



REST Connection

JAVA API Integration manual

**Version:** 1.0

**Date:** 01/06/2018

**Reference:** RS.TE.CEL.MAN.0025 RS.TE.CEL.MAN.0025



Redsys, Servicios de Procesamiento, S.L. – c/ Francisco Sancha, 12 – 28034 Madrid (España)

[www.redsys.es](http://www.redsys.es)

## Autorizaciones y control de versión

---

AUTOR: Redsys	VALIDATE BY: Comercio Electrónico	APPROVED BY: Redsys
Company: Redsys	Company: Redsys	Company: Redsys
Firma:	Firma:	Firma:
Date:	Date: 01/06/2018	Date: 01/06/2018

<b>Versión</b>	<b>Fecha</b>	<b>Afecta</b>	<b>Brief description of the change</b>
1.0	01/06/2020	ALL	First version

La propiedad intelectual de este documento pertenece a Redsys. Queda prohibida su reproducción, venta o cesión a terceros.

## ÍNDICE

<b>1. INTRODUCTION</b>	<b>4</b>
<b>2. API'S STRUCTURE</b>	<b>4</b>
2.1 ES.REDSYS.REST.API.CONSTANS PACKAGE	4
2.2 ES.REDSYS.REST.API.MODEL PACKAGE	5
2.3 ES.REDSYS.REST.API.MODEL.ELEMENT PACKAGE	5
2.4 ES.REDSYS.REST.API.MODEL.MESSAGE PACKAGE	5
2.5 ES.REDSYS.REST.API.SERVICE PACKAGE	5
2.6 ES.REDSYS.REST.API.SERVICE.IMPL PACKAGE	6
2.7 ES.REDSYS.REST.API.UTILS PACKAGE	6
2.8 ES.REDSYS.REST.TEST.EXAMPLE PACKAGE	6
<b>3. CREATION OF THE CONECTION SERVICE</b>	<b>7</b>
<b>4. PARAMETERS</b>	<b>7</b>
4.1 MANDATORY REQUEST PARAMETERS	8
4.2 OPTIONAL REQUEST PARAMETERS	9
4.3 PARAMETERS AND FUCTIONS. RESTOPERATIONSERVICE	10
4.4. PARAMETERS AND FUCTIONS. RESTINITIALREQUESTSERVICE	12
4.5.PARAMETERS AND FUNCTIONS. RESTAUTHENTICATIONREQUESTSERVICE	12
4.6 RESPONSE PARAMETERS	13
<b>5. EXAMPLES</b>	<b>15</b>
<b>6. REQUIREMENTS AND TECHNICAL SPECIFICATIONS</b>	<b>16</b>

## 1. Introduction

---

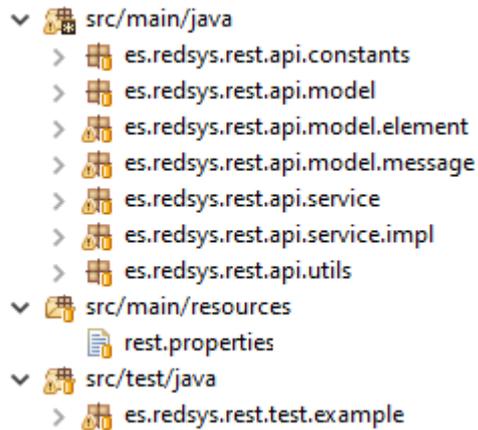
The present document define the REST integration guide between the commerce and Redsys's Virtual-POS using the JAVA's API.

## 2. API's Structure

---

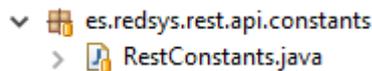
To use the API, the commerce must install in their system the packet that Redsys provides at your service. It's only necessary to add it to the library project of your Built Path.

Onces the library is import, the commerce can use the necessary classes and operatives to send the operation data. The API consist of the following elements:



*NOTE: At src/main/resources there is a file (rest.properties) with changeables proprieties of the API (response time, service definition...). Care must be taken to manage this file, a bad configuration can completely disable the API.*

### 2.1 es.redsys.rest.api.constans package



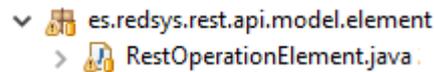
This package contains the constants's name and value that are use it in the rest's message Exchange (request and response).

