Using the i4Go Technical Reference Guide

The *i4Go Technical Reference Guide* provides an introduction to i4Go®, reviews the implementation methods available to a developer, and reviews the implementation parameters developers will use to test and implement i4Go with Shift4.

Introduction to i4Go

i4Go was designed to secure cardholder data (CHD) in eCommerce business environments with web browser-based applications that integrate online (website) and onsite (kiosk, Software as a Service) technologies by intercepting CHD at the point of entry—before it ever enters the merchant's Web server or hosting provider's system—and working with Shift4's PA-DSS validated Universal Transaction Gateway® (UTG®) or API Web service* and PCI DSS-compliant Lighthouse Transaction Manager (LTM) payment gateway to replace the CHD with a payment token, a unique ID to reference the actual data. This will drastically reduce eCommerce merchants' PCI DSS scope and may qualify them to use the SAQ-A (EP).

The application will use the resulting payment token to process the transaction via UTG or API Web service and LTM. At no time does real CHD exist in the merchant's devices, applications, event logs, transport mechanisms, databases, Web servers, or hosting provider's systems.

*If the environment cannot support the use of a UTG, direct server-to-server Web service calls can be used to replace the CHD with a payment token and subsequently process the transaction.

WARNING! i4Go is designed to keep real and sensitive CHD information out of the merchant's Web server or hosting provider's system. If you send this information to the merchant's systems, you are defeating i4Go's purpose.

Security Best Practices

Please review these important details:

- Review the *Securing the Merchant's Site That Uses i4Go* document.
- Shift4 highly recommends that the merchant's server has a SSL certificate.

Other Implementation Requirements

Please review these important details, which are supplemental to the requirements outlined later in the document based on the implementation method chosen:

- The application must be able to exchange an Auth Token for an Access Token. (For additional information, see the *Prior to Implementation* section.)
- The merchant's server must have a valid Access Token and must pass the token to i4Go using the `i4go_accesstoken` parameter.
- The merchant's server must be able to perform a server-to-server call to obtain an access block. If the merchant's server communication package does not recognize Shift4's SSL certificate, a ZIP is available here under i4Go: https://myportal.shift4.com/index.cfm?action=support.security.
- For the Standard Direct Post implementation method, i4Go does an HTTP/HTTPS redirect back to the browser or merchant's server with the response data.
- The developer can implement i4Go on their payment information form such that their end users are not aware they are using i4Go; however, Shift4 recommends language be added introducing i4Go and briefly explaining how it protects CHD. (When using the iFrame implementation method, we have included content in each template which is hidden by default but can be displayed. We recommend you use the text that can be displayed in the templates.)

Prior to Implementation

For developers that are implementing i4Go and Shift4 API integration, the Shift4 API team will provide the developer with the Auth Token during the certification process.

For certification and production, an Auth Token will need to be exchanged for an Access Token. (See RESTful API in MyPortal API Corner for instructions on Access Token Exchange.)

Note: If only i4Go is being implemented and you are not planning to use the Shift4 API, it may be possible to generate an Access Token instead of generating an Auth Token and exchanging it for an Access Token. For additional information, contact your Shift4 API analyst.

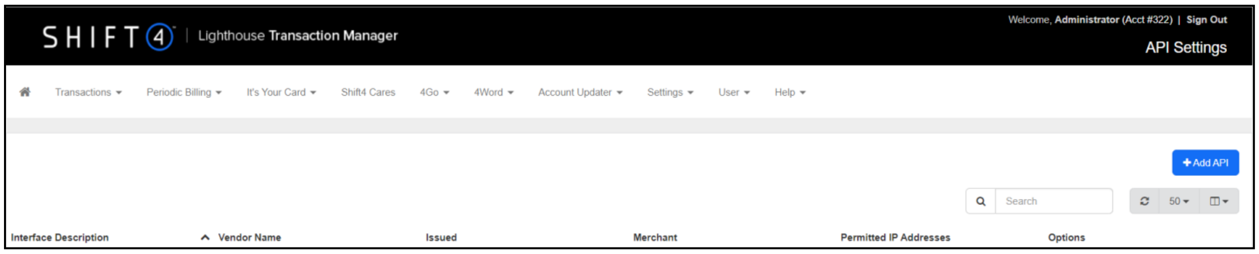
Note: If you are interested in using Shift4 Risk Management Services to perform a risk assessment during the i4Go tokenization process, contact your Shift4 API analyst for assistance. In addition, the `i4go_basket` is required to support this feature.

Note: If you are interested in having the payment card's bank verify the card as part of the payment process, see the *Card Verify* information in the *Implementing Step 1* section. In addition, the `i4go_basket` is required to support this feature.

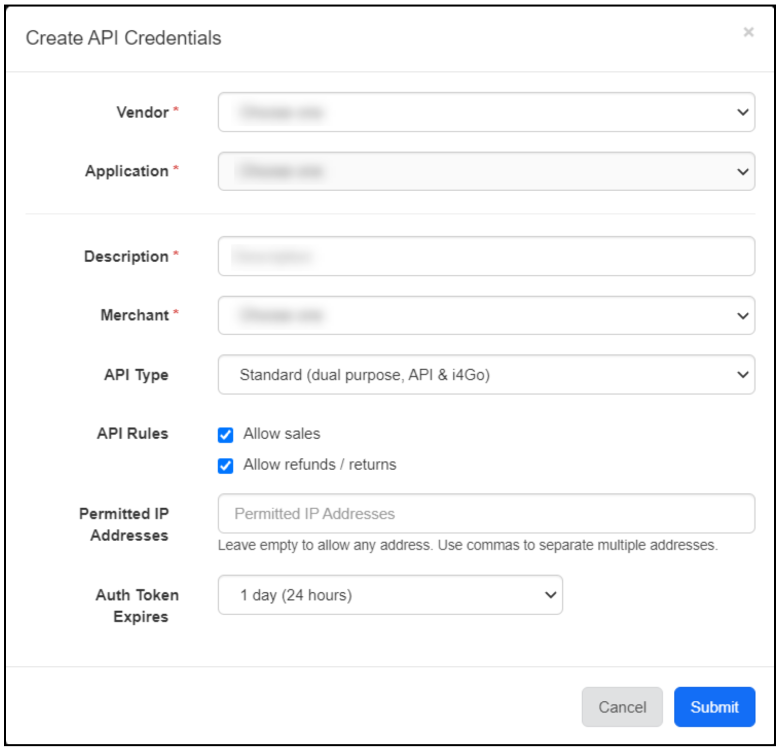
Note: If you are interested in using 3-D Secure (3DS), see the *3DS* information in the *Implementing Step 1* and *Implementing Step 3* sections. In addition, the `i4go_basket` is required to support this feature.

The process for generating an Auth Token for the production environment is briefly outlined below:

1. The Account Administrator signs in to LTM.
2. From the menu, the Account Administrator selects **Settings > API Settings**.
3. On the API Settings page, the Account Administrator clicks **Add API** to manually create API credentials.

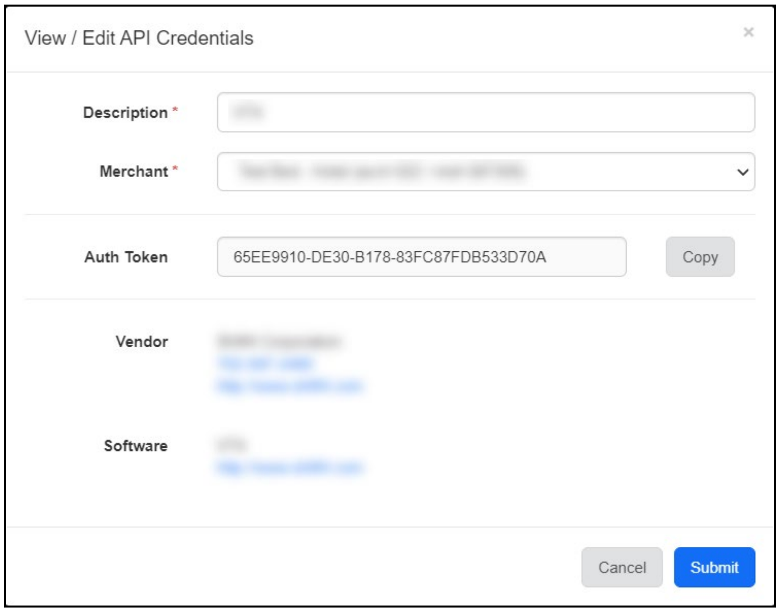


4. In the **Create API Credentials** window, the Account Administrator configures the applicable options and clicks **Submit**.


Create API Credentials

Vendor *	
Application *	
Description *	
Merchant *	
API Type	Standard (dual purpose, API & i4Go)
API Rules	✓ Allow sales ✓ Allow refunds / returns
Permitted IP Addresses	Permitted IP Addresses Leave empty to allow any address. Use commas to separate multiple addresses.
Auth Token Expires	1 day (24 hours)

5. In the View/Edit API Credentials window, the Account Administrator records and provides the Auth Token to the application and clicks **Submit**.


View / Edit API Credentials

Description *

Merchant *

Auth Token 65EE9910-DE30-B178-83FC87FDB533D70A

Vendor

Software

Note: For additional information on this process, the Account Administrator can see the *Account Administrator Guide*.

i4Go Implementation Methods

While i4Go implementation methods may vary, the use of i4Go is designed to be seamless whether the end user is entering their CHD on an eCommerce website or the end user is processing a customer’s purchase at a kiosk.

Note: The term “end user” refers to the person who is entering information on the payment information form.

i4Go supports the following implementation methods:

- *iFrame*
- *AJAX Using JSON*
- *Standard Direct Post*

Shift4’s preferred implementation method is using the iFrame to maximize i4Go’s benefit. However, if the iFrame method isn’t desired, then Shift4 recommends a combination of both technologies: AJAX using JSON for a seamless integration with better communication error trapping than Standard Direct Post, and Standard Direct Post as a fallback in the event that scripting is disabled on the client browser.

iFrame

Shift4's preferred implementation method is using the iFrame to maximize i4Go's benefit.

To implement i4Go using this method, there are five key steps; each step requires developers to implement certain functionality to ensure the payment information is tokenized. The five steps are briefly outlined below and described in greater detail in the following subsections.

Requirement: The payment information must be tokenized for each transaction in order to benefit from reduced PCI DSS scope.

1. The purchase is initiated at the point of sale in a web browser-based environment. The end user's IP address is sent with the merchant's Access Token through the merchant's server to the i4Go server, thus requesting authorization† for Step 4.
2. The i4Go server returns an access block and the i4Go server address to the merchant's server. The merchant's server must modify the values of the i4m initialization parameters to include the access block and to post to the returned i4Go server address.

Requirement: The merchant must add rate limiting logic at this point in the process to protect against the iFrame being continuously refreshed in an attempt to keep generating access blocks. For example, protect against three access blocks being generated in one second and ten access blocks being generated in one minute.

3. The CHD is entered on the payment information form (which is in an iFrame) and submitted directly to i4Go. i4Go sends the CHD to LTM where it is replaced with a payment token.

Tip: Swiping payment cards is supported. The end user simply clicks inside the iFrame and swipes the payment card using a point-to-point encryption (P2PE) enabled swipe reader or unencrypted magnetic swipe reader (MSR). Whether to support unencrypted MSRs can be configured. In addition, a script can be used to detect a card swipe without having to click inside the iFrame. For additional information, see the *Implementing Step 3* section.

⚠ **WARNING!** While i4Go's swipe collection logic supports both P2PE swipe readers and unencrypted swipe readers, Shift4 recommends always using P2PE swipe readers because the card data is encrypted inside the device, providing encryption from the point of swipe. These devices add another layer of security that unencrypted MSRs cannot provide. In addition, Shift4 recommends using P2PE devices with SRED as a function of the device, such as ID TECH SecuRED™ or SREDKey™.

4. The i4Go exit parameters are returned and mapped to the appropriate vendor-supplied form fields. (These fields will typically be hidden from the end user and only available for the vendor to retrieve and use for the next step.)
5. The application uses the payment token to process the transaction. (Note that processing the transaction happens outside of i4Go, either via local UTG or our server-to-server APIs.)

†This will attempt to authorize the end user's IP address to submit a single transaction through i4Go, regardless of where in the world the end user resides.

Implementing Step 1

Step 1: The purchase is initiated at the point of sale in a web browser-based environment. The end user's IP address is sent with the merchant's Access Token through the merchant's server to the i4Go server, thus requesting authorization for Step 4.

Please review these important details:

- This step is initiated from the merchant's Web server.
- This step requires an Access Token. (For additional information, see the *Prior to Implementation* section.)
- This step must be done for each tokenization attempt because the access block does expire.
- This step requires the use of `fuseaction=account.preauthorizeClient`, `i4go_clientip`, and `i4go_accesstoken` posted to https://access.shift4test.com (for certification) or https://access.i4go.com (for production). (The response will be JSON. For additional information, see the *i4Go Entry Parameters for the `preauthorizeClient` Request* section.)
    - *(Optional)* To include support for Card Verify, this step requires the use of:
        - `i4go_basket`
        - `paymentAPI` and setting `steps` to include `CARD_VERIFY`. (For additional information, see the corresponding row in the table in the *i4Go Entry Parameters for the `preauthorizeClient` Request* section.)
    - *(Optional)* To include support for 3DS, this step requires the use of:
        - `i4go_basket`
        - `paymentAPI` and setting `steps` to include `3DS_STANDALONE`. (For additional information, see the corresponding row in the table in the *i4Go Entry Parameters for the `preauthorizeClient` Request* section.)

Requirement: When 3DS is in use, the access block and/or invoice number should only be used one time. If an error occurs during the process or on the challenge window, a new access block and/or invoice number is needed.

-
    - *(Optional)* To include support for Apple Pay® and Google Pay™ wallets, this step requires the use of:
        - `i4go_basket` (For additional requirements, see the *Implementing Apple Pay and Google Pay Wallets* section.)
    - *(Optional)* To include support for Shift4 Risk Management Services, this step requires the use of:
        - `i4go_basket`
    - *(Optional)* To return a signed JSON Web Token (JWT) to ensure data is not tampered with during the tokenization process, this step requires the use of `i4go_hs256key`.

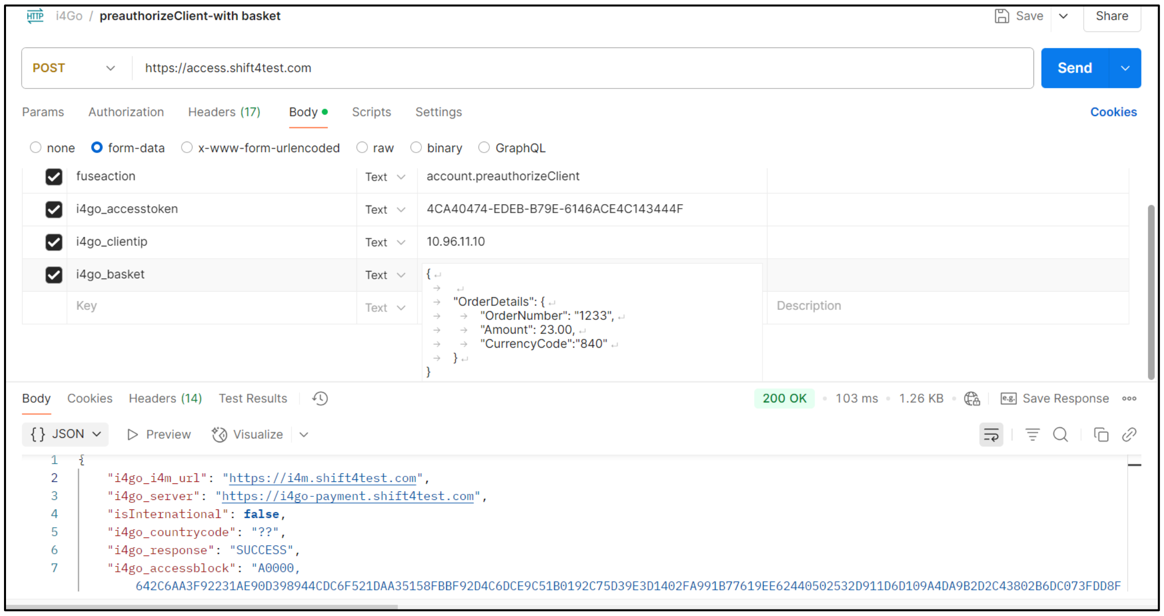
Note: For additional information on `paymentAPI`, see the *i4Go Entry Parameters for the preauthorizeClient Request* section.

