-US": {
      "1": "Are you trying to transfer <b>{{.amount}}</b>?"
    },
    "PT-PT": {
      "1": "Está a tentar transferir <b>{{.amount}}</b>?"
    }
  },
  "transaction_duration": 30,
  "description": "Transfer",
  "risk": {
    "amount": 100
  },
  "risk_modules": {
    "ip_reputation": {
      "weight": 100,
      "mandatory": false
    },
    "lrc": {
      "weight": 100,
      "mandatory": false
    }
  }
}
```

Attachment #2

Blocked custom headers, returns as a parameter with error code `E_400090076`

```
[
  "Header", "Access-Control-Allow-Credentials", "Access-Control-Allow-Headers",
  "Access-Control-Allow-Methods", "Access-Control-Allow-Origin",
  "Access-Control-Expose-Headers", "Access-Control-Max-Age", "Accept-Ranges",
  "Age", "Allow", "Alternate-Protocol", "Cache-Control", "Client-Date", "Client-Peer",
  "Client-Response-Num", "Connection", "Content-Disposition", "Content-Encoding",
  "Content-Language", "Content-Length", "Content-Location", "Content-MD5",
  "Content-Range", "Content-Security-Policy", "X-Content-Security-Policy",
  "X-WebKit-CSP", "Content-Security-Policy-Report-Only", "Content-Type",
  "Date", "ETag", "Expires", "HTTP", "Keep-Alive", "Last-Modified", "Link",
  "Location", "P3P", "Pragma", "Proxy-Authenticate", "Proxy-Connection",
  "Refresh", "Retry-After", "Server", "Set-Cookie", "Status",
  "Strict-Transport-Security", "Timing-Allow-Origin", "Trailer",
  "Transfer-Encoding", "Upgrade", "Vary", "Via", "Warning", "WWW-Authenticate",
  "X-Aspnet-Version","X-Content-Type-Options", "X-Frame-Options",
  "X-Permitted-Cross-Domain-Policies", "X-Pingback", "X-Powered-By",
  "X-Robots-Tag", "X-UA-Compatible", "X-XSS-Protection", "X-Request-Correlation-ID"
]
```

Attachment #3

Error codes used in the SDK

Error Code	Description	Parameters
E_500010002	Concurrent writes on event store for the same aggregate id	None
E_500030003	The application cannot handle device removal at this time	Application Name
E_500040001	Invalid Operation	None
E_500050001	Invalid risk rule	None

Error Code	Description	Parameters
E_500050003	Mandatory risk module isn't registered	Risk Module
E_400010004	Application has no access to the transaction	None
E_400020001	There is an existing contract already for this user	App User ID
E_400020002	The specified contract doesn't exist	None
E_400020005	Missing contract's username	None
E_400020006	Invalid contract's app user ID	None
E_400020007	Invalid registration token duration. It should be between 0s and 600s.	None
E_400030010	Device does not exist	None
E_400040001	Device has no access to contract	Device key
E_400080010	Transaction doesn't support state update	None
E_400080011	Invalid state	None
E_400080012	Transaction is already in final state	None
E_400090007	Parameter key not found	Parameter key
E_400090010	Field is not valid	Field
E_400090013	SIBS 3DS Transaction not allowed	None
E_400090014	Generic Transaction not allowed	None
E_400090015	Empty action	None
E_400090016	No rule logic expression provided	None
E_400090017	Could not evaluate rule logic expression	None
E_400090018	Invalid value for data type condition	Data Type
E_400090019	You can't specify a message identifier if you specify the message for the transaction	None
E_400090020	You should either specify the message or a message identifier	None
E_400090022	Operation message identifier is not valid	Message Identifier
E_400090023	Invalid language tag	Language
E_400090024	Invalid message identifier	Message Identifier
E_400090026	You can't specify risk level for the transaction without overriding risk calculation	None
E_400090027	You can't specify risk level for the parameter without overriding risk calculation	Parameter
E_400090028	You can't specify risk message for the parameter without overriding risk calculation	Parameter
E_400090029	Invalid risk level for param and risk	Parameter, Risk Parameter
E_400090030	Invalid risk level	Risk Level
E_400090031	There is a parameter with no key specified	None
E_400090032	Missing required parameter in the transaction	Parameter
E_400090033	Specified parameter does not exist on the operation	Parameter
E_400090034	Invalid parameter data type	None
E_400090035	Error validating parameter, invalid data type	Parameter, Currency
E_400090036	Missing currency in aux fields	None
E_400090037	Invalid currency	None
E_400090038	Currency isn't valid	Currency
E_400090039	Transaction's message can't be empty	None
E_400090040	Transaction's action message can't be empty	None
E_400090041	Transaction duration is too low	Duration
E_400090042	Transaction duration is too high	Duration
E_400090044	Invalid value for money condition	None
E_400090045	Invalid currency for money condition	None
E_400090046	Invalid value for metadata attribute	Metadata Attribute
E_400090047	Invalid required authentication	Required Authentication
E_400090050	Couldn't calculate risk for a mandatory module	Risk Module

Error Code	Description	Parameters
E_400090051	Not found risk level for param	Parameter
E_400090052	Param doesn't fit in any rule	None
E_400090057	Invalid Transaction Type	Transaction Type
E_400090061	Masked PAN is empty	None
E_400090062	Service operation player code is empty	None
E_400090063	MBWay partner ID is empty	None
E_400090064	MBWay partner key is empty	None
E_400090065	MBWay phone prefix is empty	None
E_400090066	MBWay phone number is empty	None
E_400090067	MBWay application ID is empty	None
E_400090068	MBWay service ID is empty	None
E_400090069	MBWay PIN is empty	None
E_400090070	MBWay activation code is empty	None
E_400090071	SIBS 3DS Transaction version not allowed	None
E_400090072	Parameters order is referring an unknown parameter	Parameter
E_400090073	Parameter is not being referred on parameters order definition	Parameter
E_400090074	Parameters order doesn't start from 0 or doesn't contain sequential numbers	None
E_400090076	Blocked header	Header Key
E_400090077	Invalid transaction action	None
E_400090078	Parameter with null value	Parameter key
E_400090079	You can't specify any more authentication mechanisms besides "default"	None
E_400090080	You can't define risk message for a parameter without defining risk level	None
E_400090081	You can't define risk level (unless it's low level) for a parameter without defining risk message	None