## 2.2 es.redsys.rest.api.model package



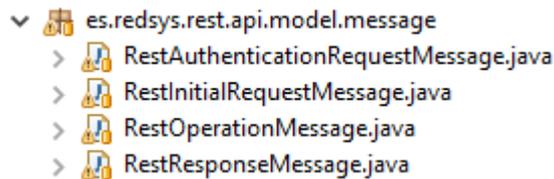
This package contains two definition interfaces relating to the two kind of message that can be used by the API (request and response).

## 2.3 es.redsys.rest.api.model.element package



The elements to catch and create the API's message response is in this package.

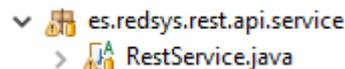
## 2.4 es.redsys.rest.api.model.message package



Inside this package there are the classes with the different kind of message that the API can use to Exchange between the commerce and Redsys's Virtual-POS. Within each class can be found the parammeters and methods can can be used for each message.

*NOTA: see "chapter 4: parameters", for a list of parameters in the request and response, also for more information about the methods that can be use in each operation.*

## 2.5 es.redsys.rest.api.service package



This package cotains the generical class for a rest call.



## 2.6 es.redsys.rest.api.service.impl package

- ▼  es.redsys.rest.api.service.impl
  - >  RestAuthenticationRequestService.java
  - >  RestInitialRequestService.java
  - >  RestOperationService.java

Here are the classes that implement the different types of operations that can be performed through the API.

## 2.7 es.redsys.rest.api.utils package

- ▼  es.redsys.rest.api.utils
  - >  RestSignatureUtils.java
  - >  RestTransformUtils.java
  - >  RestUtils.java

This package contains the necessary methods to make the call via REST.

## 2.8 es.redsys.rest.test.example package

- ▼  es.redsys.rest.test.example
  - >  Example3DSecureV1CardData.java
  - >  Example3DSecureV1Reference.java
  - >  Example3DSecureV2CardData.java
  - >  Example3DSecureV2ExemptionLWV.java
  - >  Example3DSecureV2Reference.java
  - >  ExampleCardValidationAndConfirmation.java
  - >  ExampleDCCDirectPayment.java
  - >  ExampleDirectPaymentAndRefund.java
  - >  ExampleDirectPaymentCardData.java
  - >  ExampleDirectPaymentReference.java
  - >  ExampleMOTOPaymentCardData.java

Finally, a test package (src/main/test) with different examples of how to use the API is included.

### 3. Creation of the connection service

---

To connect to Redsys, the abstract class RestService should be used, and depending on the type of operation to be performed, will be instantiated the following classes of the package *es.redsys.rest.api.service*:

- **RestInitialRequestService:** We will use this class to request about card information, for 3D Secure, exemptions, and for the use of DCC operations. The response of this request allow us to know the necessary request data of these functionalities that are required for the operation request (RestOperationService)
- **RestOperationService:** General class to carry out all general payment operations, both to an operation without authentication and with it. In the second case, depending on the response (frictionless or challenge), we will obtain the operation final response or a new request with the RestAuthenticationRequestService class will be necessary.
- **RestAuthenticationRequestService:** Once the operation have been performed with the "RestOperationService" class, we will use this class to authenticate the cardholder and obtain the final response of the operation.

When making a request, one of the previously exposed classes will be instantiated (depending of the kind of operation) with two parameters:

- **Commerce signatura key (signature):** Signature key for operations. This value can be found in the management module. Storing this value must be secure as a vulnerability in the trading system can cause the value to be compromised and lead to fraudulent operations.
- **Environment:** Environment to which the message will be sent, either test (SANDBOX) or production (PRODUCTION). *NOTE: Its posible to define a different url address (CUSTOM).*

### 4. Parameters

---

When sending a message through the API, the properties of that message must be set. Here are the list of mandatory parameters, which must be sent in any operation that wants to be performed through the API:

## 4.1 Mandatory Request Parameters

These parameters must always be sent for any type of operation:

- **Amount:** Amount of the operation. For an actual amount of 78.95 the value reported in the field will be 7895 without a decimal point. Decimals will depend on the code of the currency used.
- **Currency:** International code according to ISO-4217 regulations. The code for the euro currency is 978.
- **Merchant Code (merchant):** FUC code of the business to use, available through the administration module.
- **Terminal:** Numerical terminal code to be used, which can be consulted through the channel management module.
- **Order Number (order):** Alphanumeric that will have the same value as that sent in the first phase.
- **TransactionType:** Using the class *es.redsys.rest.api.constants.RestConstants.TransactionType* we will establish the type of operation to perform and can use the following values:
  - **Authorization** (AUTHORIZATION)
  - **Refund** (REFUND)
  - **Preauthorization** (PREAUTHORIZATION)
  - **Confirmation of preauthorization** (CONFIRMATION)
  - **Cancellation of preauthorization**(CANCELLATION)
  - **Delete a reference** (DELETE\_REFERENCE)
  - **Card validation** (VALIDATION)
  - **Confirmation of a card validation** (VALIDATION\_CONFIRMATION)

An example for using these parameters through API functions would be:

```
RestOperationMessage request = new RestOperationMessage();
request.setAmount("123");
request.setCurrency("978");
request.setMerchant("999008881");
request.setTerminal("20");
request.setOrder(orderID);
request.setCardNumber("4548810000000003");
request.setCardExpiryDate("3412");
request.setCvv2("123");
request.setTransactionType(TransactionType.AUTHORIZATION);
```

## 4.2 Optional Request Parameters

They are functions to add optional parameters to all requests:

- **Card data:** card number, expiry date and cvv2:

```
request.setCardNumber("4548810000000003");  
request.setCardExpiryDate("3412");  
request.setCvv2("123");
```

- **Create reference:** create a reference with the card data

```
request.createReference();
```

- **Use reference:** After obtaining the reference, it is possible to use it to continue carrying out future operations..

```
request.useReference("0197620cb6e8a4d74b5597ff9ed9b28671e52a37");
```

- **COF operation:** It is possible to carry out the first COF operation by the type of COF operation:

```
request.setCOFOperation(RestConstants.REQUEST_MERCHANT_COF_TYPE_INSTALLMENTS)
```

Tipo de COF	Valor DS_MERCHANT_COF_TYPE
Installments	I
Recurring	R
Reauthorization	H
Resubmission	E
Delayed	D
Incremental	M
No Show	N
Otras	C

For successive operations:

```
request.setCOFTxid("2006031152000");
```

- **idOper:** After obtaining the idOper using the Insite integration iframe, the token can be used as follows:

```
request.setOperID(operID);
```

- **Other parameters:** In case that you want to use an optional parameter that is not one of the previous parameters, the function "addParameter" can be used, sending in it the name of the parameter with its value as follows:

```
request.addParameter("DS_MERCHANT_PRODUCTDESCRIPTION", "Valor del parámetro opcional");
```

- **DS\_MERCHANT\_EMV3DS:** Some specific parameter can be added to this authentication element, you can use the following function for each of the request values. For example:

```
request.addEmvParameter("protocolVersion", "1.0.2");
```

*NOTE: For the rest of the optional parameters, see the parameter guide.*

### 4.3 Parameters and functions. RestOperationService

This class is for performing an operation, either authenticating or through direct payment:

- **Direct payment:** When this option is checked, the operation will be carried out in an unsafe manner as long as the merchant is configured for it. Authentication will not be requested from the cardholder.

```
request.useDirectPayment();
```

- **MOTO Payment:** The operation is carried out in an unsafe way for MOTO type operations:

```
request.useMOTOPayment();
```

- **Exemptions:** It is possible to use an authentication exemption:

```
request.setExemption("Valor de la exención a usar");
```

- **DCC:** Method for a payment operation through the DCC operation with the card values of the initial request:

```
request.dccOperation(dccCurrency, dccAmount);
```

- **Authentication parameters with protocolVersion 1.0.2:** In the case where a safe operation is required and the protocolVersion is 1.0.2.

```
request.setEMV3DSParamsV1();
```

- **Authentication parameters with protocolVersion 2.X.0:** In the case of a safe operation is required and the protocolVersion is 2.X.0, the parameters collected by the merchant of the 3DSMethod operation must be sent in the function.