Note: The *Implementing Apple Pay and Google Pay Wallets* section provides guidelines and other information pertinent to implementing wallet support to an existing i4Go integration.

Requirement: Any call to https://access.shift4test.com or https://access.i4go.com must be direct posted (using traditional form post with individual form fields) and cannot be JSON or XML (though the `paymentAPI` and `i4go_basket` mentioned above are serialized JSON strings). In addition, developers must ensure the application retains a log of all authorization requests, including the client IP address, for troubleshooting purposes.

Sample – `preauthorizeClient` Request

Note: The `preauthorizeClient` request is a post from the merchant's Web server, which will be in their language of choice. The example below is a Postman script.



POST https://access.shift4test.com

Key	Type	Value
fuseaction	Text	account.preauthorizeClient
i4go_accesstoken	Text	4CA40474-EDEB-B79E-6146ACE4C143444F
i4go_clientip	Text	10.96.11.10
i4go_basket	Text	{ "OrderDetails": { "OrderNumber": "1233", "Amount": 23.00, "CurrencyCode":"840" } }

```
{
    "i4go_i4m_url": "https://i4m.shift4test.com",
    "i4go_server": "https://i4go-payment.shift4test.com",
    "isInternational": false,
    "i4go_countrycode": "??",
    "i4go_response": "SUCCESS",
    "i4go_accessblock": "A0000,
        642C6AA3F92231AE90D398944CDC6F521DAA35158FBBF92D4C6DCE9C51B0192C75D39E3D1402FA991B77619EE62440502532D911D6D109A4DA9B2D2C43802B6DC073FDD8F
```

Card Verify

If you are interested in having the payment card's bank verify the card as part of the payment process, contact your Shift4 API analyst for assistance.

Additionally, see the *Verify Card with Processor* section in the REST API documentation, ensuring to click the **Card Number Unencrypted** option because that is what i4Go uses.

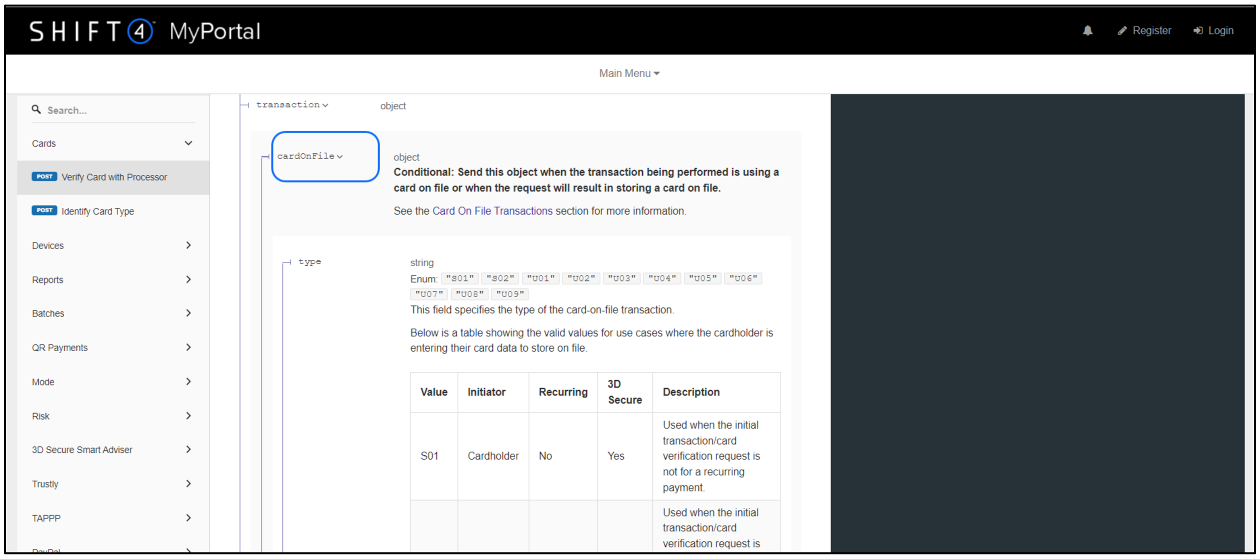
Shift4 REST API	i4Go API
`card.number`	Populated by the i4Go iFrame.
`card.expirationDate`	Populated by the i4Go iFrame.
`card.securityCode.indicator`	Populated by the i4Go iFrame.
`card.securityCode.value`	Populated by the i4Go iFrame.
`customer.addressLine1`	Populated by the i4Go iFrame or `basket.consumer.billingAddress.address1`.

Shift4 REST API	i4Go API
`customer.firstName`	Populated by `basket.consumer.billingAddress.firstName`.
`customer.lastName`	Populated by `basket.consumer.billingAddress.lastName`.
`customer.postalCode`	Populated by the i4Go iFrame or `basket.consumer.billingAddress.postalCode`.
`currencyCode`	Populated by `basket.orderDetails.currencyCode`.

`cardOnFile`

In addition, `cardOnFile` and its parameters may be used with Card Verify as needed. For additional information, complete the following steps:

1. Click this REST API documentation link: ***Cards.***
2. Scroll down to the *Verify Card with Processor* section.
3. Scroll down to the `transaction` object and click to expand it.
4. Scroll down to the `cardOnFile` object and click to expand it.



`transaction` object

`cardOnFile` object

Conditional: Send this object when the transaction being performed is using a card on file or when the request will result in storing a card on file.

See the Card On File Transactions section for more information.

`type` string

Enum: `"S01"` `"S02"` `"U01"` `"U02"` `"U03"` `"U04"` `"U05"` `"U06"` `"U07"` `"U08"` `"U09"`

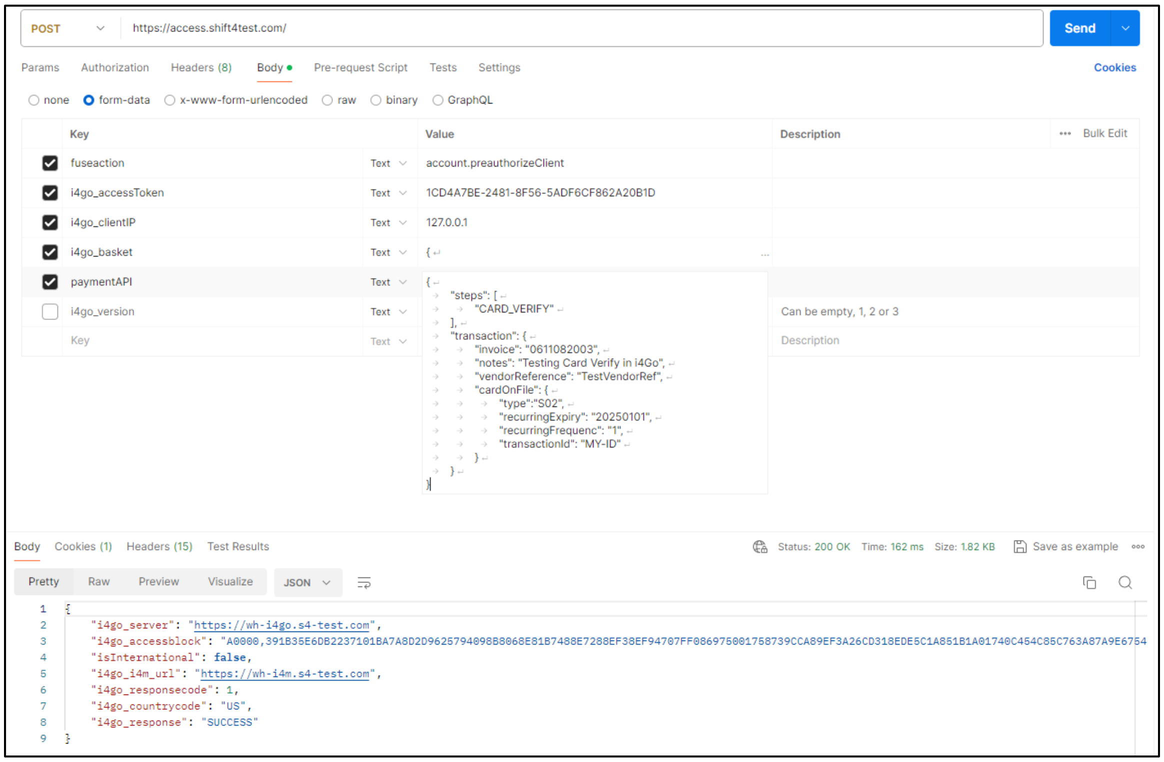
This field specifies the type of the card-on-file transaction.

Below is a table showing the valid values for use cases where the cardholder is entering their card data to store on file.

Value	Initiator	Recurring	3D Secure	Description
S01	Cardholder	No	Yes	Used when the initial transaction/card verification request is not for a recurring payment.
				Used when the initial transaction/card verification request is

`paymentAPI` Example for Card Verify Support

To support the Card Verify feature, `paymentAPI` and setting `steps` to include `CARD_VERIFY` are required. Optionally, `cardOnFile` and its parameters may be used if desired.



POST https://access.shift4test.com/

Body: form-data

Key		Value	Description
✓ fuseaction	Text	account.preauthorizeClient	
✓ i4go_accessToken	Text	1CD4A7BE-2481-8F56-5ADF6CF862A20B1D	
✓ i4go_clientIP	Text	127.0.0.1	
✓ i4go_basket	Text	{	
✓ paymentAPI	Text	(see below)	
i4go_version	Text		Can be empty, 1, 2 or 3

```
{
    "steps": [
        "CARD_VERIFY"
    ],
    "transaction": {
        "invoice": "0611082003",
        "notes": "Testing Card Verify in i4Go",
        "vendorReference": "TestVendorRef",
        "cardOnFile": {
            "type":"S02",
            "recurringExpiry": "20250101",
            "recurringFrequenc": "1",
            "transactionId": "MY-ID"
        }
    }
}
```

Status: 200 OK Time: 162 ms Size: 1.82 KB

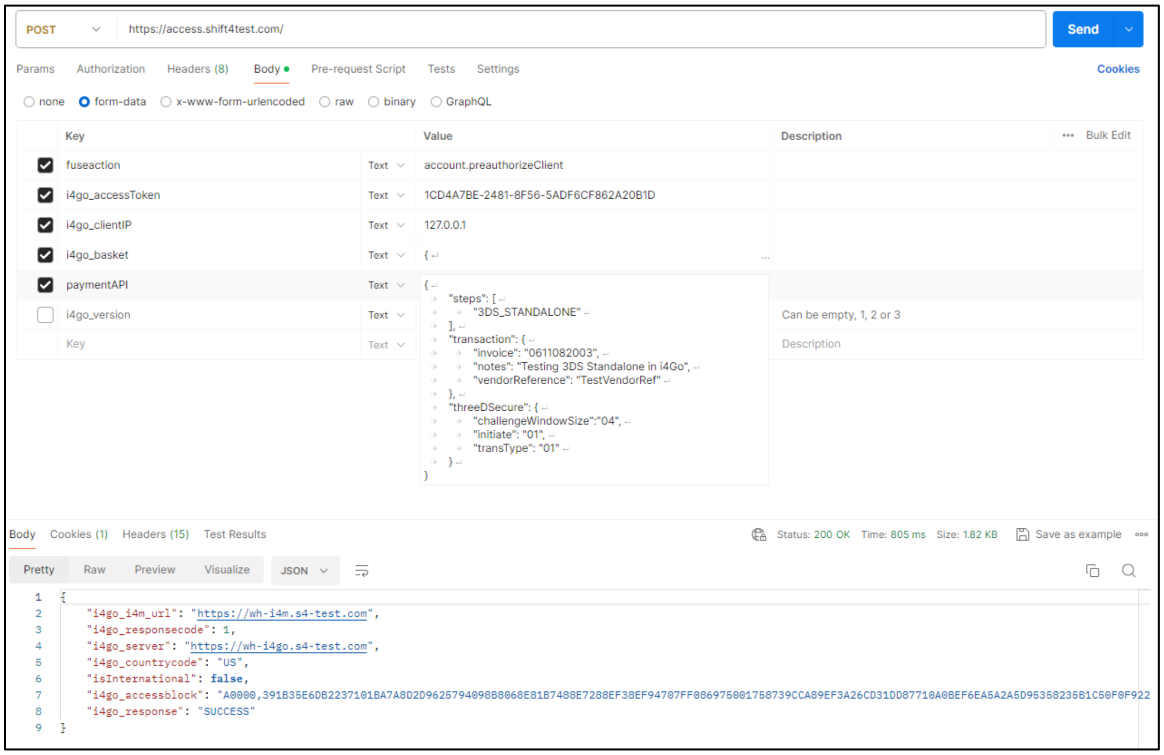
```
{
    "i4go_server": "https://wh-i4go.s4-test.com",
    "i4go_accessblock": "A0000,391B35E6DB2237101BA7A8D2D9625794098B8068E81B7488E7288EF38EF94707FF086975001758739CCA89EF3A26CD318EDE5C1A851B1A01740C454C85C763A87A9E6754
    "isInternational": false,
    "i4go_i4m_url": "https://wh-i4m.s4-test.com",
    "i4go_responsecode": 1,
    "i4go_countrycode": "US",
    "i4go_response": "SUCCESS"
}
```

3DS

If you are interested in 3DS, contact your Shift4 API analyst for assistance. For additional information, see the *3D Secure Standalone* section in the REST API documentation.

`paymentAPI` Example for 3DS Support

To support the 3DS feature, `paymentAPI` and setting `steps` to include `3DS_STANDALONE` are required.



POST https://access.shift4test.com/

	Key		Value	Description
✓	fuseaction	Text	account.preauthorizeClient	
✓	i4go_accessToken	Text	1CD4A7BE-2481-8F56-5ADF6CF862A20B1D	
✓	i4go_clientIP	Text	127.0.0.1	
✓	i4go_basket	Text	{	
✓	paymentAPI	Text	(see below)	
	i4go_version	Text		Can be empty, 1, 2 or 3
	Key	Text		Description

```
{
    "steps": [
        "3DS_STANDALONE"
    ],
    "transaction": {
        "invoice": "0611082003",
        "notes": "Testing 3DS Standalone in i4Go",
        "vendorReference": "TestVendorRef"
    },
    "threeDSecure": {
        "challengeWindowSize":"04",
        "initiate": "01",
        "transType": "01"
    }
}
```

Status: 200 OK Time: 805 ms Size: 1.82 KB

```
{
    "i4go_i4m_url": "https://wh-i4m.s4-test.com",
    "i4go_responsecode": 1,
    "i4go_server": "https://wh-i4go.s4-test.com",
    "i4go_countrycode": "US",
    "isInternational": false,
    "i4go_accessblock": "A0000,391B35E6DB2237101BA7A8D2D9625794098B8068E81B7488E7288EF38EF94707FF086975001758739CCA89EF3A26CD31DD87710A0BEF6EA5A2A5D95358235B1C50F0F922
    "i4go_response": "SUCCESS"
}
```

3DS Testing

To ensure the 3DS implementation is working, it must be tested using the applicable test card numbers as outlined here.

Important: When `3DS_STANDALONE` is included in the `preauthorizeClient` request, you cannot use non-3DS test card numbers to test.

Requirement: When `3DS_STANDALONE` or `CARD_VERIFY` is used, then the `cvv2Code` attributes `visible` and `required` must be set to `true`.

Note: If an international merchant uses Google Pay to attempt processing a transaction and PAN only is returned as what is being used to process the transaction, i4Go will attempt a 3DS call that includes `i4go_extendedcarddata`.

Remember, if Apple Pay or Google Pay wallet payment is used for a transaction, you must include `i4go_extendedcarddata` and `i4go_uniqueid` in your payment request. Failure to do so may cause the payment transaction to fail.

Implementing Step 2

Step 2: The i4Go server returns an access block and the i4Go server address to the merchant's server. The merchant's server must modify the values of the i4m initialization parameters to include the access block and to post to the returned i4Go server address.

Please review these important details:

- For a successful `preauthorizeClient` request, this step returns `i4go_response`, `i4go_responsecode`, `i4go_countrycode`, `i4go_accessblock`, and `i4go_server`.
  - The returned access block must be entered as the corresponding value to the `accessBlock` parameter.
  - The returned i4Go server address must be entered as the corresponding value to the `server` parameter.

Note: Modifying the values of the i4m initialization parameters is demonstrated in the *Sample Code - i4m Initialization and i4Go Exit Parameters* section.

