
InSite Integration

Version: 3.0

04/18/2022

RS.ACQUI.CNOPRESENCECOMM.LIST.0000



Redsys · C/ Francisco Sancha, 12 · 28034 · Madrid · Spain

Authorizations and version control

| Version | Date | It affects | Brief description of the change |
|---------|------------|---|--|
| 1.0 | 04/03/2017 | All document | Initial version of the document |
| 1.1 | 05/28/2018 | All document | Inclusion of Section 2 on Concepts and Advantages, and inclusion of improvements in the wording for greater clarity and understanding. |
| 1.2 | 06.03.2018 | URL is modified download of bookstores | The library download URL is modified (point 5) |
| 1.3 | 07.03.2019 | The access URL is included. test environment | Point 4 includes the URL to access the test environment. |
| 1.4 | 05.07.2019 | It is included new flow authentication of the SIS WS/REST | In point 6 the new authentication flow via WS/REST is included |
| 2.0 | 10/15/2019 | All document | Version 2 is included with new features. |
| 2.1 | 11/18/2019 | All document | Interface modifications and inclusion of new features |
| 2.2 | 09/12/2021 | Point 4.1 | A new optional parameter is included to hide the entity logo. |
| 2.3 | 04/18/2022 | Point 5 | Improvements section inSite version V3 |
| 3.0 | 04/18/2022 | All document | Document updated to inSite V3 with new features |

TABLE OF CONTENTS

| | |
|---|-----------|
| 1. Objective of this guide | 4 |
| 2. Concepts and advantages of inSite connection | 5 |
| 3. Flow Overview..... | 6 |
| 4. Payment Page – Obtaining Transaction ID | 7 |
| 4.1 Unified Integration (All-in-One) | 7 |
| 4.2 Integration of independent elements | 13 |
| 5. Error Catalogue..... | 16 |
| 6. Language Catalogue | 17 |
| 7. Sending an operation after generating an operation ID | 19 |
| 6.1 Implementation without using helper libraries | 19 |
| 8. Identifying On-Site Operations in the Virtual POS Administration Portal | 21 |
| 9. 3DSecure Authentication in InSite | 21 |
| 8.1 3DSecure v1.0.2 | 21 |
| 8.1.1 Request authorization | 21 |
| 8.1.2 Performing Authentication..... | 23 |
| 8.1.3 Post-Challenge 3DSecure 1.0 Authorization Confirmation | 24 |
| 8.2 EMV3DS | 25 |
| 8.2.1 Start Request..... | 25 |
| 8.2.2 Executing the 3DSMethod..... | 26 |
| 8.2.3 Authorization request with EMV3DS data | 28 |
| 8.2.4 Execution of the Challenge | 30 |
| 8.2.5 Post-Challenge EMV3DS Authorization Confirmation | 31 |

1. Objective of this guide

This document describes how to implement the connection in a web shop **inSite**Virtual POS, a connection model that allows the collection of customer payment data without the customer having to leave the merchant's website.

The advantages of this type of integration are numerous and are described in more detail in the following section. The main objective is to have a fast, simple, and fully integrated payment process within the web store's pages, fully adapted to the online store's design, easy to use and integrate, while maintaining the security of the payment information entered by the customer, avoiding the merchant having to endure costly security processes derived from mandatory compliance with the PCI DSS1 regulation.

This guide focuses on the specifics of this type of integration. For general information on how the SIS Virtual POS service works, please refer to the corresponding documentation.

¹PCI DSS (Payment Card Industry Data Security Standard) establishes the security requirements that Those involved in the card payment process must comply. The solution described in the document facilitates the consideration of the process as a SAQ-A type, by basing the implementation on iframes whose content is only accessible through our servers.

2. Concepts and advantages of inSite connection

With the payment solution **inSite** the online store or business gains a series of advantages that favor increased sales conversion:

- **Asimple and satisfying payment experience** for its customers, being **fully integrated** on the trade's web pages and without navigation jumps.
- **Greater control** of the checkout and payment flow, since every request is made synchronously by the web store server and without the need for asynchronous "listening" processes.
- **Ease of use in its integration,**
- **High level of security,** similar to the solution based on redirecting the customer to an external payment page.

In short, in addition to a fully integrated payment process from the buyer's checkout, the business is allowed greater **flexibility and control** in the payment process, also being able to separate the steps of data capture and execution of the operation.

When integrating the connection **inSite** exist **two possibilities**:

- Unified integration (all in one)
- Integration by independent elements

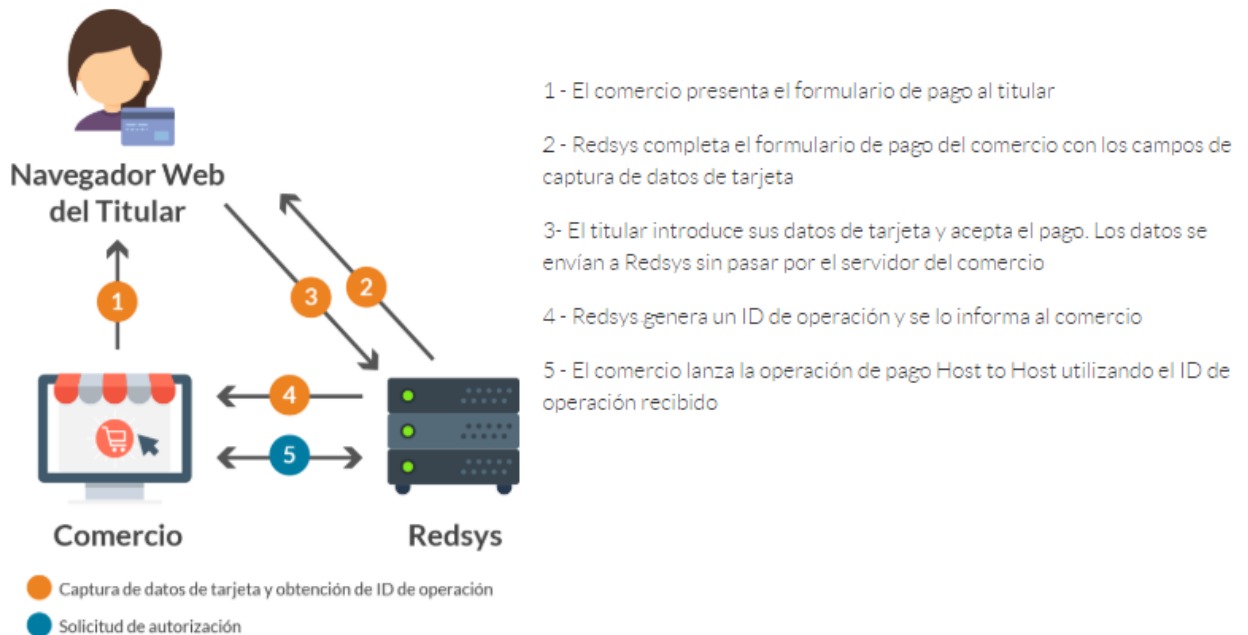
In both cases, integration can be performed using code snippets presented as examples, requiring only changing specific values such as the merchant ID or the keys used. Additionally, Redsys provides libraries for the main programming languages as additional assistance.