*NOTE: The 3DSMethod operation must be performed by the merchant. For more information, see the "Integration Manual - REST. pdf"*

```
request.setEMV3DSParamsV2(protocolVersion, browserAcceptHeader, browserUserAgent, browserJavaEnable,  
browserJavaScriptEnabled, browserLanguage, browserColorDepth, browserScreenHeight,  
browserScreenWidth, browserTZ, threeDSServerTransID, notificationURL, threeDSCompInd);
```

*NOTE: It is possible not to use these functionalities to authenticate and instead use the function "addEmvParameter" to add the parameters of the authentication request one by one.*

#### 4.4. Parameters and fuctions. RestInitialRequestService

This class is created for operations based on a "InitialRequest", it is possible to use any of the previously written parameters (mandatory and optional), while using some specific ones of this operation:

- **3D Secure Card Data:** For this there is a specific function in this class that can be used to receive this information in the response.

```
request.demandCardData();
```

- **DCC Card Info:** To get DCC information:

```
request.demandDCCinfo();
```

- **Exemption Info:** Its possible to demand the list of exemption that the commerce can use:

```
request.demandExemptionInfo();
```

#### 4.5.Parameters and functions. RestAuthenticationRequestService

In the case where authentication by the client is required (challenge response), this class can be used. After processing the authentication values returned by the "RestOperationService" request (sending the parameters to be authenticated), the return parameters of the authentication url must be sent to each of these methods depending on the protocolVersion:

- **Ds\_Merchant\_Emv3ds version 1.0.2:** In this case, the response parameters should be sent after the authentication return as follows:

```
request.challengeRequestV1(pares, md);
```

- **Ds\_Merchant\_Emv3ds version 2.X.0:** In this case, the protocol version will have to be taken into account, and the cres value must be collected to proceed with its delivery:

```
request.challengeRequestV2(protocolVersion, cres);
```

## 4.6 Response parameters

The service will generate a "RestResponse" response message that the merchant must analyze. Examples of various types of operations are included in the es.redsys.rest.test.example package.

- **Operation result :** The analysis of the response must be performed depending on the result parameter (result) for each of the operations.

```
response.getResult ()
```

There are three possibilities:

- a. The operation has been treated correctly: the result parameter will be of type OK, the apiCode parameter will be 0 and the operation object will be informed with the response data of the operation.
- b. There has been some error in the operation: the result parameter will be of type KO, the apiCode parameter will have a value of type SISXXXX and the operation object will come to null.
- c. The operation requires authentication: The result parameter will be of type AUT, the apiCode will be 0 and in the operation object it will be informed with the necessary data for the authentication phase.

If the operation has been successful ("OK" / "AUT" values), the different functions that the api makes available to you can analyze the message.

- **Operaciones COF:** to obtain the id in COF operations:

```
response.getCOFTxnid();
```

- **ThreeDSInfo:** To get the response type:

```
String threeDSInfo = response.getThreeDSInfo();
```

- **ProtocolVersion:** In the case in which a "initialRequest" is carried out, the response will contain the specific parameter of the version to be used for the next step of the operation. In the case that you want to authenticate, it is necessary to obtain the value of this parameter:

```
protocolVersion = response.protocolVersionAnalysis();
```

- **ThreeDSServerTransID y threeDSMethodURL:** In the case that the protocol version is "2.X.0", it will be necessary to obtain the following values for the operation request:

```
threeDSServerTransID = response.getThreeDSServerTransID();
```

```
threeDSMethodURL = response.getThreeDSMethodURL();
```

- **Exemptions:** In the case where the request demand for the list of exemptions that can be used for commerce, it is possible to obtain this list using the following function:

```
response.getExemption();
```

- **DCC:** To obtain the card data applied by DCC in the response to the initialRequest:

```
response.getDCCurrency();  
response.getDCCAmount();
```