Requirement: The merchant must add rate limiting logic at this point in the process to protect against the iFrame being continuously refreshed in an attempt to keep generating access blocks. For example, protect against three access blocks being generated in one second and ten access blocks being generated in one minute.

- For a failed `preauthorizeClient` request, this step returns `i4go_response` and `i4go_responsecode`. (For additional information, see the *Accepted i4Go Exit Parameters* section.)

Implementing Step 3

Step 3: The CHD is entered on the payment information form (which is in an iFrame) and submitted directly to i4Go. i4Go sends the CHD to LTM where it is replaced with a payment token.

Please review these important details:

- This step is initiated from the end user's browser session.
- jQuery 3.x or greater is required.

Important: jQuery 2.x does not support Internet Explorer 6, 7, and 8. If support for those browsers is needed, you will need to implement i4Go using the AJAX Using JSON and Standard Direct Post implementation methods. For additional information, see that section in this document.

- The i4m script must be in use, and it can be downloaded directly to the client by using the result in the `i4go_i4m_url` parameter like [i4go_i4m_url]/js/jquery.i4goTrueToken.js.
- *(Optional)* To configure 3DS options, set the following:
  - `threeDSecure.challengeMode`: Must be set to `redirect` or `iframe`. Default is `redirect`. If an invalid value is sent, `redirect` will be used.
    - `redirect`: This will redirect the entire page to 3DS.
    - `iframe`: This will allow an iFrame to be displayed on the website.
  - `threeDSecure.iframeElementId`: When `threeDSecure.challengeMode` is set to `iframe`, the element ID is required to specify where the 3DS iFrame should be added to on the website. If the element ID is not provided, `shift4-3ds-iframe` will be used and is the default. In addition, the HTML element should have `position: relative`, CSS attribute, and a height specified so that the iFrame displays properly. (The minimum window size that the 3DS API allows is 250px by 400px [width by height], so do not specify smaller than that.)
  - `threeDSecure.onOtpChallengeOpen`: A callback function to indicate to the merchant's website that the OTP challenge was opened. (For example, it allows the merchant to open a modal using their JavaScript library.)
  - `threeDSecure.onOtpChallengeClose`: A callback function to indicate to the merchant's website that the OTP challenge was completed/closed. (For example, it allows the merchant to close a modal using their JavaScript library.)
- *(Optional)* To support swiping payment cards when focus is anywhere on the page, the script must be in use and it can be downloaded directly to the client by using the result in the `i4go_i4m_url` parameter like [i4go_i4m_url]/js/jquery.cardswipe.js. If the script is not in use, then the end user must click inside the iFrame before swiping the payment card.
- To support unencrypted MSRs, the `encryptedOnlySwipe` parameter must be set to `false`. If `true`, then only P2PE swipe readers will be supported. (When set to `false`, unencrypted and encrypted MSRs are supported.)

- To only support card entry by use of an approved, secure device, set `deviceEntryOnly required` to `true`. By setting this to `true`, the card data will need to be captured on a card entry device (via dipping, swiping, manual entry, etc.) to continue with the transaction process. If `false`, manual entry of card data into the iFrame payment form will be allowed.

Tip: If you set `deviceEntryOnly required` to `true`, then you may want to set `cvv2Code visible`, `streetAddress visible`, and `postalCode visible` to `false`. This will allow the end user to bypass having to enter these values into the iFrame since the encrypted card reader will typically prompt for AVS and CVV when the payment card is manually keyed into the device. Conversely, when the payment card is swiped, the encrypted device will not prompt for AVS or CVV. Note that the swiped payment card data will typically come with a carriage return that will automatically submit the form.

- To support processing gift cards (including It's Your Card® gift cards) without requiring the end user to enter the card's expiration date, set the JavaScript flag `gcDisablesExpiration` to `true` and pass it to the iFrame. This will cause the Expiration field to be hidden in the iFrame when the end user selects Gift Card from the Payment Type list. If the flag is set to `false`, the Expiration Date field will be displayed in the iFrame regardless of which Payment Type is selected.
- To support processing gift cards (including It's Your Card gift cards) without requiring the end user to enter the card's security code, set the JavaScript flag `gcDisablesCVV2Code` to `true` and pass it to the iFrame. This will cause the Card Security Code field to be hidden in the iFrame when the end user selects Gift Card from the Payment Type list. If the flag is set to `false`, the Card Security Code field will be displayed in the iFrame regardless of which Payment Type is selected.
- Where you want the iFrame to be built, there must be an empty `div` with an `id` of `i4goFrame`.

Tip: The page that calls the iFrame can specify a width, and then the content will stay within that width. If `frameAutoResize` is set to `true`, the iFrame will expand vertically as needed. And while the content within the iFrame is responsive, you will need to use the Bootstrap templates and make the iFrame itself responsive. The responsive calculations within the iFrame are based on the iFrame width.

- There must be a hidden object called `i4goTrueToken`, which is where all the configurations occur:
  - `server`: Must be populated with the server address that is returned upon a successful `preauthorizeClient` request.

- `accessBlock`: Must be populated with the access block that is returned upon a successful `preauthorizeClient` request.
- `self`: Must be set to `document.location`.
- `template`: Must be populated with the name of the template to be used (see the *Example of i4Go in an iFrame* section for additional information):
  - `bootstrap3`
  - `bootstrap3-horizontal`
  - `bootstrap4`
  - `bootstrap5`
  - `bootstrap5-labeled`
  - `shift4shop`
  - `plain`
  - `table`

WARNING! The Bootstrap 5 template is not supported with Internet Explorer 11 or earlier.

- `url`: Must be populated with the appropriate i4m server address:
  - `https://i4m.shift4test.com` (for certification)
  - `https://i4m.i4go.com` (for production)

Note: Developers should only use `onSuccess` and `onFailure` or `formAutoSubmitOnSuccess` and `formAutoSubmitOnFailure`.

If the developer wants to control what happens when the tokenization request was successful and failed, `onSuccess` and `onFailure` should be used.

If the developer wants the merchant's payment information form (not the iFrame) to be automatically posted to the merchant's server (does not include any CHD), `formAutoSubmitOnSuccess` and `formAutoSubmitOnFailure` should be used.

The latter option is the simplest implementation for developers who are not JavaScript savvy.

- One of the following options:
  - `onSuccess` and `onFailure`: Must configure events for success and failure if using client side event handling of the tokenization request.
  - `formAutoSubmitOnSuccess` and `formAutoSubmitOnFailure`: Set to `true` to automatically post the merchant's payment information form (not the iFrame) to the merchant's server (does not include any CHD) if using server side event handling of tokenization events.
- *(Optional)* `onError`: If an error occurs within the iFrame (like the iFrame is submitted but a payment card number was not entered), `onError` will return a JSON object that includes an `errorCode` of `400` and an `errorList` array of possible validation errors.

Requirement: If a timeout or null response is received, the process to tokenize the payment information must begin again at step 1.

- *(Optional)* `language`: Set to the desired language. If not set, English is used.
  - "`en`" for English
  - "`es`" for Spanish
  - "`fr`" for French
  - "`pt`" for Portuguese
  - "`lt`" for Lithuanian

Tip: For an example, see https://myportal.shift4.com/index.cfm?action=development.i4mDemo.

- *(Optional)* `i4goInfo`: Set to `false` by default to hide the i4Go text and logo. May be set to `true` to display the content, which is recommended by Shift4. (For additional information, see the *Example of i4Go in an iFrame* section.)
- *(Optional)* The `cardNumber` field has attributes that can be changed:
  - `label`
  - `placeholder`
  - `message`

Note: When using the Table, Plain, Bootstrap3, or Shift4Shop templates and wanting to change the label of the Expiration fields in the iFrame, the `expirationMonth` and `expirationYear` labels are used.

When using Bootstrap4, Bootstrap4-lvloop, Bootstrap5, or Bootstrap5-labeled template and wanting to change the label of the Expiration field in the iFrame, the `expiration` label is used.

- *(Optional)* The `expirationMonth` and `expirationYear` fields have attributes that can be changed:
  - `label`
  - `placeholder`
  - `message`
- *(Optional)* The `expiration` field has attributes that can be changed:
  - `label`
  - `placeholder`
  - `message`

Tip: The configured `label`, `placeholder`, and `message` text is displayed to the end user. To support a different language, change the configured text. In addition, you can modify `cssRules` to make the iFrame match your site's design.

- *(Optional)* The `cardType`, `cvv2Code`, `cardholderName`, `streetAddress`, and `postalCode` fields also have attributes that can be changed:
  - `label`
  - `placeholder`
  - `message`
  - `visible`
  - `required`

Note: The `cardType` option does not have a `required` field. If `visible` is set to `true`, the field is required. For the `cvv2Code`, `cardholderName`, `streetAddress`, and `postalCode` options, if `visible` is set to `false`, the required setting is ignored.

Requirement: When `3DS_STANDALONE` or `CARD_VERIFY` is used, then the `cvv2Code` attributes `visible` and `required` must be set to `true`.

- *(Optional)* The `submitButton` (Secure My Payment Information) has the following attributes that can be changed:
  - `label`
  - `visible`

Note: If you hide the `submitButton` by setting `visible` to `false`, then you will need the following call from the parent page to the iFrame to tokenize the information:

- `i4goTrueTokenObj.submit();`

Sample Code – `jQuery`, `i4m`, and `cardswipe` Scripts

WARNING! The sample code was designed for demonstration purposes only. While you can use this code as a template, additional code will most likely need to be added to make this sample "production ready."

```
<script src="[jQuery file]" type="text/javascript"></script>
<script src="https://i4m.i4go.com/js/jquery.i4goTrueToken.js" type="text/javascript"></script>
<script src="https://i4m.i4go.com/js/jquery.cardswipe.js" type="text/javascript"></script>
```

Sample Code – i4m Initialization and i4Go Exit Parameters

WARNING! The sample code was designed for demonstration purposes only. While you can use this code as a template, additional code will most likely need to be added to make this sample "production ready."

```
var i4goTrueTokenObj = $("#form-main").i4goTrueToken({


        url:             "https://i4m.shift4test.com",   // i4go server address. If not provided, value
will be calculated based on the "server" parameter. Defaults to https://i4m.i4go.com (production). Must
override for testing and certification to https://i4m.shift4test.com

        server:         "https://i4go-payment.shift4test.com", // REQUIRED - get this from
access.i4go.com (production) or access.shift4test.com (certification) call

        accessBlock:
        "A0000,8DD3D28AC0813D6ADEECE5F9C78D9CAB26729AAE428F1DCECD138A61146D04C6DC9DC2649A67546D6A388DB81F2
578E22C5EF74D99D376E7CCF5A05B9CC9B081CAAEA735518E2ACF15D584DB3D66B4A10379A952E96A9611AD7E484E404E7798F0D27
DB7AF1AA4D1E56685252669BA7486FFD1F8C7A9D27C5C4D3454A9BCBBDE4A8422F025895F60E995D5BE0CCB6F3864AAEEEFDD415EA
1D8496C75F36F13CB0DF20C00EB619ADBF4892CE2A04FADA88D34CD36604C2DA839CCFDAEB8CEF396CD6429115192F4E4E1C0FAE52
E11CA16ADBE2C6D3C0E269309AC24FE60B431449EDC4BDF2A9A621BED6A1DB32A772D38443241BFCEB738E397B8A1C4A62035EA",
// REQUIRED - get this from access.i4go.com (production) or access.shift4test.com (certification) call)

        self:           document.location,      // REQUIRED – use document.location or URL of the current
page. If URL used, it MUST MATCH EXACTLY INCLUDING QUERY PARAMETERS

        template:       "bootstrap4",   // The template you want. (Currently defaults to bootstrap3-
horizontal. The Bootstrap 5 template is not supported with Internet Explorer 11 or earlier.)

                        // Options: bootstrap3, bootstrap3-horizontal, bootstrap4, bootstrap4-lvloop,
                bootstrap5, bootstrap5-labeled, shift4shop, table, plain


        threeDSecure:   {

                challengeMode:  "iframe",

                iframeElementId: "shift4-3ds-iframe",

                onOtpChallengeOpen:     function () {

                        console.log("OTP challenge opened");

                        // … your code here …

                },

                onOtpChallengeClose:    function () {

                        console.log("OTP challenge closed");

                        // … your code here …

                }

        },
        gcDisablesExpiration:    true,   // Set to true to cause the Expiration field to be hidden in the
iFrame when the end user selects Gift Card from the Payment Type list. If the flag is set to false, the
Expiration field will be displayed in the iFrame regardless of which Payment Type is selected.

        gcDisablesCVV2Code:      true,   // Set to true to cause the Card Security Code field to be hidden
in the iFrame when the end user selects Gift Card from the Payment Type list. If the flag is set to false,
the Card Security Code field will be displayed in the iFrame regardless of which Payment Type is selected.

        encryptedOnlySwipe:      true,

        deviceEntryOnly:{ classes:"", label:"", required:false },        // If you set required to true,
the card data will need to be captured via dipping, swiping, manual entry, etc. on a secure and approved
card entry device to continue with the transaction process. If false, manual entry of card data into the
iFrame payment form will be allowed. To customize the message displayed to the end user, use label.

        frameContainer:  "i4goFrame",    // Only used if frameName does not exist

        frameName:       "",     // Auto-assigned if left empty

        frameAutoResize: true,   // iframe will expand vertically as needed so content fits

        submitButton:    {

                label:"Secure My Payment Information"

                visible:true

        },      // The text in quotes will be the text on the button. Button is hidden after submitting
form. Button can be visible or not; if not, then the following call is needed from the parent page to the
iFrame to tokenize the information: i4goTrueTokenObj.submit();

        frameClasses:    "",


        formAutoSubmitOnSuccess: false,

        formAutoSubmitOnFailure: false,


        onSuccess:       function( form,data ) {

                console.log("i4goTrueToken- onSuccess()",data);

        },

        onFailure:       function( form,data ) {

                console.warn("i4goTrueToken- onFailure()",data);

        },

        onComplete:      function( form,data ) {

                console.log("i4goTrueToken- onComplete()",data);

        },

        onError:         function( response ) {

                console.log("i4goTrueToken- onError()",response);

        },


        //Wallet will asynchronously trigger individual callbacks.

        onWalletInit:    function( wallet:string, enabled:boolean, reason:string ) {

                console.log("i4goTrueToken- onWalletInit()",wallet);
```

```
},

//This is used for wallet support. See the Dynamic Shipping and Sales Tax section in Appendix D
for more information.
onPaymentDataChanged:    function( iobj, intermediatePaymentData, paymentDataRequestUpdate ) {
     console.log("i4goTrueToken- onPaymentDataChanged()",intermediatePaymentData);
     return paymentDataRequestUpdate;
},

language:        "en",   // Set to desired language: en for English, es for Spanish, fr for
French, pt for Portuguese, and lt for Lithuanian. If not set, English is used.
acceptedPayments:        "AX,DC,GC,JC,MC,NS,VS",

// Auto populated form fields. Precedence: field name, field id
formPaymentResponse:     "customNameFori4go_response",
formPaymentResponseCode: "customNameFori4go_responsecode",
formPaymentResponseText: "customNameFori4go_responsetext",
formPaymentMaskedCard:   "customNameFori4go_maskedcard",
formPaymentToken:        "customNameFori4go_uniqueid",
formPaymentExpMonth:     "customNameFori4go_expirationmonth",
formPaymentExpYear:      "customNameFori4go_expirationyear",
formPaymentType:         "customNameFori4go_cardtype",
formCardholderName:      "customNameFori4go_cardholdername",
formStreetAddress:       "customNameFori4go_streetaddress",
formPostalCode:          "customNameFori4go_postalcode",
formMetaToken:           "customNameFori4go_metatoken",
formExtendedCardData:    "customNameFori4go_extendedcarddata",
formApplePayToken:       "customNameFori4go_applepaytoken",
formGooglePayToken:      "customNameFori4go_googlepaytoken",


// Advanced Options
/*
    payments: [
        { type: "VS", name: "Visa" },
        { type: "MC", name: "MasterCard" },
        { type: "AX", name: "American Express" },
        { type: "DC", name: "Diners Club" },
                { type: "NS", name: "Discover" },
                { type: "JC", name: "JCB" },
                { type: "GC", name: "Gift Card" }
            ],
            body:             { styles:{ "background-color": "##aaa", borderLeft: "5px solid ##ccc" } },
            label:            { classes:"control-label col-xs-4" },
            container:        { classes:"" },
            cardType:         { classes:"", label:"", placeholder:"", message:"", visible:true },
            // If visible, which is the default setting, it is required.
            cardNumber:       { classes:"", label:"", placeholder:"", message:"" },
            // Always visible and always required.
            expirationMonth:  { classes:"", label:"", placeholder:"", message:"" },
            // Always visible and always required.
            expirationYear:   { classes:"", label:"", placeholder:"", message:"" },
            // Always visible and always required.
            cvv2Code:         { classes:"", label:"", placeholder:"", message:"", visible:true,
required:true },
            // Can be visible or not; if visible, can be required or not. Default settings are visible and
required. If not visible, required setting is ignored. (If you set deviceEntryOnly required to true, and
you plan to require the card security code to be entered in the secure device, then you may want to set
cvv2Code visible to false. This will allow the end user to bypass having to enter the card security code
in the iFrame since it was already captured via the device.)
            cardholderName:   { classes:"", label:"", placeholder:"", message:"", visible:false,
required:true },
            // Can be visible or not; if visible, can be required or not. Default setting is not visible,
which means required setting is ignored.
            streetAddress:    { classes:"", label:"", placeholder:"", message:"", visible:false,
required:true },
            // Can be visible or not; if visible, can be required or not. Default setting is not visible,
which means required setting is ignored.
            postalCode:       { classes:"", label:"", placeholder:"", message:"", visible:false,
required:true },
            // Can be visible or not; if visible, can be required or not. Default setting is not visible,
which means required setting is ignored.
            i4goInfo:        { classes:"", label:"", visible:true, },
            // By default, set to false to hide the i4Go text and logo. May be set to true to display the
content, which is recommended by Shift4. (For additional information, see Appendix B.)
            cssRules: [
                "body{background-color: ##aaa;font-family:"Trebuchet MS", Arial, Helvetica, sans-serif;}",
                "body{text-size: ""}"
                "label{color:##444;font-size:80%;font-weight:bold;}"
            ],
            // Debug flags: Simply creates additional console log messages
            // A debugger needs to be running to view. In no event does CHD get logged.
            debug:       false, // If true, displays console messages. Parent side
            remoteDebug: false  // If true, indicates width. Adds width indicators within the iframe
contents to help with frame sizing
        */
});
```