In the inSite connection, the online store is provided with the necessary pieces or "fields" of the payment form so that they integrate one by one (or as a whole) perfectly embedded in the online store's checkout page. In addition, each element allows for design customization with configurable styles, perfectly in tune with the design of the rest of the store's website.

Security is maintained so that the resulting form containing customer payment information is inaccessible to the merchant's own server or even to third parties who may have compromised the merchant's web server.

3. Flow overview

The following diagram presents the general flow of an operation carried out with the new Virtual POS scheme.



In short, the payment information entered by the customer is sent from the merchant's website to the Virtual POS, where it is temporarily stored and associated with a Transaction ID, which is then returned to the merchant. With this Transaction ID (which is an alias for the customer's payment information), the merchant can later request the desired payment transaction directly from the Virtual POS.

4. Payment Page – Obtaining Transaction ID

As a first step to integrate the card data entry fields directly into your own website, you must include the Javascript file hosted on the Redsys server with the following line of code (the file varies depending on whether you are using the test environment or the real production environment):

- Test Environment:

```
<script src="https://sis-t.redsys.es:25443/sis/NC/sandbox/redsysV3.js"></script>
```

- Production Environment:

```
<script src="https://sis.redsys.es/sis/NC/redsysV3.js"></script>
```

The next step in including payment form elements depends on the alternative you want to implement. When integrating the connection **inSite** exist **two possibilities**:

a) Unified integration (all in one) Payment elements, such as the card number, expiration date, CVV, and payment button input boxes, are embedded as a single responsive element, featuring a lightweight design and customizable CSS styles. Animated interactive help tools are included by default and offer excellent user experience.

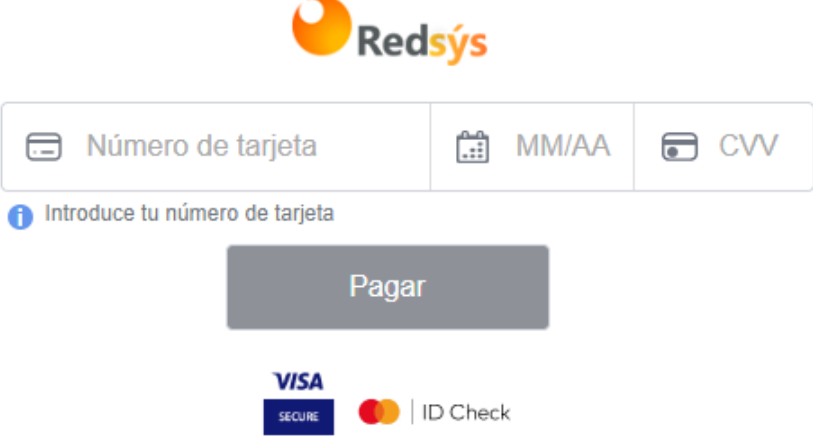
b) Integration by independent elements: Fields should be embedded independently within the webshop's webpage, allowing full control over the design, positioning, error handling, etc.

4.1 Unified Integration (All-in-One)

In this version of the inSite integration, a single, tightly sized iframe will be provided containing the entire payment form. As for customization, the merchant can apply CSS styles to the various elements.

It includes interactive elements that facilitate usability, such as card brand recognition, displaying the card logo, verifying formats and content, and visually highlighting any incorrect ones (check digit, expiration date, etc.).

Example:



The image shows a Redsys payment form. At the top is the Redsys logo. Below it is a form with three input fields: 'Número de tarjeta' (card number), 'MM/AA' (month/year), and 'CVV'. Below these fields is a blue information icon and the text 'Introduce tu número de tarjeta'. A large grey button labeled 'Pagar' (Pay) is centered below the form. At the bottom, there are logos for VISA SECURE, Mastercard, and ID Check.

Once the JS file has been imported, the payment form must be created. To securely collect card data, Redsys will create and host the data entry fields.

A single container must be created, with a unique ID, since it must be indicated so that an iframe is generated with the elements in it.

```
<div id="card-form-example"></div>
```

A message listener function must be included to receive the operation ID when it is generated. The function must be used *storeIdOper* with the following definition:

```
storeIdOper(event, idToken, idErrorMsg, validationFunction);
```

in which the event picked up by the listener must be indicated (*event*), the DOM element ID must be stored in the operation ID once it is generated (*idToken*), the identifier of the element in which error codes will be stored in case of data validation errors (*idErrorMsg*). In the example below, both are stored in a "hidden" input.