- **Obtaining parameters to perform the authentication:** The response of the operation can be of type "frictionless" or "challenge". On the first case, an OK response will be received, and in the second AUT. In the case of the challenge, it is necessary to collect the response parameters necessary to send them through a "POST" to the authentication url. These parameters will be different depending on the version of the protocolVersion:

```
//ProtocolVersion 1.0.2
String acsURL = response.getAcsURLParameter();
String pAReq = response.getPAReqParameter();
String md = response.getMdParameter();
String termUrl = "http://www.comercio.com/authentication-response.jsp";
```

```
//ProtocolVersion 2.X.0
String acsURL = response.getAcsURLParameter();
String cReq = response.getCReqParameter();
```

*NOTE: For information on the treatment of this data and on the flow of operations, see the "REST integration" guide.*

## 5. Examples

---

A series of examples are available where some cases of common operations and their form of integration using the API are found:

- **3DSecureV1CardData:** Example of an authorized payment with card data and cardholder authentication version 1.0.2
- **3DSecureV1Reference:** Example of an authorized payment with reference and cardholder authentication version 1.0.2
- **3DSecureV2CardData:** Example of an authorized payment with card data and cardholder authentication version 2.X.0
- **3DSecureV2Reference:** Example of an authorized payment with reference and cardholder authentication version 2.X.0
- **3DSecureV2ExemptionLWV:** Example of an authorized card payment where an authentication exemption is requested.

- **CardValidationAndConfirmation:** Example of a card validation, and confirmation of the validation.
- **DCCDirectPayment:** Example of an authorized payment without cardholder authentication and where DCC operation is used.
- **DirectPaymentAndRefund:** Example of an authorized card payment without authentication and refund.
- **DirectPaymentCardData:** Example of an authorized card payment without authentication.
- **DirectPaymentReference:** Example of an authorized payment with reference and without authentication.
- **MOTOPaymentCardData:** Example of an authorized MOTO payment (without authentication).

*NOTE: For execute the examples it's necessary to add at the JAVA's configuration argument "-Dhttps.protocols=TLSv1.1,TLSv1.2".*

*NOTE: The examples provided in the API are a representation of how to use the different API functions and should not be used as provided, since they do not contain security and business validations typical of the implementation of each trade. Redsys is not responsible for the use of these examples on the merchant's server as provided.*

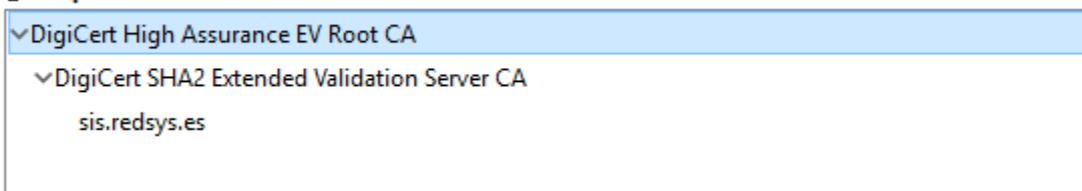
## 6. Requirements and technical specifications

---

The requirements that the commerce server must meet in order to use the Java connection API are the following:

- Have an application server capable of interpreting Java language in its version 7 or higher.
- Enable the outbound connection to the IPs of the Redsys servers, which are:
  - o 193.16.243.158
    - o 195.76.9.150
    - o 195.76.9.130
    - o 193.16.243.58
  - The CAs of the connection certificates are (can be obtained from <https://sis.redsys.es>):
    - o Autoridad de Certificación Raíz (DigiCert High Assurance EV Root CA)
    - o Autoridad de Certificación Intermedia (DigiCert SHA2 Extended Validation Server CA)

### Jerarquía de certificados



- The Cypher compatible list:
  - o TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384
  - o TLS\_ECDH\_RSA\_WITH\_AES\_256\_CBC\_SHA384
  - o TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA
  - o TLS\_ECDH\_RSA\_WITH\_AES\_256\_CBC\_SHA
  - o TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256
  - o TLS\_ECDH\_RSA\_WITH\_AES\_128\_CBC\_SHA256
  - o TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
  - o TLS\_ECDH\_RSA\_WITH\_AES\_128\_CBC\_SHA
  - o TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384
  - o TLS\_ECDH\_RSA\_WITH\_AES\_256\_GCM\_SHA384
  - o TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
  - o TLS\_ECDH\_RSA\_WITH\_AES\_128\_GCM\_SHA256