Sample Code – `i4goFrame` Div and `challengeIframe` Div

WARNING! The sample code was designed for demonstration purposes only. While you can use this code as a template, additional code will most likely need to be added to make this sample "production ready."

```
<div id="i4goFrame" style="width: 500px;"></div> // specify width to keep content within it

<div id="challengeIframe" class="modal-body" style="height:500px;position:relative"></div> // challenge
will be added as an iframe in the HTML div element
```

Note: The `id` (i.e., `i4goFrame`) can be changed, but it must match the text that is configured in the `frameContainer` parameter.

Implementing Step 4

Step 4: The i4Go exit parameters are returned and mapped to the appropriate vendor-supplied form fields.

Please review these important details:

- The form fields have the default names displayed below:
    - `i4go_response`
    - `i4go_responsecode`
    - `i4go_responsetext`
    - `i4go_maskedcard`
    - `i4go_uniqueid`
    - `i4go_expirationmonth`
    - `i4go_expirationyear`
    - `i4go_cardtype`
    - `i4go_cardholdername`
    - `i4go_streetaddress`
    - `i4go_postalcode`
    - `i4go_extendedcarddata`
    - `i4go_applepaytoken`
    - `i4go_googlepaytoken`
    - `i4go_signeddata`

Note: This is where the i4Go exit parameters are returned so they can be posted to the merchant's Web server. The form field names are a part of a hidden form and displayed in the *Sample Code - i4m Initialization and i4Go Exit Parameters* section. This section of code is only used if the developer would like to change the default names.

Important: Field names, as with most names in JavaScript, are case sensitive.

Requirement: Use the value returned with i4go_uniqueid in the following step. Do not use the i4go_utoken value.

Sample Code – i4Go Exit Parameters

WARNING! The sample code was designed for demonstration purposes only. While you can use this code as a template, additional code will most likely need to be added to make this sample “production ready.”

```
<form action="[merchant web server URL that processes the i4Go response, as per Implementing Step 5]"
method="post" style="display: none;">
    <input id="i4go_response" name="i4go_response" type="hidden" />
    <input id="i4go_responsecode" name="i4go_responsecode" type="hidden" />
    <input id="i4go_responsetext" name="i4go_responsetext" type="hidden" />
    <input id="i4go_cardtype" name="i4go_cardtype" type="hidden" />
    <input id="i4go_maskedcard" name="i4go_maskedcard" type="hidden" />
    <input id="i4go_uniqueid" name="i4go_uniqueid" type="hidden" />
    <input id="i4go_expirationmonth" name="i4go_expirationmonth" type="hidden" />
    <input id="i4go_expirationyear" name="i4go_expirationyear" type="hidden" />
    <input id="i4go_cardholdername" name="i4go_cardholdername" type="hidden" />
    <input id="i4go_streetaddress" name="i4go_streetaddress" type="hidden" />
    <input id="i4go_postalcode" name="i4go_postalcode" type="hidden" />
    <input id="i4go_extendedcarddata" name="i4go_extendedcarddata" type="hidden" />
</form>
```

Implementing Step 5

Step 5: The application uses the payment token (`i4go_uniqueid`) to process the transaction. (Note this step happens outside of i4Go.)

Requirement: If tokenization included 3DS, results will be stored with the `i4go_uniqueid`. If an Apple Pay or Google Pay wallet payment was processed, you must include `i4go_extendedcarddata` and `i4go_uniqueid` in your payment request. Failure to do so may cause the payment transaction to fail.

Note: The process is not over at this point because the payment token still needs to be authorized. This step happens outside of i4Go. For additional information on the authorization process, which uses Shift4's UTG, see RESTful API in MyPortal API Corner.

Accepted i4Go Parameters

This section describes the accepted i4Go entry parameters developers will use for the `preauthorizeClient` request. This section also describes the accepted i4Go exit parameters that will be used to return data to the application.

Note: i4Go is not case sensitive with inbound parameter names. Outbound parameter names will always be in lowercase. For example, `i4go_uniqueid`.

Note: The Uniform Resource Identifier (URI) has a maximum limit of 2048 bytes in length.

i4Go Entry Parameters for the `preauthorizeClient` Request

Parameter	Valid Value	Required?	Description
`fuseaction`	• `account.preauthorizeClient` • Up to 255 bytes in length	**Yes**	Use the `fuseaction=account.preauthorizeClient` parameter to authorize the end user's IP address to submit a transaction through i4Go. When `fuseaction=account.preauthorizeClient` is in use, the `i4go_clientip` and `i4go_accesstoken` parameter must be used in conjunction.
`i4go_clientip`	• Numeric • xxx.xxx.xxx.xxx • Up to 255 bytes in length	**Yes**	Use the `i4go_clientip` parameter to post the end user's public IP address to i4Go. If the end user’s IP address falls in the following ranges, then we substitute the requestor’s IP address for the end user’s IP address because all of these addresses are considered to be internal addresses. • 127.0.0.1/32 • 10.0.0.0/8 • 172.16.0.0/12 • 192.168.0.0/16
`i4go_accesstoken`	• String • Up to 255 bytes in length	**Yes**	Use the `i4go_accesstoken` parameter to post the merchant's Access Token to i4Go.
`i4go_hs256key`	• String • Up to 100 characters in length	No	Use the `i4go_hs256key` parameter to post the merchant-defined value to i4Go, thus returning a signed JWT to ensure the data is not tampered with during the tokenization process.

Parameter	Valid Value	Required?	Description
i4go_version	• 2 or 3	No	Use the i4go_version parameter to specify if the merchant is a US merchant or an international merchant: • 2: Use this for US merchants. The request will be routed to the US Gateway. • 3: Use this for international merchants. The request will be routed to the International Gateway. **Note:** If blank, 2, or a value other than 3 is specified, the request will be routed to the US Gateway. **Note:** This can also be configured by Shift4, negating the use of this parameter. For additional information, contact your Shift4 API analyst.
i4go_basket Begin			
i4go_basket	• String	Yes	Use the i4go_basket container to include support for: • Apple Pay and Google Pay wallets • Card Verify • 3DS • Shift4 Risk Management Services i4go_basket comprises a serialized JSON string containing the following: • OrderDetails • Consumer
OrderDetails Begin			
OrderNumber	• String (50)	**Yes**	This represents your Order Number or transaction identifier.
Amount	• Numeric (20)	**Yes**	Total transaction amount with the decimal.

Parameter	Valid Value	Required?	Description
`CurrencyCode`	• String (3)	**Yes**	Three-digit ISO 4217 value. Accepts either the Currency Number or Currency Code (e.g., "840" or "USD"). **Note:** The currency code will affect the currency symbol displayed when an Apple Pay or a Google Pay wallet is in use.
`OrderDetails` End			
`Consumer Begin`			
`MobilePhone`	• String	Maybe	If you are using Shift4 Risk Management Services to perform a risk assessment, then use the `Consumer` object to send the customer's `MobilePhone`. If you are not using Shift4 Risk Management Services but you are using the Card Verify feature, then you do not need to use this as the information required for the feature's use will be collected via i4Go's use.
`Email1`	• String	Maybe	If you are using Shift4 Risk Management Services to perform a risk assessment, then use the `Consumer` object to send the customer's `Email1` (i.e., their email address). If you are not using Shift4 Risk Management Services but you are using the Card Verify feature, then you do not need to use this as the information required for the feature's use will be collected via i4Go's use.
`Email2`	• String	Maybe	If you are using Shift4 Risk Management Services to perform a risk assessment, then use the `Consumer` object to send the customer's `Email2` (i.e., their alternate email address). If you are not using Shift4 Risk Management Services but you are using the Card Verify feature, then you do not need to use this as the information required for the feature's use will be collected via i4Go's use.

Parameter	Valid Value	Required?	Description
`BillingAddress`	• String	Maybe	If you are using Shift4 Risk Management Services to perform a risk assessment, then use the `BillingAddress` object to send the customer's billing information. If you are not using Shift4 Risk Management Services but you are using the Card Verify feature, then you do not need to use this as the information required for the feature's use will be collected via i4Go's use. `BillingAddress` comprises a serialized JSON string containing the following: • `FirstName` • `LastName` • `Address1` • `Address2` • `City` • `State` • `PostalCode` • `CountryCode` • `Phone1`

Parameter	Valid Value	Required?	Description
ShippingAddress	• String	Maybe	If you are using Shift4 Risk Management Services to perform a risk assessment, then use the ShippingAddress object to send the customer's shipping information. If you are not using Shift4 Risk Management Services but you are using the Card Verify feature, then you do not need to use this as the information required for the feature's use will be collected via i4Go's use. ShippingAddress comprises a serialized JSON string containing the following: • FirstName • LastName • Address1 • Address2 • City • State • PostalCode • CountryCode • Phone1
Consumer End			
i4go_basket End			
paymentAPI Begin			
paymentAPI	• string	No	Use the paymentAPI JSON field to include support for Card Verify and/or 3DS. paymentAPI comprises a serialized JSON string containing the following: • steps The steps field allows for an array of strings so one or both valid values may be included: • CARD_VERIFY • 3DS_STANDALONE

Parameter	Valid Value	Required?	Description
`CARD_VERIFY` Begin			
`transaction.invoice`	• String • Up to 10 characters in length	No	The 10-digit invoice number assigned by the interface to identify a transaction. An invoice number serves as a unique key that identifies a transaction within a batch in Shift4's Gateway.
`transaction.notes`	• String • Up to 4096 characters in length	No	A free-form notes field that supports the use of HTML tags. This can be used for reference in LTM and is not sent to the authorization host. Escaped quotation marks should not be sent in the Notes field.
`transaction.vendorReference`	• String • Up to 50 characters in length	No	Optional field for information that can be searched in the merchant portal.
`transaction.cardOnFile.type`	• String	No	This field specifies the type of the card-on-file transaction. See the REST API documentation for details.
`transaction.cardOnFile.recurringExpiry`	• String • YYYYMMDD	Maybe	Date after which no further authorizations shall be performed. This field is limited to 8 characters, and the accepted format is YYYYMMDD. **Conditional:** This field is required if it's the first recurring transaction (`cardOnFile.type` is `S02`). This field is not needed if the transaction is not recurring or if the transaction is a subsequent recurring transaction. See the REST API documentation for details.
`transaction.cardOnFile.recurringFrequency`	• String	Maybe	**Conditional:** This field is required if it's the first recurring transaction (`cardOnFile.type` is `S02`). This field is not needed if the transaction is not recurring or if the transaction is a subsequent recurring transaction See the REST API documentation for details.
`transaction.cardOnFile.transactionId`	• String • Up to 15 characters	No	This field is returned in the initial `cardOnFile` response, and ties subsequent `cardOnFile` transactions to the original authorization.
`CARD_VERIFY` End			

Parameter	Valid Value	Required?	Description
3DS_STANDALONE Begin			
`threeDSecure.initiate`	• String	**Yes**	Indicates whether to initiate the 3DS authentication process. See the REST API documentation for details.
`threeDSecure.challengeWindowSize`	• String	**Yes**	Dimensions of the challenge window that will be displayed to the cardholder. The issuer replies with content that is formatted to appropriately render in this window to provide the best possible user experience. Preconfigured window sizes are given in "width x height" in pixels. See the REST API documentation for details.
`threeDSecure.transType`	• String	**Yes**	Identifies the type of transaction being authenticated. The values are derived from ISO 8583. See the REST API documentation for details.
`threeDSecure.addressMatch`	• String	**Yes**	Indicates whether the Cardholder Shipping Address and Cardholder Billing Address are identical. See the REST API documentation for details.
`threeDSecure.reqChallengeInd`	• String	**Yes**	Indicates whether a challenge is requested for this transaction. For example, for payment authentication, a merchant may have concerns about the transaction and request a challenge. See the REST API documentation for details.
`transaction.invoice`	• String • Up to 10 characters	**Yes**	The 10-digit invoice number assigned by the interface to identify a transaction. An invoice number serves as a unique key that identifies a transaction within a batch in Shift4's Gateway.
`transaction.notes`	• String • Up to 4096 characters	No	A free-form notes field that supports the use of HTML tags. This can be used for reference in LTM and is not sent to the authorization host. Escaped quotation marks should not be sent in the Notes field.

Parameter	Valid Value	Required?	Description
`transaction.vendorReference`	• String • Up to 50 characters	No	Optional field for information that can be searched in the merchant portal.
`3DS_STANDALONE` End			
`paymentAPI` End			

`OrderDetails` Object Example

```
{
  "OrderDetails": {
    "OrderNumber": "",
    "Amount": "",
    "CurrencyCode": ""
  },

}
```

`paymentAPI` and `3DS_STANDALONE` Example

```
{
    "paymentAPI": {
        "steps": [
            "3DS_STANDALONE"
        ],
        "transaction": {
            "invoice": "0000000001",
            "notes": "Testing 3DS Standalone in i4Go",
            "vendorReference": "TestVendorRef"
        },
        "threeDSecure": {
            "challengeWindowSize": "05",
            "initiate": "01",
            "transType": "01"
        }
    }
}
```

`paymentAPI` and `CARD_VERIFY` Example

```
{
    "paymentAPI": {
        "steps": [
            "CARD_VERIFY"
        ],
        "transaction": {
            "cardOnFile": {
                "type": "S01"
            }
        }
    }
}
```

Accepted i4Go Exit Parameters

The accepted i4Go exit parameters returned in JSON format by i4Go are described and defined in the following table.

Parameter	Valid Value	When Returned?	Description
`i4go_response`	• `SUCCESS` or `FAILURE` • Up to 255 bytes in length	Always	The `i4go_response` parameter is used to return i4Go's Response Message.
`i4go_responsecode`	• Numeric list of one or more numbers • Up to 255 bytes in length	Always	The `i4go_responsecode` parameter is used to return all applicable i4Go Response Codes. For example: • `1` = SUCCESS • `301` = Not authorized For a complete list of Response Codes and Messages, please see *Appendix A*.
`i4go_countrycode`	• String • 2 bytes in length	Always	The `i4go_countrycode` parameter is used to return the two-character country code as assigned by iana.net and other Internet address authorities. For example: • `us` = United States • `??` = Unknown Note: International Organization for Standardization (ISO) Alpha-2 country codes are returned. Shift4 has seen at least two unofficial country codes (for example, `AP`) returned from Internet address authorities, which should be treated as unknown country codes.