Optionally, the merchant can run a custom function to perform pre-validations. The transaction ID will only continue to be generated if the validation function executed by the merchant returns a value *true*.

```

<input type="hidden" id="token" ></input>
<input type="hidden" id="errorCode" ></input>
<script>
functionmerchantValidationExample(){
//Insert validations...
return true;
}
<!-- Listener -->
window.addEventListener("message", function receiveMessage(event) {
    storeIdOper(event,"token", "errorCode",merchantValidationExample);
});
</script>

```

Once the listener is ready to receive the data, the provided function will be called to generate the card data entry elements. Two functions are available, and you can use whichever you prefer. In the first, we'll pass each piece of data in a parameter (`getInSiteForm()`), and in the second we will pass the data in JSON format(`getInSiteFormJSON()`). The advantage of the latter is that we can pass only the data we need, without having to send empty parameters.

```

<!-- Classic iframe load request-->
    getInSiteForm(containerId, buttonStyle, bodyStyle, boxStyle, inputStyle, buttonValue, fuc,
    terminal, merchantOrder, insiteLanguage, showLogo, reducedStyle, insiteStyle);

<!-- Example -->
    getInSiteForm('card-form', "", "", "", 'Payment button text', '123456789', '1', 'ped4227', 'ES', true,
    false, 'twoRows');

<!-- JSON iframe load request-->
var insiteJSON = {
    "id": "card-form",
    "fuc": "123456789",
    "terminal": "1",
    "order": "ped4227",
    "InsiteStyle": "inline"
}

```

InSite Integration

```
getInSiteFormJSON(insideJSON);
```

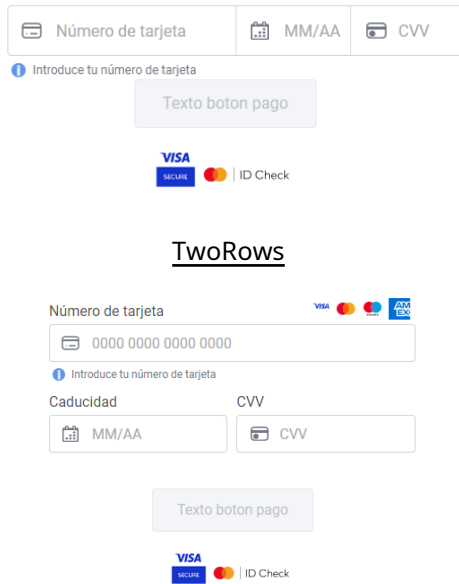
Function parameters will include the ID of the container reserved for its generation, as well as the required style for the different elements (CSS format). In this mode, styles can be included for different elements:

- *Payment button* → Full customization of the payment button is allowed.
- *Body of the form* → It is recommended to use it to set a background color or modify the color or style of texts.
- *Data entry box* → You can set a different background color for the input box. The text color applied to this element will be applied to the element's placeholder.
- *Data entry inputs* → It is recommended if you want to use a different font or change the text color in data entry fields.

Additionally, you can customize the text to be included in the button, and finally, you must enter the FUC value, terminal, and order number (an alphanumeric text string of between 4 and 12 digits) in the request to load the iframe with the payment form.

Four optional parameters are included in the iframe load. The optional parameters are as follows (in order):


- **Language:** Indicates the language of the texts. The relationship of codes can be found in the section [Language catalog](#) . Either the SIS code or the ISO 639-1 code of the language can be used.
If no language is specified or an incorrect code is entered, the default language will be Spanish.
- **Entity logo.** Establishes whether to display the entity's logo, indicating **true** if you want to show it or **false** if otherwise.
If no value is set, the entity's logo will be displayed by default.
- **Reduced style.** Indicates whether Insite should be displayed at a reduced width.
- **Insite style.** You can choose between two predefined Insite styles, '**inline**' or '**twoRows**'. By default, the inline style will be used. Below is an example of each Insite style, along with their respective reduced versions:

Inline


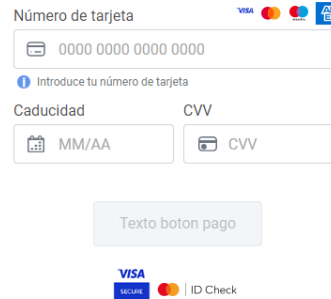
Inline payment form layout. It consists of three input fields in a single row: 'Número de tarjeta' (card number), 'MM/AA' (expiry date), and 'CVV'. Below the first field is a hint 'Introduce tu número de tarjeta'. A 'Texto boton pago' (payment button text) is centered below the inputs. At the bottom are logos for VISA, Mastercard, and American Express, along with an 'ID Check' label.

Reduced inline


Reduced inline payment form layout. It consists of two input fields in a single row: 'Número de tarjeta' (card number) and 'CVV'. The 'MM/AA' (expiry date) field is omitted. Below the first field is a hint 'Introduce tu número de tarjeta'. A 'Texto boton pago' (payment button text) is centered below the inputs. At the bottom are logos for VISA, Mastercard, and American Express, along with an 'ID Check' label.

TwoRows


TwoRows payment form layout. It consists of two rows of input fields. The first row contains 'Número de tarjeta' (card number) and logos for VISA, Mastercard, and American Express. The second row contains 'Caducidad' (expiry date) and 'CVV'. Below the first field is a hint 'Introduce tu número de tarjeta'. A 'Texto boton pago' (payment button text) is centered below the inputs. At the bottom are logos for VISA, Mastercard, and American Express, along with an 'ID Check' label.

TwoRows reduced


TwoRows reduced payment form layout. It consists of two rows of input fields. The first row contains 'Número de tarjeta' (card number) and logos for VISA, Mastercard, and American Express. The second row contains 'Caducidad' (expiry date) and 'CVV'. Below the first field is a hint 'Introduce tu número de tarjeta'. A 'Texto boton pago' (payment button text) is centered below the inputs. At the bottom are logos for VISA, Mastercard, and American Express, along with an 'ID Check' label.

You can see the list of parameters to send the data in JSON format below:

| Parameter | Mandatory / Optional | Default | Default |
|---------------------|----------------------|----------------------------------|---------|
| id | Mandatory | - | - |
| fuc | Mandatory | - | - |
| terminal | Mandatory | - | - |
| order | Mandatory | - | - |
| styleButton | Optional | CSS Style | " |
| styleBody | Optional | CSS Style | " |
| styleBox | Optional | CSS Style | " |
| styleBoxText | Optional | CSS Style | " |
| buttonValue | Optional | Payment button text | Pay |
| languageInsite | Optional | Language catalog | IS |
| showLogoInsite | Optional | true, false | true |
| ReducedInsite style | Optional | true, false | false |
| Insite style | Optional | inline, twoRows | inline |

*** The merchantOrder used in loading the iframe and generating the idOper must be reused in the subsequent authorization request.**

This way, when the customer enters their card details in the elements generated by Redsys and clicks the payment button, an ID associated with the transaction will be generated and stored in the merchant's form so that the customer can formalize the purchase without having to process card details.

4.2 Integration of independent elements

This inSite integration option allows merchants to fully customize their payment page, allowing them to freely position card entry fields and the payment button by generating distinct, customizable iframes with styles for each.

Also included are elements to improve usability, such as card brand recognition by displaying the card icon, CVV length based on the card brand, and the expiration date format.

Once the file is imported, the payment form must be created. To securely collect card data, Redsys will create and host the data entry fields.

Empty containers must be created, with a unique ID, as this must be indicated in order for the data entry field to be generated in it.

```
<div class="cardinfo-card-number">
  <label class="cardinfo-label" for="card-number">Card number</label>
  <div class="input-wrapper" id="card-number"></div>
</div>
<div class="cardinfo-exp-date">
  <label class="cardinfo-label" for="expiration-month">Expiration Month (MM)</label>
  <div class="input-wrapper" id="expiration-month"></div>
</div>
<div class="cardinfo-exp-date2">
  <label class="cardinfo-label" for="expiration-year">Expiration Year (YY)</label>
  <div class="input-wrapper" id="expiration-year"></div>
</div>
<div class="cardinfo-cvv">
  <label class="cardinfo-label" for="cvv">CVV</label>
  <div class="input-wrapper" id="CVV"></div>
</div>
<div id="button"></div>
```

This example uses separate date elements, one for month ("expiration-month") and one for year ("expiration-year").

If you want to display the month and year corresponding to the expiration date in the same field, an element is available that includes both values in mm/yy format.

To do this we will replace the elements "*expiration-month*" and "*expiration-year*" by the element "*card-expiration*".

```
<div class="cardinfo-card-number">
  <label class="cardinfo-label" for="card-number">Card number</label>
  <div class="input-wrapper" id="card-number"></div>
</div>

<div class="cardinfo-exp-date">
  <label class="cardinfo-label" for="card-expiration">Expiration</label>
  <div class="input-wrapper" id="card-expiration"></div>
</div>

<div class="cardinfo-cvv">
  <label class="cardinfo-label" for="cvv">CVV</label>
  <div class="input-wrapper" id="cvv"></div>
</div>

<div id="boton"></div>
```

A message listener function must be included to receive the operation ID when it is generated. The function must be used *storeIdOper* with the following definition:

```
storeIdOper(event, idToken, idErrorMsg, validationFunction);
```

in which the event picked up by the listener must be indicated (*event*), the DOM element ID must be stored in the operation ID once it is generated (*idToken*), the identifier of the element in which error codes will be stored in case of data validation errors (*idErrorMsg*). In the example below, both are stored in a "hidden" input.

Optionally, the merchant can run a custom function to perform pre-validations. The transaction ID will only continue to be generated if the validation function executed by the merchant returns a value *true*.

```
<input type="hidden" id="token" ></input>
<input type="hidden" id="errorCode" ></input>
<script>
function merchantValidationExample(){
//Insert validations...
return true;
}
```

InSite Integration

```
<!-- Listener -->
window.addEventListener("message", function receiveMessage(event) {
    storeIdOper(event,"token", "errorCode",merchantValidationExample);
});
</script>
```

Once the data has been sent and received, the provided functions will be called to generate the card data entry elements:

```
<!-- Request to load iframes -->
getCardInput('card-number', boxStyle, placeholder, inputStyle);
getExpirationMonthInput('expiration-month', CSSstyles, placeholder);
getExpirationYearInput('expiration-year', CSSstyles, placeholder);
getCVVInput('cvv', CSSstyles, placeholder);
getPayButton('button', CSS styles, 'Pay with Redsys', fuc, terminal, merchantOrder);
```

If the unified date element (mm/yy) is used, we will replace the `getExpirationMonthInput()` and `getExpirationYearInput()` functions with the `getExpirationInput()` function.

```
// Request to load iframes
getCardInput('card-number', boxStyle, placeholder, inputStyle);
getExpirationInput('card-expiration', CSSstyles, placeholder);
getCVVInput('cvv', CSSstyles, placeholder);
getPayButton('button', CSS styles, 'Pay with Redsys', fuc, terminal, merchantOrder);
```

Function parameters include the ID of the container reserved for its generation, the required style (CSS format), and the placeholder for that field. In the case of the `getCardInput()` function, two CSS style fields are passed: one with the style of the outer box and the other with the style of the input it contains. Additionally, the text of the payment button can be customized, and finally, the FUC value, terminal, and order number (alphanumeric with 4 to 12 digits) must be entered in the request to load the iframe with the payment button.

The merchantOrder used in generating the idOper must be reused in the subsequent authorization request.

This way, when the customer enters their card details in the elements generated by Redsys and clicks the payment button, an ID associated with the transaction will be generated and stored in the merchant's form so that the customer can formalize the purchase without having to process card details.

5. Error catalog

When you click the button generated by Redsys, validations will be launched based on the data entered. You may receive the following error list.

Error descriptions are included, but it is the merchant's responsibility to display them in the manner they deem appropriate.

| | |
|-------|--|
| msg1 | You must fill in the card details |
| msg2 | The card is mandatory |
| msg3 | The card must be numeric |
| msg4 | The card cannot be negative |
| msg5 | The card expiration month is mandatory |
| msg6 | The card expiration month must be numerical. |
| msg7 | The card expiration month is incorrect |
| msg8 | The card expiration year is mandatory |
| msg9 | The card expiration year must be numerical. |
| msg10 | The card expiration year cannot be negative. |
| msg11 | The card security code is not the correct length |
| msg12 | The card security code must be numeric. |
| msg13 | The card security code cannot be negative |
| msg14 | The security code is not required for your card |
| msg15 | The card length is not correct |
| msg16 | You must enter a valid card number (no spaces or hyphens). |
| msg17 | Incorrect validation by the merchant |
| msg18 | Domain initialization error |