Parameter	Valid Value	When Returned?	Description
`i4go_accessblock`	• String • Up to 1024 bytes in length	Successful `preauthorizeClient` Request	The `i4go_accessblock` parameter is used to return i4Go's access block.
`i4go_server`	• String • Up to 128 bytes in length	Successful `preauthorizeClient` Request	The `i4go_server` parameter is used to return the name of the i4Go server.
`i4go_i4m_url`	• String • Up to 128 bytes in length	Successful `preauthorizeClient` Request	The `i4go_i4m_url` parameter is used to return the name of the iFrame URL.

Parameter	Valid Value	When Returned?	Description
`i4go_responsetext`	• String • Up to 255 bytes in length	Tokenization Request	The `i4go_responsetext` parameter is used to return a user friendly description that details why the request failed. If the request was successful, nothing is returned with the parameter.
`i4go_maskedcard`	• Asterisks and numeric • Up to 20 bytes in length	Successful Tokenization Request	The `i4go_maskedcard` parameter is used to return the masked payment card number. In addition, it can be used to display the information to the end user. For example, ********1119.
`i4go_uniqueid`	• String • 16 bytes in length	Successful Tokenization Request	The `i4go_uniqueid` parameter is used to return the payment token (which can be a TrueToken® or a Global Token Vault [GTV] token). The application will use and store the payment token to process the transaction.

Parameter	Valid Value	When Returned?	Description
`i4go_expirationmonth`	• 1 or 2 Numeric Digits • Up to 2 bytes in length	Successful Tokenization Request	The `i4go_expirationmonth` parameter is used to return the expiration month of the payment card. For example, 04 - April is returned as `04`.
`i4go_expirationyear`	• 2 or 4 Numeric Digits • Up to 4 bytes in length	Successful Tokenization Request	The `i4go_expirationyear` parameter is used to return the expiration year of the payment card. For example, 2025 is returned as `2025`.
`i4go_cardtype`	• `VS`, `MC`, `AX`, `DC`, `NS`, `JC`, `YC`, `GC`, and `PL` • Up to 2 bytes in length	Successful Tokenization Request	The `i4go_cardtype` parameter is used to return the two-character code that identifies the payment card type being used. For example: • `VS` - Visa • `MC` - MasterCard • `AX` - American Express • `DC` - Diners Club/Carte Blanche • `NS` - Novus/Discover • `JC` - Japanese Credit Bureau (JCB) • `YC` - IT'S YOUR CARD • `GC` - Gift Card • `PL` - Private Label Payment Card

Parameter	Valid Value	When Returned?	Description
i4go_cardholdername	• String • Up to 255 bytes in length	Successful Tokenization Request	The i4go_cardholdername parameter is used to return the name on the payment card, as entered by the end user, to i4Go. Or, when Apple Pay/Google Pay is used as the payment method on the transaction, then the billing/shipping name from the associated wallet will be returned.
i4go_postalcode	• String • Up to 20 bytes in length	Successful Tokenization Request	The i4go_postalcode parameter is used to return the postal/ZIP code from the address that corresponds to the payment card, as entered by the end user, to i4Go.
i4go_streetaddress	• String • Up to 50 bytes in length	Successful Tokenization Request	The i4go_streetaddress parameter is used to return the numerical portion of the street address that corresponds to the payment card, as entered by the end user, to i4Go.

Parameter	Valid Value	When Returned?	Description
`i4go_extendedcarddata`	• String • Up to 8192 bytes in length	Successful Tokenization Request as Required	The `i4go_extendedcarddata` parameter is used to return extended card authentication data that corresponds to Apple Pay and Google Pay wallets support.
`i4go_applepaytoken`	• String • Up to 4096 bytes in length‡	Successful Tokenization Request and Apple Pay Was Used	This is optional and is a serialized JSON string representing the raw Apple Pay token that i4Go received. This token contains additional payment or cardholder information your application can leverage. This information is only returned if Apple Pay was used. For additional information, see Apple Pay documentation at: https://developer.apple.com/documentation/apple_pay_on_the_web/applepaypaymentmethodselectedevent/1778025-paymentmethod.
`i4go_googlepaytoken`	• String • Up to 4096 bytes in length‡	Successful Tokenization Request and Any Wallet Was Used	This is optional and is a serialized JSON string representing the raw Google Pay token (or a mimicked representation from an Apple Pay token) that i4Go received. This information is only returned if Google Pay or Apple Pay was used. For additional information, see Google Pay documentation at: https://developers.google.com/pay/api/web/reference/response-objects.

Parameter	Valid Value	When Returned?	Description
`i4go_signeddata`	• JWT	Successful Tokenization Request as Required	When `i4go_hs256key` is used, the `i4go_signeddata` parameter is used to return the corresponding JWT. (The JWT's signature can then be verified before attempting to submit the transaction. For additional information on JWT, see https://jwt.io/.)

‡A maximum length of 4096 bytes should be safe; however, this is not controlled by Shift4.

`risk` Object Details

If you are using Shift4 Risk Management Services to perform a risk assessment during the i4Go tokenization process, then your response will also include the `risk` object.

Parameter	Valid Value	When Returned?	Description
`dateTime`	• String	Successful Tokenization Request	The `dateTime` parameter is used to return the information in ISO 8601 format including the timezone offset (yyyy-mm-ddThh:mm:ss.nnn+hh:mm) of when the risk assessment was submitted.
`amount`	See Description.	Successful Tokenization Request	The `amount` object is used to return the following: • `total` • `tax` The `total` is the total amount submitted for the risk assessment, including `tax`. The valid values are: • Numeric • Up to 14 bytes in length
`server`	See Description.	Successful Tokenization Request	The `server` object is used to return the following: • `name` The `name` is the risk assessment's server name. The valid value for it is: • String
`correlationId`	• String • Up to 36 bytes in length	Successful Tokenization Request	The `correlationId` parameter is used to return the Correlation ID, which is a value used to correlate transactions together for reporting purposes.

Parameter	Valid Value	When Returned?	Description
risk	See Description.	Successful Tokenization Request	The risk object contains the following: • tranId • assessment The tranId parameter is used to return the unique transaction ID for this response. Store this value and use it to find additional details. The assessment parameter is used to return the result of the risk assessment, and it will include one of the following: • **A:** Approve • **D:** Deny| • **R:** Review • **E:** Escalate| The valid value for them is: • String |If D or E is retuned as the result, it will cause a 301 (Not authorized) to be returned in the i4go_responsecode parameter.
transaction	See Description.	Successful Tokenization Request	The transaction object contains the following: • s4RiskId • invoice The s4RiskId parameter is used to return the unique transaction identification number generated by Shift4 to identify a specific risk transaction. The valid value for it is: • String The invoice parameter is used to return the invoice number assigned by the interface to identify a transaction. The valid values for it are: • String • Up to 10 bytes in length
currencyCode	• String • 3 characters	Successful Tokenization Request	The currencyCode parameter is used to return the three-digit ISO 4217 value, either the Currency Number or Currency Code (e.g., 840 or USD).

Parameter	Valid Value	When Returned?	Description
`card`	See Description.	Successful Tokenization Request	The `card` object contains the `token` object, which contains the following: • `value` The `value` parameter is used to return the payment token. The valid values for it are: • String • 16 bytes in length

***`risk`* Object Example**

```
"risk": {
        "dateTime": "2023-04-28T07:42:01.685868-08:00",
        "amount": {
            "total": 50,
            "tax": 0
        },
        "server": {
            "name": "KNDEV1"
        },
        "correlationId": "EAD9584E-D689-420D-88F4-955706DA2F33",
        "risk": {
            "tranId": "KHVT0CRXVJRR",
            "assessment": "A"
        },
        "transaction": {
            "s4RiskId": "D31698F0-B6A6-43C4-A408-EEA0FD282BF5",
            "invoice": "Z100009020"
        },
        "currencyCode": "USD",
        "card": {
            "token": {
                "value": 8053679897173332
            }
```

Example of i4Go in an iFrame

To render an example of i4Go in an iFrame, click here and then click on the available templates, options, and languages.

Note: The example displays what is possible. Based on defaults or changes you make to the code, some fields may or may not be displayed. In addition, you can change the width using `i4goFrame`.

WARNING! The Bootstrap 5 template is not supported with Internet Explorer 11 or earlier.

Implementing Apple Pay and Google Pay Wallets

Wallets provide cardholders a friendly, consistent, and simple user experience.

Before implementing wallets to an i4Go integrated application, we STRONGLY recommend you start with a working i4Go integration first. Adding wallet support to an existing i4Go integration is very easy and the changes are detailed here.

Apple Pay

What is Apple Pay?

Apple Pay is easy and works with the Apple devices you use every day. You can make contactless, secure purchases in stores, in apps, and on the web. And you can send and receive money from friends and family right in Messages. Apple Pay is a safer way to pay, and even simpler than using your physical card.

Apple Pay Prerequisites

Before adding Apple Pay to your application, you must read, understand, follow, and accept the Apple Pay on the Web: Acceptable Use Guidelines. These guidelines are published and mandated by Apple and are outside the control of Shift4.

Apple Pay does require proof of domain ownership prior to assigning credentials. This process is detailed in the *Step 1 – Account Setup* section below.

For more information, see Apple Pay on the Web Overview.

Google Pay

What is Google Pay?

Google Pay brings together all the ways you can pay with Google.

Enter your card information once and use it to:

- Tap and pay to make purchases with your phone (see country and device availability).
- Buy items in apps and on websites (see country availability).
- Fill in forms automatically on Chrome.
- Buy Google products.
- Send money to friends and family (US only).

You can also use your gift cards, loyalty cards, tickets, and coupons with Google Pay when you shop at your favorite stores.

Google Pay Prerequisites

Before adding Google Pay to your application, you must review and adhere to the following Google Pay API Terms of Service, Acceptable Use Policy, and Brand Guidelines. These guidelines are published and mandated by Google and are outside the control of Shift4.

In addition, the following documents can be used for reference: Google Pay Web Developer Documentation, Google Pay Web Integration Checklist, and Google Pay for Payments Overview.

Implementing Wallets

Now that you have completed your i4Go integration for tokenizing cardholder data, it is time to add wallet support. The wallets, Apple Pay and Google Pay, will allow for a much quicker and streamlined user experience, along with adding an additional security layer for your customers.

This section assumes you have an understanding of the i4Go API and a working i4Go integration. Only details for implementing wallet support to your interface are included below.

Step 1 – Account Setup

Before requesting Apple Pay and/or Google Pay setup and credentials, go through and thoroughly read and understand the prerequisites detailed previously.

Shift4 is in the process of making the setup process and the request for development/testing and production credentials self-serve, but that is under development. Until this is available, you will need to speak directly to your Shift4 representative for account setup and credentials. Please check back for further updates as this is a work in progress.

Apple Pay does require proof of domain ownership prior to assigning credentials. Before Apple Pay credentials can be assigned, you will be provided a file from Shift4 and instructions on where to put them on your server (located here). The file is for proof of domain ownership and does not contain any sensitive information. The file should remain on the server(s) even after ownership confirmation.

Step 2 – CSS and JavaScript – Adding Additional Includes

i4Go wallet support required the addition of two Shift4 include files: a style sheet (CSS) and a JavaScript (JS) file. They can be downloaded directly to the client from:

- https://i4m.i4go.com/css/wallets.css
- https://i4m.i4go.com/js/wallets.js

In addition, for Google Pay support, a Google JavaScript (JS) file is required:

- https://pay.google.com/gp/p/js/pay.js

Sample Code

WARNING! The sample code was designed for demonstration purposes only. While you can use this code as a template, additional code will most likely need to be added to make this sample "production ready."

```
<link rel="stylesheet" type="text/css" href="https://i4m.i4go.com/css/wallets.css">
<script src="https://i4m.i4go.com/js/wallets.js" type="text/javascript"></script>
<script src="https://pay.google.com/gp/p/js/pay.js" type="text/javascript"></script>
```

Step 3 - i4GoTrueToken Initiation

When including wallet support, we have added the following:

- `onWalletInit: function( wallet:string, enabled:boolean, reason:string ):` This is optional. Wallet will asynchronously trigger individual callbacks.
- `i4go_extendedcarddata`: **This is required.** You must receive the extended card data and pass it to your Shift4 payment request. Failure to pass the information will result in authorization failures and settlement downgrades (higher rates). The result is supplied into the `extendedcarddata` field of the Shift4 API.
- `i4go_applepaytoken`: This is optional and is a serialized JSON string representing the raw Apple Pay token that i4Go received. This token contains additional payment or cardholder information your application can leverage. This information is only returned if Apple Pay was used. For additional information, see Apple Pay documentation at: https://developer.apple.com/documentation/apple_pay_on_the_web/applepaypaymentmethodselectedevent/1778025-paymentmethod.
- `i4go_googlepaytoken`: This is optional and is a serialized JSON string representing the raw Google Pay token (or a mimicked representation from an Apple Pay token) that i4Go received. This information is only returned if Google Pay or Apple Pay was used. For additional information, see Google Pay documentation at: https://developers.google.com/pay/api/web/reference/response-objects.

Sample Code

WARNING! The sample code was designed for demonstration purposes only. While you can use this code as a template, additional code will most likely need to be added to make this sample "production ready."

```
…
// Wallet event
onWalletInit: function(wallet:string, enabled:boolean, reason:string ) {
        console.log("i4goTrueToken- onWalletInit()",wallet);
},
// Auto populated form fields. Precedence: field name, field id
formPaymentResponse:          "customNameFori4go_response",
formPaymentResponseCode:      "customNameFori4go_responsecode",
formPaymentResponseText:      "customNameFori4go_responsetext",
formPaymentMaskedCard:        "customNameFori4go_maskedcard",
formPaymentToken:             "customNameFori4go_uniqueid",
formPaymentExpMonth:          "customNameFori4go_expirationmonth",
formPaymentExpYear:           "customNameFori4go_expirationyear",
formPaymentType:              "customNameFori4go_cardtype",
formCardholderName:           "customNameFori4go_cardholdername",
formStreetAddress:            "customNameFori4go_streetaddress",
formPostalCode:               "customNameFori4go_postalcode",
formExtendedCardData:         "customNameFori4go_extendedcarddata",
formApplePayToken:            "customNameFori4go_applepaytoken",
formGooglePayToken:           "customNameFori4go_googlepaytoken",
…
```

Step 4 – HTML – Adding Buttons

We're to the easiest part. Simply include one or both buttons in your HTML code. The buttons are hidden by default and will only reveal themselves if the associated wallet is available. Feel free to add your own style to the buttons, just be sure you refer to the branding guidelines (see the *Apple Pay Prerequisites* or *Google Pay Prerequisites* section).

Sample Code

WARNING! The sample code was designed for demonstration purposes only. While you can use this code as a template, additional code will most likely need to be added to make this sample "production ready."

```
<button class="pay-button pay-hidden apple-pay-button"></button>
<button class="pay-button pay-hidden google-pay-button"></button>
```

Step 5 – Test

Test your integration to ensure it works.

Step 6 – Advanced Features

For most of the advanced wallet features, i4Go uses Google Pay as the standard. Meaning, if you code for Google Pay (with very few or no exceptions) you will receive Apple Pay compatibility.

We chose this path because in our view, Google Pay seems to have more developer friendly documentation. Also, developers have a wider selection of developer and debugging tools whereas Apple Pay is limited to tools running in iOS only.

Google Pay Button Styles

Google Pay controls the button styles via JavaScript settings, which are passed through our i4goTrueToken settings.

```
wallet:   {

          // For the following button attributes, see Google documentation. Default settings used when
left empty.

          buttonColor:          "",       // default: A Google-selected default value. Currently black
but it may change over time (default).

                                          // black: A black button suitable for use on white or light
backgrounds.

                                          // white: A white button suitable for use on colorful
backgrounds.

          buttonType:           "",       // buy: "Buy with Google Pay" button (default).

                                          // donate: "Donate with Google Pay" button.

                                          // plain: Google Pay button without additional text.

                                          // translated button label may appear if a language specified
in the viewer's browser matches an available language.

          buttonSizeMode:       "",       // static: Button has a static width and height (default).

                                          // fill: Button size changes to fill the size of its container.

          buttonRootNode:       ""        // HTMLDocument or ShadowRoot

}
```

For additional information, see Google Pay documentation at: https://developers.google.com/pay/api/web/reference/request-objects#ButtonOptions.

Apple Pay Button Styles

Apple Pay controls the button styles via CSS classes, which you add directly to the button tag. For additional information, see Apple Pay documentation at: https://developer.apple.com/documentation/apple_pay_on_the_web/styling_the_apple_pay_button_with_css.