6. Language catalog

| LANGUAGE | CODE | ISO 639-1 |
|------------|------|-----------|
| Spanish | 1 | IS |
| English | 2 | IN |
| Catalan | 3 | AC |
| French | 4 | FR |
| German | 5 | OF |
| Dutch | 6 | NL |
| Italian | 7 | ITEM |
| Swedish | 8 | SV |
| Portuguese | 9 | PT |
| Valencian | 10 | GOES |
| Polish | 11 | PL |
| Galician | 12 | GL |
| Basque | 13 | EU |
| Bulgarian | 100 | BG |
| Chinese | 156 | ZH |
| Croatian | 191 | HR |
| Czech | 203 | CS |
| Danish | 208 | DA |
| Estonian | 233 | ET |
| Finnish | 246 | FI |
| Greek | 300 | HE |
| Hungarian | 348 | HU |
| Indian | 356 | HI |
| Japanese | 392 | HA |
| Korean | 410 | KO |
| Latvian | 428 | LV |
| Lithuanian | 440 | LT |
| Maltese | 470 | MT |
| Romanian | 642 | RO |
| Russian | 643 | RU |

| | | |
|-----------|-----|----|
| Arab | 682 | AR |
| Slovak | 703 | SK |
| Slovenian | 705 | SL |
| Turkish | 792 | TR |

7. Sending an operation after generating an operation ID

Once the transaction ID has been received and stored by the merchant as described in the previous sections, they can launch the authorization transaction using any of the interfaces available in the Virtual POS.

In the authorization transaction, the DS_MERCHANT_IDOPER parameter must be sent instead of the usual card data fields. Additionally, the same order number (DS_MERCHANT_ORDER) used to generate the idOper must be used.

Bookstores that simplify this connection in Java and PHP are available for download. Downloads are available in the download section of the Redsys developer website:

<https://pagosonline.redsys.es/descargas.html>

The library download includes help documentation for its use.

6.1 Implementation without using helper libraries

If you don't want to use the help libraries or want to implement them for other programming languages, you can implement the REST call directly to the Virtual POS. **For more information**, it is recommended to consult the REST Integration documentation.

The authorization request is made through a request to the Virtual POS. In this request you must include the following parameters:

- Ds_SignatureVersion: Constant indicating the signature version being used.
- Ds_MerchantParameters: String in JSON format with all the request parameters encoded in Base 64 and without carriage returns (See Input and Output Parameters).
- Ds_Signature: Signature of the data sent. This is the SHA256 HMAC result of the Base64-encoded JSON string sent in the previous parameter.

These parameters must be sent to the following endpoints depending on whether you want to make a request in the test environment or real operations:

| Connection URL | Around |
|---|----------|
| https://sis-t.redsys.es:25443/sis/rest/trataPeticionREST | Evidence |
| https://sis.redsys.es/sis/rest/trataPeticionREST | Real |

Once the query is processed, the Virtual POS will inform the merchant's server of the result, including the information in a JSON file. The file will include the following fields:

- Ds_SignatureVersion: Constant indicating the signature version being used.
- Ds_MerchantParameters: String in JSON format with all the response parameters encoded in Base 64 and without carriage returns.
- Ds_Signature: Signature of the data sent. Result of the SHA256 HMAC of the Base64-encoded JSON string sent in the previous parameter. **The merchant is responsible for validating the HMAC sent by the Virtual POS to ensure the response is valid. This validation is necessary to ensure that the data has not been manipulated and that the source is truly the Virtual POS.**

The following describes the data that the Ds_MerchantParameters must include to send an authentication request to the REST Service:

```
{
  "DS_MERCHANT_ORDER": "1552572812",
  "DS_MERCHANT_MERCHANTCODE": "999008881",
  "DS_MERCHANT_TERMINAL": "2",
  "DS_MERCHANT_CURRENCY": "978",
  "DS_MERCHANT_TRANSACTIONTYPE": "0",
  "DS_MERCHANT_AMOUNT": "1000",
  "DS_MERCHANT_IDOPER": "455097a74c21b761be86acb26c32609dce222e66",
}
```

The answer will be:

```
{
  "Ds_Amount": "1000",
  "Ds_Currency": "978",
  "Ds_Order": "1552572812",
  "Ds_MerchantCode": "999008881",
  "Ds_Terminal": "2",
  "Ds_Response": "0000",
  "Ds_AuthorisationCode": "694432",
  "Ds_TransactionType": "0",
  "Ds_SecurePayment": "1",
  "Ds_Language": "1",
  "Ds_CardNumber": "454881*****0004",
  "Ds_Card_Type": "C",
  "Ds_MerchantData": "",
  "Ds_Card_Country": "724",
  "Ds_Card_Brand": "1"
}
```

8. Identification of in-site operations in the Virtual POS Administration Portal

In the Virtual POS administration portal, you can identify transactions performed with inSite by consulting the "entry" field of the transaction query.

Operations will be logged with the "inSite REST" entry

9. 3DSecure Authentication in InSite

Merchants using the inSite connection have the option to include the 3DSecure (3DS) protocol to authenticate cardholders and provide an additional level of fraud protection.

Including 3DS authentication involves redirecting the customer's navigation to the bank/card issuer's authentication server so that it can request the necessary credentials. This step must be performed after the card data collection step described in the previous sections.

To use 3DS authentication, the Virtual POS terminal must be configured by the financial institution to support 3D Secure authentication. Similarly, due to the Virtual POS configuration, this step may not only be optional but also required for proper transaction authorization (if you have any questions, consult the 3DS configuration with your financial institution that provides the Virtual POS).

8.1 3DSecure v1.0.2

8.1.1 Request authorization

The authorization request is made through a request to the Virtual POS. In this request you must include the following parameters:

- **Ds_SignatureVersion:** Constant indicating the signature version being used.
- **Ds_MerchantParameters:** String in JSON format with all the request parameters encoded in Base 64 and without carriage returns (See Input and Output Parameters).
- **Ds_Signature:** Signature of the data sent. This is the SHA256 HMAC result of the Base64-encoded JSON string sent in the previous parameter.

These parameters must be sent to the following endpoints depending on whether you want to make a request in the test environment or real operations:

| Connection URL | Around |
|---|----------|
| https://sis-t.redsys.es:25443/sis/rest/trataPeticionREST | Evidence |
| https://sis.redsys.es/sis/rest/trataPeticionREST | Real |

Once the query is processed, the Virtual POS will inform the merchant's server of the result, including the information in a JSON file. It will include the following fields:

- Ds_SignatureVersion: Constant indicating the signature version being used.
- Ds_MerchantParameters: String in JSON format with all the response parameters encoded in Base 64 and without carriage returns.
- Ds_Signature: Signature of the data sent. Result of the SHA256 HMAC of the Base64-encoded JSON string sent in the previous parameter. **The merchant is responsible for validating the HMAC sent by the Virtual POS to ensure the response is valid. This validation is necessary to ensure that the data has not been manipulated and that the source is truly the Virtual POS.**

The following describes the data that the Ds_MerchantParameters must include to send an authentication request to the REST Service:

```
{
  "DS_MERCHANT_ORDER":1552642885,
  "DS_MERCHANT_MERCHANTCODE":"999008881",
  "DS_MERCHANT_TERMINAL":"2",
  "DS_MERCHANT_CURRENCY":"978",
  "DS_MERCHANT_TRANSACTIONTYPE":"0",
  "DS_MERCHANT_AMOUNT":"1000",
  "DS_MERCHANT_IDOPER":" 455097a74c21b761be86acb26c32609dce222e66",
  "DS_MERCHANT_EMV3DS":{
    "threeDSInfo":"AuthenticationData",
    "protocolVersion":"1.0.2",
    "browserAcceptHeader":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8,application/json",
    "browserUserAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36" }
}
```

InSite Integration

The following response will be obtained:

```
{
  "Ds_Order": "1552642885",
  "Ds_MerchantCode": "999008881",
  "Ds_Terminal": "2",
  "Ds_Currency": "978",
  "Ds_Amount": "1000",
  "Ds_TransactionType": "0",
  "Ds_EMV3DS": {
    "threeDSInfo": "ChallengeRequest",
    "protocolVersion": "1.0.2",
    "acsURL": "https://sis.redsys.es/sis-simulator-web/authenticationRequest.jsp",
    "PAREq": "ejxVUttygjAQ/RWG95KEooKzpkPVjj7QOpZ+QBp2KIYuDVDx77tRqS0zmdmzj+zlnMBDXxycbzR
NXpUzV3jcdDbDUVZaXHzP3LX26C90HCenOIC5eUXcGJSTYN0oDnTybuU1Rq7zPsFGlFmIU+mOfi4j7vrAfn
3DOw9HYIbCjt/gI4dpKUifPBzZAqmn0TpWtBKW/HtfPMhgFYsIAXSEUaNYL+brcJstNnCy381X8nAK7pKF
UBcp5RUjnlZOuH5sO35XzZFSXIbAzD7rqytac5MQPgA0AOnOQu7atp4wdj0fPYNacGk9XBTBLAbvNtuls1F
CpPs9kso/7IzQ+Jftln6R09p88WcRHOjNg9gZkqkU5KOIIPhVi6kfAznIQhZ1BintPcNr0gqC2TeKBsszfDJAHHi
w6yWgS0hYDAuzrqkS6QbL+xkDOaEY73Cafr6zGuiXZ/loleYWLnPjK2WkzhBJC7ILABm/2VXJ9n1GVD07
3n8AOa7wW0=",
    "MD": "cd164a6d0b77c96f7ef476121acfa987a0edf602"
  }
}
```

8.1.2 Performing authentication

The merchant must set up a form that sends a POST to the `acsURL` parameter URL obtained in the response to the previous authorization request. This form sends three parameters required for authentication:

- *PaReq*, whose value is obtained from the *PAREq* parameter obtained in the response of the previous authorization request.
- *MD*, whose value is obtained from the *MD* parameter obtained in the response of the previous authorization request.
- *TermUrl*, which identifies the URL to which the issuing entity will POST the authentication result. This form will send a single parameter *Peers*, which contains the result of the authentication and must be collected by the merchant for later submission in the authorization confirmation request.

8.1.3 Post-Challenge 3DSecure 1.0 Authorization Confirmation

The following describes the data that the Ds_MerchantParameters must include to send a 3DSecure 1.0 authorization confirmation request to the REST Service:

```
{
  "DS_MERCHANT_ORDER":1552642885,
  "DS_MERCHANT_MERCHANTCODE":"999008881",
  "DS_MERCHANT_TERMINAL":"2",
  "DS_MERCHANT_CURRENCY":"978",
  "DS_MERCHANT_TRANSACTIONTYPE":"0",
  "DS_MERCHANT_AMOUNT":"1000",
  "DS_MERCHANT_EMV3DS":{
    "threeDSInfo":"ChallengeResponse",
    "protocolVersion":"1.0.2",
    "Pairs":"eJzFWNmSo0iyfecrymoenNVVsWqBNmWPBKlaJvcAbmwBJLALe9vWDIJVZWT3VNn3v
w70yyRR4uDvu
ESeOu8X2X0N+/dLFdZOVxctX9Dvy9UtchGWUFcnLV8vkvhFf//W6NdM6jhkjDu91/LpV4qbxk/hL .....",
    "MD":"035535127d549298f11d7d2fc1b0d4e9300f93f1" }
}
```

The final result of the operation will be obtained as a response:

```
{
  "Ds_Amount":"1000",
  "Ds_Currency":"978",
  "Ds_Order":"1552642885",
  "Ds_MerchantCode":"999008881",
  "Ds_Terminal":"2",
  "Ds_Response":"0000",
  "Ds_AuthorisationCode":"694432",
  "Ds_TransactionType":"0",
  "Ds_SecurePayment":"1",
  "Ds_Language":"1",
  "Ds_CardNumber":"454881*****0004",
  "Ds_Card_Type":"C",
  "Ds_MerchantData":"",
  "Ds_Card_Country":"724",
  "Ds_Card_Brand":"1"
}
```

8.2 EMV3DS

8.2.1 Start Petition

This request allows you to obtain the type of 3D Secure authentication that can be performed, as well as the URL of the 3DSMethod, if one exists.

The initial request is made through a REST request to the Virtual POS. In this request you must include the following parameters:

- Ds_SignatureVersion: Constant indicating the signature version being used.
- Ds_MerchantParameters: String in JSON format with all the request parameters encoded in Base 64 and without carriage returns (See Input and Output Parameters).
- Ds_Signature: Signature of the data sent. This is the SHA256 HMAC result of the Base64-encoded JSON string sent in the previous parameter.