Name, Address, and Static Shipping Options

The wallets can optionally return information from the cardholder, including: name, email address, phone number, and shipping address. Any or all of these fields can be required. Due to various local, state, and federal personally identifiable information (PII) laws, it is recommended that you only require what you need to fulfill your order. In other words, if you are not shipping goods, do not require shipping information as this would simply increase your liability scope for PII.

```
wallet:  {
         nameRequired:            null,            // true/false/null - null uses addressRequired
         addressRequired:         false,
         emailRequired:           false,
         phoneNumberRequired:     false,
         allowedCountryCodes:     [],
         shippingOptionRequired:  false,
         shippingOptions:         [],              // first option will be the default
         shippingType:            "shipping"       // shipping, delivery, storePickup, servicePickup
}
```

Dynamic Shipping and Sales Tax

Dynamic shipping and sales tax support is an extension of the prior *Name, Address, and Static Shipping Options* section. The difference between static and dynamic is that dynamic requires an additional event to be defined in order to support this feature. The feature allows for shipping options to change based on different shipping addresses being selected and provided by the cardholder within the wallet.

For example, if the cardholder selects to use their home address, one set of shipping options is made available. If the cardholder changes the shipping address to a work address or a hotel address, a different set of shipping options are displayed. In both these cases, the appropriate sales tax is applied to the order.

Tip: Before adding dynamic shipping and sales tax options, get the prior *Name, Address, and Static Shipping Options* section working first.

```
wallet:                         {
        nameRequired:           null,           // true/false/null - null uses addressRequired.
        addressRequired:        false,
        emailRequired:          false,
        phoneNumberRequired:    false,
        allowedCountryCodes:    [],
        shippingOptionRequired: false,
        shippingOptions:        [],             // first option will be the default.
        shippingType:           "shipping"      // shipping, delivery, storePickup, servicePickup
}
```

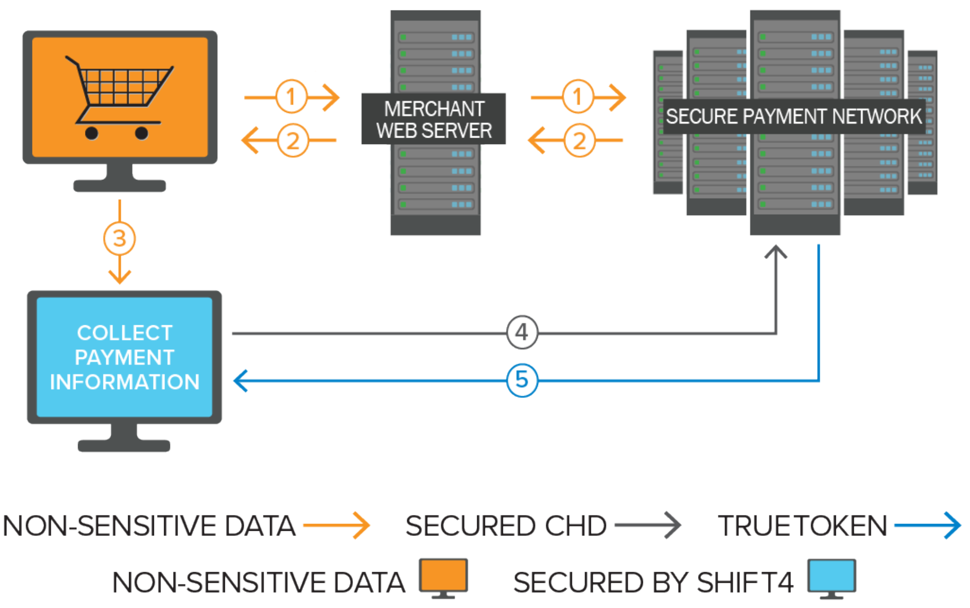
AJAX Using JSON and Standard Direct Post

Shift4 recommends a combination of both technologies be implemented: AJAX using JSON for a seamless integration with better communication error trapping than Standard Direct Post, and Standard Direct Post as a fallback in the event that scripting is disabled on the client browser.

AJAX Using JSON

To implement i4Go using this method, there are five key steps; each step requires developers to implement certain functionality to ensure the payment information is tokenized. The five steps are briefly outlined below and described in greater detail in the following subsections.

Requirement: The payment information must be tokenized for each transaction in order to benefit from reduced PCI DSS scope.



Note: Step 1 is initiated from the merchant's Web server.

1. A purchase is initiated at the point of sale in an internet browser-based environment. The end user's IP address is sent with the merchant's Access Token through the merchant's server to the i4Go server, thus requesting authorization† for Step 4.
   - This step requires the use of `fuseaction=account.authorizeClient`, `i4go_clientip`, and `i4go_accesstoken` posted to https://access.shift4test.com (for certification) or https://access.i4go.com (for production). (The response will be JSON. For additional information, see the *i4Go Entry Parameters for the `authorizeClient` Request* section.)

Requirement: Any call to https://access.shift4test.com or https://access.i4go.com must be direct posted and cannot be JSON or XML. In addition, developers must ensure the application retains a log of all authorization requests, including the client IP address, for troubleshooting purposes.

2. The i4Go server returns an access block and the i4Go server address to the merchant's server. The merchant's server must modify the payment information form to include the access block and to post to the returned i4Go server address.
   - For a successful `authorizeClient` request, this step returns `i4go_response`, `i4go_responsecode`, `i4go_countrycode`, `i4go_accessblock`, and `i4go_server`.
   - For a failed request, this step returns `i4go_response` and `i4go_responsecode`.
3. The CHD is entered on the payment information form.

Note: Step 4 is initiated from the end user’s browser session.

4. Over an encrypted connection, the CHD and access block are directly submitted from the end user’s browser to the i4Go server address that is returned with the `i4go_server` parameter. i4Go sends the encrypted CHD to Shift4’s PCI DSS-compliant LTM gateway where it is replaced with a payment token.
   - This step requires the use of the parameters in the *i4Go Entry Parameters for the Tokenization Request* table and `i4go_accessblock`.
5. LTM returns the payment token to the i4Go server. The i4Go server returns the payment token to the browser.
   - For a successful tokenization request, this step returns `i4go_response`, `i4go_responsecode`, `i4go_cardtype`, `i4go_uniqueid`, `i4go_expirationmonth`, and `i4go_expirationyear`.
   - For a failed request, this step returns `i4go_response` and `i4go_responsecode`.

Requirement: If a timeout or null response is received, the process to tokenize the payment information must begin again at step 1.

Note: It is important to note the process is not over at this point because the payment token still needs to be authorized. For additional information on the authorization process, which uses Shift4's UTG, see RESTful API in MyPortal API Corner.

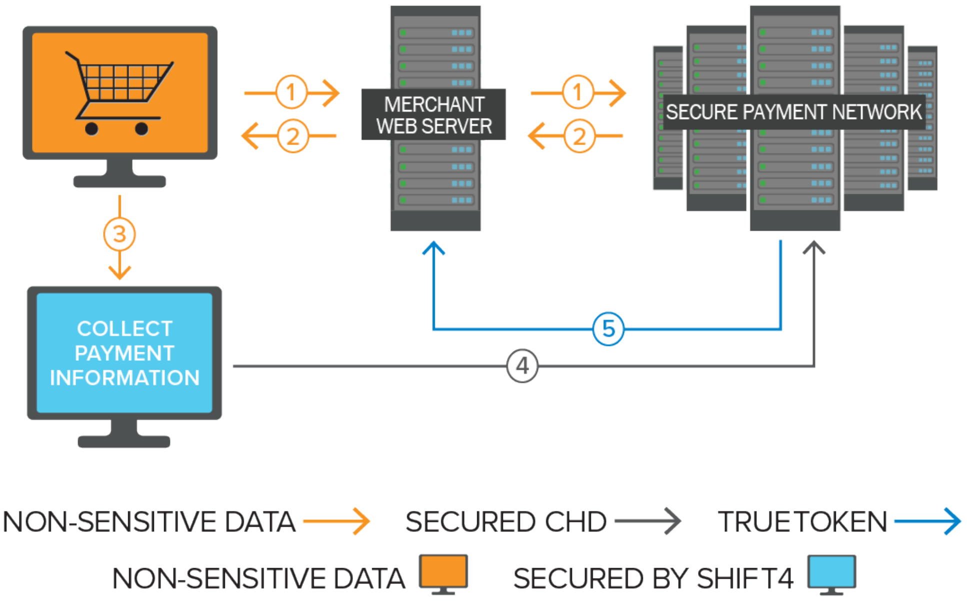
†This will attempt to authorize the end user's IP address to submit a single transaction through i4Go, regardless of where in the world the end user resides.

Standard Direct Post

To implement i4Go using this method, there are five key steps; each step requires developers to implement certain functionality to ensure the payment information is tokenized. The five steps are briefly outlined below and described in greater detail in the following subsections.

Tip: If JavaScript is enabled, use the AJAX using JSON implementation method, as it provides better error handling.

Requirement: The payment information must be tokenized for each transaction in order to benefit from reduced PCI DSS scope.



Note: Step 1 is initiated from the merchant's Web server.

1. A purchase is initiated at the point of sale in an internet browser-based environment. The end user's IP address is sent with the merchant's Access Token through the merchant's server to the i4Go server, thus requesting authorization† for Step 4.
    - This step requires the use of `fuseaction=account.authorizeClient`, `i4go_clientip`, and `i4go_accesstoken` posted to https://access.shift4test.com (for certification) or https://access.i4go.com (for production). (The response will be JSON. For additional information, see the *i4Go Entry Parameters for the* `authorizeClient` *Request* section.)

Requirement: Any call to https://access.shift4test.com or https://access.i4go.com must be direct posted and cannot be JSON or XML. In addition, developers must ensure the application retains a log of all authorization requests, including the client IP address, for troubleshooting purposes.

2. The i4Go server returns an access block and the i4Go server address to the merchant's server. The merchant's server must modify the payment information form to include the access block and to post to the returned i4Go server address.
    - For a successful `authorizeClient` request, this step returns `i4go_response`, `i4go_responsecode`, `i4go_countrycode`, `i4go_accessblock`, and `i4go_server`.
    - For a failed request, this step returns `i4go_response` and `i4go_responsecode`.
3. The CHD is entered on the payment information form.

Note: Step 4 is initiated from the end user’s browser session.

4. Over an encrypted connection, the CHD and access block are directly submitted from the end user’s browser to the i4Go server address that is returned with the `i4go_server` parameter. i4Go sends the encrypted CHD to Shift4’s PCI DSS-compliant LTM gateway where it is replaced with a payment token.
    - This step requires the use of the parameters in the *i4Go Entry Parameters for the Tokenization Request* table and `i4go_accessblock`.
5. LTM returns the payment token to the i4Go server. The i4Go server returns the payment token to the merchant's server.
    - For a successful tokenization request, this step returns `i4go_response`, `i4go_responsecode`, `i4go_cardtype`, `i4go_uniqueid`, `i4go_expirationmonth`, and `i4go_expirationyear`.
    - For a failed request, this step returns `i4go_response` and `i4go_responsecode`.

Note: It is important to note the process is not over at this point because the payment token still needs to be authorized. For additional information on the authorization process, which uses Shift4's UTG, see RESTful API in MyPortal API Corner.

†This will attempt to authorize the end user's IP address to submit a single transaction through i4Go, regardless of where in the world the end user resides.

Accepted i4Go Parameters

This section describes the accepted i4Go Entry and Exit Parameters developers will use to post the payment information to i4Go. This section also describes the accepted parameters i4Go will use to return data to the application.

Note: i4Go is not case sensitive with inbound parameter names. Outbound parameter names will always be in lowercase. For example, `i4go_uniqueid`.

Note: The Uniform Resource Identifier (URI) has a maximum limit of 2048 bytes in length.

Accepted i4Go Entry Parameters

The accepted i4Go Entry Parameters that can be posted to i4Go, enabling processing, are described and defined in the tables below.

i4Go Entry Parameters for the `authorizeClient` Request

Parameter	Valid Value	Required?	Description
`fuseaction`	• `account.authorizeClient` • Up to 255 bytes in length	**Yes**	Use the `fuseaction=account.authorizeClient` parameter to authorize the end user's IP address to submit a transaction through i4Go. When `fuseaction=account.authorizeClient` is in use, the `i4go_clientip` and `i4go_accesstoken` parameter must be used in conjunction.
`i4go_clientip`	• Numeric • xxx.xxx.xxx.xxx • Up to 255 bytes in length	**Yes**	Use the `i4go_clientip` parameter to post the end user's public IP address to i4Go. If the end user’s IP address falls in the following ranges, then we substitute the requestor’s IP address for the end user’s IP address because all of these addresses are considered to be internal addresses. • 127.0.0.1/32 • 10.0.0.0/8 • 172.16.0.0/12 • 192.168.0.0/16
`i4go_accesstoken`	• String • Up to 255 bytes in length	**Yes**	Use the `i4go_accesstoken` parameter to post the merchant's Access Token to i4Go.

i4Go Entry Parameters for the Tokenization Request

Parameter	Valid Value	Required?	Description
`fuseaction`	• `api.jsonPostCardEntry` OR • `api.xmlPostCardEntry` OR • `form.cardEntry` AND • Up to 255 bytes in length	**Yes**	Use the `fuseaction=api.jsonPostCardEntry` parameter if you do not need to support browsers prior to Internet Explorer 8 or if you are writing your own AJAX using JSON interface. This is the preferred method. The response will be returned to the end user's browser in JSON format. Use this parameter with the AJAX using JSON implementation method. OR Use the `fuseaction= api.xmlPostCardEntry` parameter, if preferred. OR Use the `fuseaction=form.cardEntry` parameter if JavaScript MUST be disabled. (It is important to note that better error handling can be achieved when JavaScript is enabled, so this option should only be used as a fallback.) The response will be returned to the `i4go_successurl` and `i4go_failureurl` parameters. Use this parameter with the Standard Direct Post implementation method.
`i4go_accessblock`	• String • Up to 1024 bytes in length	**Yes**	Use the `i4go_accessblock` parameter to post the received access block to i4Go. The application will need to modify the payment information form to include the access block (which includes the merchant's Access Token).

Parameter	Valid Value	Required?	Description
`i4go_successurl`	• String • Up to 255 bytes in length	No	If the `fuseaction= form.cardEntry` parameter is in use, use the `i4go_successurl` to post the return URL for a successful request to i4Go.
`i4go_failureurl`	• String • Up to 255 bytes in length	No	If the `fuseaction= form.cardEntry` parameter is in use, use the `i4go_failureurl` to post the return URL for a failed request to i4Go.
`i4go_p2pedevicetype`	• Numeric • `01`	No	If point-to-point encryption (P2PE) is in use, use the `i4go_p2pedevicetype` parameter to post P2PE device type `01` to i4Go. If the `i4go_p2pedevicetype` parameter is in use, the `i4go_p2peblock` parameter must be used in conjunction.
`i4go_p2peblock`	• String • Up to 255 bytes in length	No	If point-to-point encryption (P2PE) is in use, use the `i4go_p2peblock` parameter to post the swiped payment card data to i4Go. If the `i4go_p2peblock` parameter is in use, the `i4go_p2pedevicetype` parameter must be used in conjunction and the following parameters are not necessary: `i4go_cardnumber`, `i4go_expirationmonth`, and `i4go_expirationyear`. Please refer to the *Track Information Format* section for the format in which the payment card's track information should be posted to i4Go.

Parameter	Valid Value	Required?	Description
`i4go_trackinformation$`	• String • Up to 255 bytes in length	No	Use the `i4go_trackinformation` parameter to post the track information of the payment card, as swiped by the end user, to i4Go. If the `i4go_trackinformation` parameter is in use, the `i4go_cardnumber`, `i4go_expirationmonth`, and `i4go_expirationyear` parameter must be used in conjunction. Please refer to the *Track Information Format* section for the format in which the payment card's track information should be posted to i4Go.
`i4go_cardnumber$`	• Numeric • Up to 24 bytes in length	**Yes**	Use the `i4go_cardnumber` parameter to post the payment card number, as entered by the end user, to i4Go. Note: The length allowed to be entered in the iFrame is determined by the card brand detected. For example, if a Visa is detected, the maximum length is 19. If a gift card is entered, the maximum length is 24.
`i4go_expirationmonth`	• 1 or 2 Numeric Digits • Up to 2 bytes in length	**Yes**	Use the `i4go_expirationmonth` parameter to post the expiration month of the payment card, as entered by the end user, to i4Go. Choose between the following formats; for example, April would be `4` or `04`.
`i4go_expirationyear`	• 2 or 4 Numeric Digits • Up to 4 bytes in length	**Yes**	Use the `i4go_expirationyear` parameter to post the expiration year of the payment card, as entered by the end user, to i4Go. Choose between the following formats; for example, the year would be 25 or `2025`.
`i4go_cvv2code$`	• String • Up to 4 bytes in length	**Yes**	Use the `i4go_cvv2code` parameter to post the card security code (CVV2) found on the front or back of the payment card, as entered by the end user, to i4Go.