These parameters must be sent to the following endpoints depending on whether you want to make a request in the test environment or real operations:

| Connection URL | Around |
|---|----------|
| https://sis-t.redsys.es:25443/sis/rest/inciaPeticonREST | Evidence |
| https://sis.redsys.es/sis/rest/inciaPeticonREST | Real |

Once the request is processed, the Virtual POS will inform the merchant's server of the result, including the information in a JSON file. It will include the following fields:

- Ds_SignatureVersion: Constant indicating the signature version being used.
- Ds_MerchantParameters: String in JSON format with all the response parameters encoded in Base 64 and without carriage returns.
- Ds_Signature: Signature of the data sent. Result of the SHA256 HMAC of the Base64-encoded JSON string sent in the previous parameter. **The merchant is responsible for validating the HMAC sent by the Virtual POS to ensure the response is valid. This validation is necessary to ensure that the data has not been manipulated and that the source is truly the Virtual POS.**

The following describes the data that the `Ds_MerchantParameters` must include to send a request to the REST Service:

```
{
  "DS_MERCHANT_ORDER": "1552571678",
  "DS_MERCHANT_MERCHANTCODE": "999008881",
  "DS_MERCHANT_TERMINAL": "2",
  "DS_MERCHANT_CURRENCY": "978",
  "DS_MERCHANT_TRANSACTIONTYPE": "0",
  "DS_MERCHANT_AMOUNT": "1000",
  "DS_MERCHANT_IDOPER": "455097a74c21b761be86acb26c32609dce222e66",
  "DS_MERCHANT_EMV3DS": {"threeDSInfo": "CardData"}
}
```

The following response will be obtained:

```
{
  "Ds_Order": "1552571678",
  "Ds_MerchantCode": "999008881",
  "Ds_Terminal": "2",
  "Ds_TransactionType": "0",
  "Ds_EMV3DS": {
    "protocolVersion": "2.1.0", "threeDSServerTransID": "8de84430-3336-4ff4-b18d-f073b546ccea", "threeDSInfo": "CardConfiguration",
    "threeDSMethodURL": https://sis.redsys.es/sis-simulator-web/threeDsMethod.jsp
  }
}
```

The parameter `Ds_EMV3DS` will be composed of the following fields:

- `protocolVersion`: This will always indicate the highest version number allowed for the transaction. The merchant is responsible for using the version number they are prepared for.
- `threeDSServerTransID`: EMV3DS transaction identifier.
- `threeDSInfo`: CardConfiguration.
- `threeDSMethodURL`: 3DSMethod URL.

8.2.2 Execution of the 3DSMethod

The 3DS Method is a process that allows the issuing entity to capture information about the device the cardholder is using. This information, along with the EMV3DS data sent in the authorization, will be used by the entity to perform a risk assessment of the transaction. Based on this, the issuer can determine that the transaction is trustworthy and therefore not require cardholder intervention to verify its authenticity (frictionless).

Device data capture is performed using a hidden iframe in the client's browser, which establishes a direct connection with the issuing entity in a manner that is transparent to the user. The merchant will receive a notification when the data capture is complete, and in the next step, when requesting authorization from the Virtual POS, the merchant must send the `threeDSCompInd` parameter indicating the execution of the 3DSMethod.

Steps to execute the 3DSMethod:

1. In the response received with the card configuration (iniciaPetición) the following data is received to execute the 3DSMethod:
 - a. **threeDSMethodURL**: url of the 3DSMethod
 - b. **threeDSSTransID**: EMV3DS transaction identifier

If **threeDSMethodURL** is not received in the response, the process ends. In the authorization, send **threeDSCompInd** = N
2. Build the JSON Object with the parameters:
 - a. **threeDSSTransID** : value received in the card query response
 - b. **threeDSMethodNotificationURL** : URL of the merchant that will be notified of the completion of the 3DSMethod from the entity
3. Encode the above JSON into Base64url encode
4. A hidden iframe must be included in the client's browser, and a field must be sent **threeDSMethodData** with the value of the previous json object, in an http post form to the url obtained in the initial query **threeDSMethodURL**
5. The issuing entity interacts with the browser to proceed with the data capture. Upon completion, the field will be sent **threeDSMethodData** in the browser's html iframe by http post to the url **threeDSMethodNotificationURL** (indicated in step 2), and the 3DSMethod terminates.
6. If the 3DSMethod has completed in less than 10 seconds it will be sent **threeDSCompInd** = Y in the authorization. If it has not been completed within 10 seconds you must stop waiting and send the authorization with **threeDSCompInd** = N

8.2.3 Authorization request with EMV3DS data

The authorization request is made through a REST request to the Virtual POS. In this request you must include the following parameters:

- Ds_SignatureVersion: Constant indicating the signature version being used.
- Ds_MerchantParameters: String in JSON format with all the request parameters encoded in Base 64 and without carriage returns (See Input and Output Parameters).
- Ds_Signature: Signature of the data sent. This is the SHA256 HMAC result of the Base64-encoded JSON string sent in the previous parameter.

These parameters must be sent to the following endpoints depending on whether you want to make a request in the test environment or real operations:

| Connection URL | Around |
|---|----------|
| https://sis-t.redsys.es:25443/sis/rest/trataPeticionREST | Evidence |
| https://sis.redsys.es/sis/rest/trataPeticionREST | Real |

Once the query is processed, the Virtual POS will inform the merchant's server of the result, including the information in a JSON file. The file will include the following fields:

- Ds_SignatureVersion: Constant indicating the signature version being used.
- Ds_MerchantParameters: String in JSON format with all the response parameters encoded in Base 64 and without carriage returns.
- Ds_Signature: Signature of the data sent. Result of the SHA256 HMAC of the Base64-encoded JSON string sent in the previous parameter. **The merchant is responsible for validating the HMAC sent by the Virtual POS to ensure the response is valid. This validation is necessary to ensure that the data has not been manipulated and that the source is truly the Virtual POS.**

The following describes the data that must be included in the Ds_MerchantParameters to send an authorization request with EMV3DS authentication to the REST Service:

```
{
  "DS_MERCHANT_ORDER":1552572812,
  "DS_MERCHANT_MERCHANTCODE":"999008881",
  "DS_MERCHANT_TERMINAL":2,
  "DS_MERCHANT_CURRENCY":978,
  "DS_MERCHANT_TRANSACTIONTYPE":0,
  "DS_MERCHANT_AMOUNT":1000,
  "DS_MERCHANT_IDOPER":" 455097a74c21b761be86acb26c32609dce222e66",
  "DS_MERCHANT_EMV3DS":
  {
    "threeDSInfo":"AuthenticationData",
    "protocolVersion":"2.1.0",
    "browserAcceptHeader":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8,application/json",
    "browserUserAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/71.0.3578.98 Safari/537.36",
    "browserJavaEnabled":false,
    "browserJavaScriptEnabled":false,
    "browserLanguage":"ES-es",
    "browserColorDepth":24,
    "browserScreenHeight":1250,
    "browserScreenWidth":1320,
    "browserTZ":52,
    "threeDSSTransID":"8de84430-3336-4ff4-b18d-f073b546ccea",
    "notificationURL":"https://comercio-inventado.es/recibe-respuesta-autenticacion",
    "threeDSCompInd":"Y"
  }
}
```

The answer will be:

- If you do a Frictionless, you will directly obtain the final result of the operation:

```
{
  "Ds_Amount":"1000",
  "Ds_Currency":"978",
  "Ds_Order":"1552572812",
  "Ds_MerchantCode":"999008881",
  "Ds_Terminal":2,
  "Ds_Response":"0000",
  "Ds_AuthorisationCode":"694432",
  "Ds_TransactionType":0,
  "Ds_SecurePayment":1,
  "Ds_Language":1,
  "Ds_CardNumber":"454881*****0004",
  "Ds_Card_Type":"C",
  "Ds_MerchantData":"",
  "Ds_Card_Country":"724",
  "Ds_Card_Brand":1
}
```

- If not, a Challenge will be requested:

```
{
  "Ds_Amount":"1000",
  "Ds_Currency":"978",
  "Ds_Order":"1552572812",
  "Ds_MerchantCode":"999008881",
  "Ds_Terminal":2,
  "Ds_TransactionType":0,
  "Ds_EMV3DS":{
    "threeDSInfo":"ChallengeRequest",
    "protocolVersion":"2.1.0",
    "acsURL":"https://sis.redsys.es/sis-simulator-web/authenticationRequest.jsp",

```

8.2.4 Execution of the Challenge

Step 1 .- Connection from the merchant to the ACS of the issuing bank

The next step is to connect from the merchant to the issuing entity so the customer can authenticate. This connection is made by sending an HTTP POST form to the bank's ACS URL. For this connection, we use the data received in the Ds_EMV3DS parameter from the previous step (acsURL and creg parameters):

Example:

With the data received in Ds EMV3DS it would be:

```
<form action="https://sis.redsys.es/sis-simulator-web/authenticationRequest.jsp" method="POST" enctype =
"application/x-www-form-urlencoded">

<input type="hidden" name="creq"
value="eyJ0aHJlZURTU2Y2dmVyVWhjbhbnNJRCl6ImU5OWMwZyZlZlTlFiZWItNGY4NS05ZmE3LTl3OTJiZjE5NDZlMlIsImFjc1Ry
YW5zSUZiOiYlMTQzZDNDPhY0wMjhlLTRmMGEOtEYnIiImDFkYmE5OTc2MTkiLCJtZXNzYWdlVHlwZSI6IkNSXZEiLCJtZXNzY
WdlVmVyc2l2bi6JiJlUzMS4wIiwiaWY2hhbGxlbmdlV2luZG93U2l6ZSI6J1InO">

</form>
```

Step 2 .- Execution of the challenge

The cardholder authenticates himself/herself using the methods required by his/her issuing entity: OTP, static password, biometrics, etc.

Step 3 .- Receiving the authentication result

Once the challenge is completed, the issuing entity will send the result to the merchant, making an http POST to the parameter URL *notificationURL* that the business previously sent in the authorization request:

"notificationURL": "https://comercio-inventado.es/recibe-respuesta-autenticacion"

InSite Integration

The merchant will receive the “cres” parameter, which will be used in the final authorization request, as described in the following section.

8.2.5 Post-Challenge EMV3DS Authorization Confirmation

The following describes the data that must be included in the Ds_MerchantParameters to send an EMV3DS authorization confirmation request to the REST Service:

```
{
  "DS_MERCHANT_ORDER":1552577128,
  "DS_MERCHANT_MERCHANTCODE":"999008881",
  "DS_MERCHANT_TERMINAL":"2",
  "DS_MERCHANT_CURRENCY":"978",
  "DS_MERCHANT_TRANSACTIONTYPE":"0",
  "DS_MERCHANT_AMOUNT":"1000",
  "DS_MERCHANT_PAN":"XXXXXXXXXXXXXXXXXX",
  "DS_MERCHANT_EXPIRYDATE":"XXXX",
  "DS_MERCHANT_CVV2":"XXX",
  "DS_MERCHANT_EMV3DS":{
    "threeDSInfo":"ChallengeResponse",
    "protocolVersion":"2.1.0",
    "cres":"eyJ0aHJlZURTU2VydmVvVHJhbnNJRiI6IjZGNDMwLTMzMzYtNGZmNC1iMTlkLWYwNzNiNTQ2Y2NIYSIsImFjc1RyYW5zSUQiOiJkYjVjOTljNC1hMmZkLTQ3ZWU0OTI2Zi1mYTBiMDk0MzUyYTAiLCJtZXNzYWdlVHlwZSI6IksNZXMiLCJtZXNzYWdlVmVyc2lvbiI6IjIuMS4wIiwidHJhbnNTdGF0dXMiOiJZIn0="
  }
}
```

The final result of the operation will be obtained as a response:

```
{
  "Ds_Amount":"1000",
  "Ds_Currency":"978",
  "Ds_Order":"1552572812",
  "Ds_MerchantCode":"999008881",
  "Ds_Terminal":"2",
  "Ds_Response":"0000",
  "Ds_AuthorisationCode":"694432",
  "Ds_TransactionType":"0",
  "Ds_SecurePayment":"1",
  "Ds_Language":"1",
  "Ds_CardNumber":"454881*****0004",
  "Ds_Card_Type":"C",
  "Ds_MerchantData":"",
  "Ds_Card_Country":"724",
  "Ds_Card_Brand":"1"
}
```

NOTE: For further details on the EMV3DS 2.0 protocol, please refer to the Virtual POS REST Integration guide.