Parameter	Valid Value	Required?	Description
`i4go_cvv2indicator`	• `0`, `1`, and `2` • Up to 255 bytes in length	No	Use the `i4go_cvv2indicator` parameter to post the reason the card security code (CVV2) was not provided, as entered by the end user, to i4Go. For example: • `0` - Not Present • `1` - Present • `2` - Unreadable
`i4go_cardholdername`	• String • Up to 255 bytes in length	No	Use the `i4go_cardholdername` parameter to post the name on the payment card, as entered by the end user, to i4Go.
`i4go_cardtype`§	• `AX`, `DC`, `JC`, `MC`, `NS`, `VS`, `YC`, and `GC` • Up to 2 bytes in length	No	Use the `i4go_cardtype` parameter to post the two-character code that identifies the payment card type being used, as entered by the end user, to i4Go. For example: • `AX` - American Express • `DC` - Diners Club/Carte Blanche • `JC` - Japanese Credit Bureau • `MC` - MasterCard • `NS` - Novus/Discover • `VS` - Visa • `YC` - IT'S YOUR CARD® • `GC` - Non-standard gift card
`i4go_postalcode`	• String • Up to 20 bytes in length	No	Use the `i4go_postalcode` parameter to post the postal/ZIP code from the address that corresponds to the payment card, as entered by the end user, to i4Go.
`i4go_streetaddress`	• String • Up to 50 bytes in length	No	Use the `i4go_streetaddress` parameter to post the numerical portion of the street address that corresponds to the payment card, as entered by the end user, to i4Go.

§i4Go is designed to keep real and sensitive CHD information out of the merchant's Web server or hosting provider's system. If you send this information to the merchant's systems, you are defeating i4Go's purpose.

Non-i4Go Entry Parameters

Note: This section does not apply to the AJAX using JSON implementation method.

In addition to sending the accepted i4Go Entry Parameters, developers can post the application's user-defined entry parameters to i4Go. These entry parameters are non-i4Go entry parameters.

Non-i4Go entry parameters are not stored in Shift4's database and therefore can contain any information that is meaningful to the merchant. The non-i4Go entry parameter and its associated value are returned as an extra parameter with the Success URL or Failure URL.

For example, if the following entry parameter is sent:

- `ExampleParameter=value`

i4Go will return "EXAMPLEPARAMETER=VALUE" with the Success URL or Failure URL.

Please note the prefix `i4go_` is reserved for accepted i4Go Entry Parameters. This prefix cannot be used to send non-i4Go entry parameters; however, there are no other restrictions on non-i4Go entry parameter names.

In addition, i4Go places no limitation on the number of characters in non-i4Go entry parameters. As in this case, the Direct Post method, the parameters are not sent as part of the URL; therefore, maximum URL lengths for browsers supported by the merchant do not apply.

Accepted i4Go Exit Parameters

When the Standard Direct Post method is in use, the i4Go Exit Parameters are returned as URL query string parameters. When the AJAX using JSON method is in use, the parameters are returned in a parsed response.

The accepted i4Go Exit Parameters returned by i4Go are described and defined in the following table.

Parameter	Valid Value	When Returned?	Description
`i4go_response`	• `SUCCESS` and `FAILURE` • Up to 255 bytes in length	Always	Use the `i4go_response` parameter to receive i4Go's Response Message.
`i4go_responsecode`	• Numeric • Up to 255 bytes in length	Always	Use the `i4go_responsecode` parameter to receive i4Go's Response Code. For example: • `1` = SUCCESS For a complete list of Response Codes and Messages, please see *Appendix A*.
`i4go_countrycode`	• String • 2 bytes in length	Always	Use the `i4go_countrycode` parameter to receive the two character country code as assigned by iana.net and other internet address authorities. For example: • `us` = United States • `??` = Unknown Note: International Organization for Standardization (ISO) Alpha-2 country codes are returned. Shift4 has seen at least two unofficial country codes (for example, `AP`) returned from internet address authorities, which should be treated as unknown country codes.
`i4go_accessblock`	• String • Up to 1024 bytes in length	Successful `authorizeClient` Request	Use the `i4go_accessblock` parameter to receive i4Go's access block. The application will modify the payment information form to include the access block (which includes the merchant's Access Token).

Parameter	Valid Value	When Returned?	Description
`i4go_server`	• String • Up to 128 bytes in length	Successful `authorizeClient` Request	Use the `i4go_server` parameter to receive the name of the i4Go server. The application will use the result as the server name in the form action for the payment information form.
`i4go_cardtype§`	• `AX`, `DC`, `JC`, `MC`, `NS`, `VS`, `YC`, and `GC` • Up to 2 bytes in length	Successful Tokenization Request	Use the `i4go_cardtype` parameter to receive i4Go's return of the two-character code that identifies the payment card type being used. For example: • `AX` - American Express • `DC` - Diners Club/Carte Blanche • `JC` - Japanese Credit Bureau • `MC` - MasterCard • `NS` - Novus/Discover • `VS` - Visa • `YC` - IT'S YOUR CARD • `GC` - Non-standard gift card
`i4go_uniqueid`	• String • 16 bytes in length	Successful Tokenization Request	Use the `i4go_uniqueid` parameter to receive the payment token (which can be a TrueToken or a GTV). The application will use and store the payment token to process the transaction.
`i4go_expirationmonth`	• 1 or 2 Numeric Digits • Up to 2 bytes in length	Successful Tokenization Request	Use the `i4go_expirationmonth` parameter to receive i4Go's return of the expiration month of the payment card. For example, based on your format, April would be `4` or `04`.
`i4go_expirationyear`	• 2 or 4 Numeric Digits • Up to 4 bytes in length	Successful Tokenization Request	Use the `i4go_expirationyear` parameter to receive i4Go's return of the expiration year of the payment card. For example, based on your format, the year would be 25 or `2025`.

§i4Go is designed to keep real and sensitive CHD information out of the merchant's Web server or hosting provider's system. If you send this information to the merchant's systems, you are defeating i4Go's purpose.

Non-i4Go Exit Parameters

Note: This section does not apply to the AJAX using JSON implementation method.

When the Standard Direct Post method is in use, in addition to the i4Go Exit Parameters, if you send a non-i4Go entry parameter, i4Go will send the same parameter/value back as an exit parameter. Refer to the *Non-i4Go Entry Parameters* section for more information.

Note: i4Go does not process the non-i4Go entry parameters. i4Go simply sends the parameters back to the application so the application can process the non-i4Go parameters as needed.

Track Information Format

This section contains a table that defines the format in which the payment card's track information should be posted to i4Go.

How was cardholder data entered?	Track Information Format	Example
Track 1 Swipe	*Track 1 Format:* • %B • Payment card number • ^ • Cardholder name • ^ • Expiration date (YYMM) • ?	*%B4321000000001119^JOHN SMITH^1812999999?*
Track 2 Swipe	*Track 2 Format:* • ; • Payment card number • = • Expiration date (YYMM) • ?	*;4321000000001119=1812999999?*

How was cardholder data entered?	Track Information Format	Example
Tracks 1 and 2 Swipe	*Track 1 Format:* • %B • Payment card number • ^ • Cardholder name • ^ • Expiration date (YYMM) • ? **-and-** *Track 2 Format:* • ; • Payment card number • = • Expiration date (YYMM) • ?	*%B4321000000001119^JOHN SMITH^1812999999?;4321000000001119=1812999999?*
P2PE Device Type 01 – Track 1 Swipe	*Track 1 Format:* • See example	*02D800800F2E00001189%*333700******5551^JOHN SMITH^1612******?*EACE6DBFCBAE3AB344465B09FC14915FB092A7DC035BE0D5BE6ECD5F39584F17C741536B7F71B163438F4041CD1242EC3B7A5C5020CB7ECED404E10360E44B4BEDC38F5D629949500100000091274F803*

How was cardholder data entered?	Track Information Format	Example
P2PE Device Type 01 – Track 2 Swipe	*Track 2 Format:* • See example	*02A8008017001E001292;543200******3332=1612******?* 88998682C7FBD67A6B60D8ADC9DEEABA92187C16CF4 CD24F4B4526CE88B1AAA84CEED6FDBFB2B51CB53E2 62FDC9308FB5E56CFA629949500100000009111ED203*
P2PE Device Type 01– Track 1 and 2 Swipe	*Track 1 Format:* • See example **-and-** *Track 2 Format:* • See example	*028301801F331E00139B%*333700******5551^JOHN SMITH^1612******?*;333700******5551=1612******?*4545 6A4FA6E645D0C886D56EA623C1BA55F75B62B500CF0 FA416A372E7985BB53D93A51D92AE316945D71BD5715 D994F772ED298192F1579C1A27B91A3EAAA920548A43 C8D65AA05EEEF904832FB7E98C1753F65BF79C16F7A 8A05A6E5137AE3716D931D37A318FDD51A0DFE5121E 258E2C9A85CC34E8F521E03707EB2B8A8E5D5FAB381 446A5629949500100000009135E4803*

Sample Code

This section contains sample code detailing the use of the `fuseaction=api.jsonPostCardEntry` parameter for the tokenization request.

WARNING! The sample code was designed for demonstration purposes only. Do not attempt to copy or use the sample code.

Sample AJAX Call

```
onSubmit: function(e) {

        var myself = this;

        e.preventDefault();


        this.hideError();

        if($(this.form).valid()){

                var mask = "********";

                postData = {

                        fuseaction:"api.jsonPostCardEntry",

                        i4go_accessblock:$(this.form).find("input[name=i4go_accessB
                lock]").val(),

                        i4go_cardtype:$(this.form).find("select[name=i4go_cardType]
                ").val(),

                        i4go_cardnumber:$(this.form).find("input[name=i4go_cardNumb
                er]").val(),

                        i4go_expirationmonth:$(this.form).find("select[name=i4go_ex
                pirationMonth]").val(),

                        i4go_expirationyear:$(this.form).find("select[name=i4go_exp
                irationYear]").val(),

                        i4go_cvv2code:$(this.form).find("input[name=i4go_cvv2Code]"
                ).val()

                };


                $(this.form).find("input[name=i4go_cardNumber]").blur().val(mask+po
        stData.i4go_cardnumber.slice(-4));

                $(this.form).find("input[name=i4go_cvv2Code]").blur().val(mask.slic
        e(0,postData.i4go_cvv2code.length));

                $("#dialog-wait").dialog("open");


                if (this.settings.debug || this.parentSettings.remoteDebug) {
```

```
        console.log("(i4goTrueTokenRemote) onSubmit(): Posting
payment data:",this.settings.server);
}
$.ajax({
        url: myself.settings.server+"/index.cfm",
        type: "POST",
        dataType: "json",
        timeout: 15000,
        data: postData,
        success: function(data) {
                myself.forwardTokenResponse.apply(myself,arguments);
        },
        error: function(jqXHR, textStatus, errorThrown){
                if (myself.settings.debug ||
        myself.parentSettings.remoteDebug) {
                        console.log("(i4goTrueTokenRemote)
                onSubmit(): AJAX post failed: ",textStatus);
                        console.log("(i4goTrueTokenRemote)
                onSubmit(): TOKENIZATION FAILED FIRST ATTEMPT -
                RETRYING STARTING WITH NEW ACCESSBLOCK...");
                }

                // we're going to try twice - the first time just
        failed!

                // get a new i4go_accessBlock
                $.ajax({
                        url: "index.cfm",
                        type: "GET",
                        dataType: "json",
                        timeout: 15000,
                        data: {
                                fuseaction: "get.refreshAccessBlock",
                        // You must write this web service call.
                                i4go_accessBlock:
                        myself.settings.accessBlock
                        },
                        success: function(data) {
                                myself.settings.server =
                        data.i4go_server;
                                myself.settings.accessBlock =
                        data.i4go_accessblock;
                                postData.i4go_accessblock =
                        data.i4go_accessblock;

                                if (myself.settings.debug ||
                        myself.parentSettings.remoteDebug) {

                                        console.log("(i4goTrueTokenRe
                                mote) onSubmit(): Second attempt,
                                posting payment
                                data:",myself.settings.server);

                                }

                                $.ajax({

                                        url:
                                myself.settings.server+"/index.cfm",

                                        type: "POST",

                                        dataType: "json",

                                        timeout: 25000,

                                        data: postData,

                                        success: function(data) {

                                                myself.forwardTokenRes
                                        ponse.apply(myself,arguments)
                                        ;

                                        },

                                        error: function(jqXHR,
                                textStatus, errorThrown){

                                                console.warn("(i4goTru
                                        eTokenRemote) onSubmit():
                                        AJAX post failed:
                                        ",textStatus);

                                                myself.displayError(te
                                        xtStatus);

                                                myself.forwardTokenRes
                                        ponse({

                                                        i4go_response:
                                                "FAILURE",

                                                        i4go_responsec
                                                ode: "-101",

                                                        i4go_responset
                                                ext: "Payment POST
                                                failed: "+textStatus

                                                });

                                        }

                                });

                        },

                        error: function(jqXHR, textStatus,
                errorThrown){

                                console.warn("(i4goTrueTokenRemote)
                        onSubmit(): Access block refresh failed:
                        ",textStatus);
```

```
                                                        myself.displayError(textStatus);
                                                        myself.forwardTokenResponse({
                                                                i4go_response: "FAILURE",
                                                                i4go_responsecode: "-102",
                                                                i4go_responsetext: "Access
                                                                block refresh failed:
                                                                "+textStatus
                                                        });
                                                }
                                        });
                                }
                        });
                        //$("button[type=submit],
                input[type=submit]").attr('disabled',true);
                }
}
```

i4Go Native Wallet Handling and Direct POST

Note: Before you start, review the Implementing Apple Pay and Google Pay Wallets section.

There are four steps to manually enable and use Apple Pay and Google Pay buttons directly.

Step 1 – i4auth Access Call (`fuseaction=account.authorizeClient`)

Standard i4go access call (`fuseaction=account.authorizeClient`) to retrieve an accessBlock. This call must include basket information as defined in the `authorizeClient` part of the API call.

Step 2 – Wallet Info Gathering

The get3ds call must be performed to get the information needed for setting up the buttons, as well as Apple Pay merchant driven negotiations to retrieve the cryptogram.

Request

GET [i4m URL from fuseaction=account.authorizeClient call]?fuseaction=get.get3dsInfo&i4go_accessBlock=[accessBlock from fuseaction=account.authorizeClient call]

Response

{

```
		"success": 1,
		"errorMsg": "",
		"walletConfig": {
			"merchantID": 107367,
			"countryCode": "US",
			"googlePay": {
				"gateway": "shift4",
				"allowedCardNetworks": [
					"AMEX",
					"DISCOVER",
					"JCB",
					"MASTERCARD",
					"VISA"
				],
				"merchantDomain": "ss-myportal.s4-test.com",
				"merchantId": "BCR2DN6TVPKLD5QV",
				"environment": "TEST", // Informational only.
				"allowedAuthMethods": [
					"CRYPTOGRAM_3DS",
					"PAN_ONLY"
				],
				"authJwt":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiJ9.eyJtZXJjaGFudElkIjoiQkNSMkRONlRWUEtMRDVRViIsImV4cCI6MS42MDc3
MTU2NUU5LCJtZXJjaGFudE9yaWdpbiI6InNzLW15cG9ydGFsLnM0LXRlc3QuY29tIiwiaWF0IjoxLjYwNzcxMjA1RTl9.lGq
BOuqndvSq322qYPYwwbhgVVp5emFFN_MJh5YhsyGeILemLw7ODfic1tSCkgCThOTIcblZSBw_Lp9lbo1CGw"
			},
			"providerDomain": "i4m.shift4test.com",
			"merchantName": "",
			"applePay": {
				"merchantIdentifier": "merchant.com.i4go.i4m",
				"supportedNetworks": [
					"amex",
                                "discover",
                                "jcb",
                                "masterCard",
                                "visa"
                        ]
"partnerInternalMerchantIdentifier": "MID-0001234567"
                },
"merchant": {
      "id": 1234567,
      "identifier": "MID-0001234567",
      "verified": false, //This must be true for the production environment. It may be false in test environments.
      "name": ""
    },
                },
                "currencyCode": "USD"
        },
        "jwt":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJPcmdVbml0SWQiOiI1ODJiZTlkZWRhNTI5MzJhOTQ2YzQ1YzQiLCJPYmpl
Y3RpZnlQYXlsb2FkIjp0cnVlLCJpc3MiOiI1ODJlMGEyMDMzZmFkZDEyNjBmOTkwZjYiLCJQYXlsb2FkIjp7IkNvbnN1bWV
yIjp7IlNoaXBwaW5nQWRkcmVzcyI6eyJQb3N0YWxDb2RlIjo4OTEzNCwiQ2l0eSI6IkxhcyBWZWdhcyIsIkNvdW50cnlDb
2RlIjo4NDAsIkxhc3ROYW1lIjoiU29tbWVycyIsIlN0YXRlIjoiTlYiLCJGaXJzdE5hbWUiOiJTdGV2ZSIsIkFkZHJlc3MxIjoiMTU
1MSBIaWxsc2hpcmUgRHIiLCJQaG9uZTEiOjcwMjU5NzI0ODAsIkFkZHJlc3MyIjoiIn0sIkFjY291bnQiOnsiQWNjb3VudE5
1bWJlciI6IiIsIkV4cGlyYXRpb25Nb250aCI6IiIsIkV4cGlyYXRpb25ZZWFyIjoiIn0sIkJpbGxpbmdBZGRyZXNzIjp7IlBvc3RhbE
NvZGUiOjg5MTM0LCJDaXR5IjoiTGFzIFZlZ2FzIiwiQ291bnRyeUNvZGUiOjg0MCwiTGFzdE5hbWUiOiJTb21tZXJzIiwiU3
RhdGUiOiJOViIsIkZpcnN0TmFtZSI6IlN0ZXZlIiwiQWRkcmVzczEiOiIxNTUxIEhpbGxzaGlyZSBEciIsIlBob25lMSI6NzAyNTk
3MjQ4MCwiQWRkcmVzczIiOiIifSwiRW1haWwxIjoic3RldmVAc2hpZnQ0LmNvbSJ9LCJPcmRlckRldGFpbHMiOnsiT3JkZ
XJDaGFubmVsIjoiUyIsIk9yZGVyTnVtYmVyIjoiMDAwMDAwMDAwMSAiLCJBbW91bnQiOjQ0NDQsIkN1cnJlbmN5Q29
kZSI6ODQwfX0sImV4cCI6MS42MDc3MTM4NUU5LCJpYXQiOjEuNjA3NzEyMDVFOSwianRpIjoiM0VDMENDNTgtRTY
1Ri1DNDIxLUMxMUY5RkQzREUwNDI3M0EiLCJDb25maXJtVXJsIjoiaHR0cHM6Ly9odHRwczovL3NzLWk0Z28taTRtLn
M0LXRlc3QuY29tL2luZGV4LmNmbT9mdXNlYWN0aW9uPXdzLmNjYUNhbGxiYWNrIn0.MGofP62SvcV6la29Apx0gn6
pYOTQnKAQVqi0A_tLY6s",
        "basket": {
                "Consumer": {
                        "MobilePhone": "",
                        "BillingAddress": {
                                "PostalCode": 89134,
                                "City": "Las Vegas",
                                "CountryCode": 840,
                                "LastName": "Sommers",
                                "State": "NV",
                                "FirstName": "Steve",
                                "Address1": "1551 Hillshire Dr",
                                "Phone1": "",
                                "Address2": ""
                        },
                        "ShippingAddress": {
                                "PostalCode": 89134,
                                "City": "Las Vegas",
                                "CountryCode": 840,
                                "LastName": "Sommers",
                                "State": "NV",
                                "FirstName": "Steve",
                                "Address1": "1551 Hillshire Dr",
                                "Phone1": "",
                                "Address2": ""
                        },
                        "Email2": "",
                        "Email1": "steve@shift4.com"
                },
                "OrderDetails": {
                        "OrderNumber": "0000000001 ",
                        "Amount": 44.44,
                        "CurrencyCode": 840
                }
        }
}
```

`walletConfig` Parameters

The table below defines the parameters in `walletConfig` that are needed for Apple Pay and/or Google Pay wallet support.

Parameters	Valid for `applePay`	Valid for `googlePay`
`merchantId`	Yes	Yes
`countryCode`	Yes	Yes
`providerDomain`	Yes	Yes
`merchantName`	Yes	Yes
`merchant`	Yes	Yes
`Gateway`		Yes
`allowedCardNetworks`		Yes
`merchantDomain`		Yes
`merchantId`		Yes
`environment`		Yes
`allowedAuthMethods`		Yes
`authJwt`		Yes
`merchantIdentifier`	Yes	
`supportedNetworks`	Yes	
`partnerInternalMerchantIdentifier`	Yes	

Step 3 – Button Setup

Using the walletconfig info received from the get3ds call above, piece together the fields required to provide the initialization parameters of Apple Pay and/or Google Pay.

Here is how we piece together the info to initialize the buttons:

```
function applePayInit(config) {
  try{
    _wallets_i4goTrueTokenObj.settings.debug && remoteLog("Checking for Apple Pay...");
    if((typeof config === "object")
      && (typeof config.applePay === "object")
      && (typeof config.applePay.merchantIdentifier === "string")
      && config.applePay.merchantIdentifier.length){
      if (window.ApplePaySession && ApplePaySession.supportsVersion(3) &&
ApplePaySession.canMakePayments()) {
        var promise = ApplePaySession.canMakePaymentsWithActiveCard(config.applePay.merchantIdentifier);
        promise.then(function (canMakePayments) {
          if(canMakePayments){
            $(".apple-pay-button").on("click",onApplePayClick);
            $(".apple-pay-button").show().removeClass("hidden").removeClass("pay-hidden");
          }
        }, function(error) {
             console.log("applePayInit Error: " + error.message);
        });
      } else {
        console.log("Apple Pay not found");
      }
    }else{
        console.log("Apple Pay not configured");
    }
  }catch(e){
    console.log("applePayInit Error: " + e.message);
  }
```

```
}

function googlePayInit( config ) {
  try{
    if((typeof config === "object")
      && (typeof config.googlePay === "object")
      && (typeof config.googlePay.authJwt === "string")
      && config.googlePay.authJwt.length){

      tokenizationSpecification.parameters.gateway = config.googlePay.gateway;
      tokenizationSpecification.parameters.gatewayMerchantId = config.merchantID.toString();
      allowedCardAuthMethods = config.googlePay.allowedAuthMethods;
      allowedCardNetworks = config.googlePay.allowedCardNetworks;
      baseCardPaymentMethod.parameters.allowedAuthMethods = allowedCardAuthMethods;
      baseCardPaymentMethod.parameters.allowedCardNetworks = allowedCardNetworks;
      cardPaymentMethod = Object.assign(
        {},
        baseCardPaymentMethod,
        {
          tokenizationSpecification: tokenizationSpecification
        }
      );

      onGooglePayLoaded();

    }else{
      console.log("Google Pay not configured");
    }
  }catch(e){
      console.error("googlePayInit Error: " + e.message);
  }
```

```
}
```

Sample button setup {Please also refer to your google developers guide}

```
{
  apiVersion: 2,
  apiVersionMinor: 0,
  callbackIntents: ['PAYMENT_AUTHORIZATION'],
  allowedPaymentMethods: [
   {
    type: 'CARD',
    parameters: {
     allowedAuthMethods: walletConfig.googlePay.allowedAuthMethods,
     allowedCardNetworks: walletConfig.googlePay.allowedCardNetworks,
    },
    tokenizationSpecification: {
     type: 'PAYMENT_GATEWAY',
     parameters: {
      gateway: walletConfig.googlePay.gateway,
      gatewayMerchantId: walletConfig.merchant?.identifier ?? walletConfig.merchantID.toString(),
     },
    },
   },
  ],
  merchantInfo: {
   merchantId: walletConfig.googlePay.merchantId,
   merchantName: walletConfig.merchantName || 'Merchant',
   merchantOrigin: walletConfig.googlePay.merchantDomain,
   authJwt: walletConfig.googlePay.authJwt,
  },
  transactionInfo: {
   displayItems: [
    {
        label: 'Donation',
        type: 'SUBTOTAL',
        price: donationAmount.toString(),
      },
    ],
    totalPriceStatus:'FINAL',
    totalPriceLabel: 'Total',
    totalPrice: donationAmount.toString(),
    currencyCode: walletConfig.currencyCode,
    countryCode: walletConfig.countryCode,
  },
}
```

Now buttons should be active – if configured and wallet is available.

Additional Apple Pay driven merchant verification and session exchange must be performed in order to retrieve Apple Pay token/cryptogram. Please review Apple Pay's documentation in the links below for more information:

- https://applepaydemo.apple.com/
- https://developer.apple.com/documentation/apple_pay_on_the_web/displaying_apple_pay_buttons_using_javascript

In addition, to see how this negotiation is performed in the Shift4 code for iFrame integration, see the following: https://i4m.i4go.com/js/wallets.js. While you will not be using this .js on your page, you can review the Apple section of the .js to get some pointers on how we handle the negotiation.

Some helpful pointers:

- The config.applePay.merchantIdentifier is received from the applepay section of the results of the get3ds call. From the sample 3ds call in the *Step 2 –Wallet Info Gathering* section, that value is "merchant.com.i4go.i4m" <= Please use the value from your get3ds call as it might differ.
- The config.applePay.partnerInternalMerchantIdentifier can also be retrieved from the results of the get3ds call. From the get3ds sample, we can see that value would be MID-0001234567 <= Please use the value from your get3ds call as will differ from what is here.
- For getApplePaySession, you will need the const url + const data

  const url =

  "https://" + _wallets_i4goTrueTokenObj.walletConfig.providerDomain + "/index.cfm?fuseaction=ws.applePaySession"; This will look something like:

```
"https://i4m.shift4test.com/index.cfm?fuseaction=ws.applePaySession"
```

where wallets_i4goTrueTokenObj.walletConfig.providerDomain is the providerDomain gotten from the get3ds call. The above const url shows the test value providerDomain (which will differ from production), so always use the value gotten from your get3ds call.

const data =

“validationURL= https://apple-pay-gateway.apple.com/paymentservices/startSession + wallets_i4goTrueTokenObj.settings.accessBlock" <= this is the accessblock from the authorizeClient fuseaction. Overall, you will have something like:

```
https://i4m.shift4test.com/index.cfm?fuseaction=ws.applePaySession&validation
URL=https://apple-pay-
gateway.apple.com/paymentservices/startSession&i4go_accessBlock=A0000,642C6AA
3F92231AE90D398944CDC6F520FD17A884032FEEB098D3125371E22D647F23A757F389AD9EE5F
1A434DB94D18283A17DB4BBFEBEFB435A52EA468C98EC9F405FB423B8F96D64671D243FA86B7F
64E447FB6AAE0279F208BDB1734E9F6BC01BCFB45BE5C914BB8EA0AF4E983B72036AE1F0ED081
80AF057FF2B2469599503AA2176CA2BD2DA435419B27A24CD3C2D0F958B60A20A140D332817B9
E2450AE4B7651F516A106B1875053067127B79BF0E6991ECAF9884723D94C31EEFA09DACE2B86
8121BEF6B2D589ACA6BD0B9624FC2D4819503C10A9FC183240A1070E8702450D4475285224034
3CDB920E9C2
```

There are some other steps and information needed, review the Apple Pay links:

- https://applepaydemo.apple.com/
- https://developer.apple.com/documentation/apple_pay_on_the_web/displaying_apple_pay_buttons_using_javascript

Step 4 – Token Exchange

Once wallet info has been received, you must exchange wallet token for Shift4 token. Upon receiving a wallet token, pass it to i4Go to get a Shift4 token.

Request

POST [i4go_server IP from access block (authorize client) request]?fuseaction=api.jsonPostCardEntry

Use form field values (NOT JSON as shown here, this was simply for readability):

{

"i4go_accessBlock":
"A0000,391B35E6DB2237101BA7A8D2D96257942D3593698F7A5345EF60E8065B0731B43CA1EAAB7832A686155
976854CD0AF68B47ADAF59B312B9758E53A335B61ED3C032389B0CAE41025878C9C586EF2C3DD5299D93AEC4A
9FF443F88DF65F79E451E1B4516582ECB847BBD4DB49AD84478BF441F08CE057C7396E1B30A92C82886D6220D0
33AE620FC87E370006753590E6FDA9B5A9C614387529112E0AC1E6BD3E68ABC65866DF886C3B0577402A05AB91
C2C9F5AC3412762CACA41A1C549F715DEC9CCFDF6B938DCADC38282F87533366CB5AA1BCBA470B77095D163E6
BF590616BFED9962080F708E6A6687E4BA2FBE0F3D6CF37B5243A4FA0A07E91AC3AAEBB007B705BC2DA6546AE3

```
C85AEE0E74DD5DC7E9615496DDFD354B3CC6ACC5632920B06D4A8B57CD1F77893067FAAD09BB107BC4420BC6C
09B7F5CCAA2399A0F9F1",
            "i4go_successURL": "",
            "i4go_failureURL": "",
            "i4go_cardNumber": "",
            "i4go_cvv2Code": "",
            "i4go_p2peBlock": "",
            "i4go_trackInformation": "",
            "i4go_applePayToken": "", //Supply wallet info here when Apple Pay wallet is in use.
            "i4go_googlePayToken":
"{\"apiVersionMinor\":0,\"apiVersion\":2,\"paymentMethodData\":{\"description\":\"Visa •••• 7059\",\"tokenizat
ionData\":{\"type\":\"PAYMENT_GATEWAY\",\"token\":\"{\\\"signature\\\":\\\"MEYCIQDAjHJsut6FUFHQJ9vHaz/
WwWDSr1LJ1E2W8K2OQpMFMgIhAKzdOvAEaPKTCMWHUN99AN4GDCeftQ3ZU7ukY8yAuhiy\\\",\\\"intermediate
SigningKey\\\":{\\\"signedKey\\\":\\\"{\\\\\\\"keyValue\\\\\\\":\\\\\\\"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQg
AEITiSX9PdY1V66C98vknpbpr2puqgVRK+DAoKKzGkW8hu7mgZ1ho8vA2xQ0GgqaKoG4PwTkWLKNangOvkioIoHA\\
\\\\\u003d\\\\\\\u003d\\\\\\\",\\\\\\\"keyExpiration\\\\\\\":\\\\\\\"1608349315283\\\\\\\"}\\\",\\\"signatures\
\\":[\\\"MEYCIQDlbB98DVDtlFrEtye0kZS1d7Fi50uMPanXkVMpolbQlgIhAIouwQNS3r+GQaJb+pZI5y3o5p4Brdj7i64G
1k/sLrCx\\\"]},\\\"protocolVersion\\\":\\\"ECv2\\\",\\\"signedMessage\\\":\\\"{\\\\\\\"encryptedMessage\\\\\\\"
:\\\\\\\"lYEHUpYnQxPVZsuasPzwbAZSw4PS6BJpkGzJ8UbeUSF81JRZO5tI0/WFvVUj+Xm36GvcHSTY9gH/rl3fpwu+pZ
gmBZtq43CyDk05QelqqV/TG052BoXWWn1CueECvh9ivZdvBP2z7fCYSY4l6g2lhQ5J5ymtnvCrawSQlmMlb2a2+PK8z1I
xhU5u7eHXaPag7ZMpjbQZao9w/26EWcxshzOJOaBMOuAV6g9G+u5Lj+A6qPRXI4jFir3MtlWaNf5TCHdAJwGZzYOmY
DFPbJlmMGqWEJ5010rg+9tR146Im0phXc5n+3iOjDje+QrgZduoYrQF67CE1/Ace7h+yrk98qO2Ff9GjvLM5x6r+dAmHd
MsWZglT4TjuUyfO9+0PnsiQcYiBGIJnxbIIEG2X3RxG4GmRLcIbb3h8rdtgKmoQXYZX//4qFD+HpYQ2fGfz1xS9iM62OEm
tY0NfEWfx+biEee0OTFHDargxw\\\\\\\u003d\\\\\\\u003d\\\\\\\",\\\\\\\"ephemeralPublicKey\\\\\\\":\\\\\\\"BJI
DVAIiUOgnXiS795ZU8MTYM7r13EhbIZ42/NxcmgXRMckertYPVGd98zJKifq6i6Abp+enhJbBn463C8p98WY\\\\\\\u0
03d\\\\\\\",\\\\\\\"tag\\\\\\\":\\\\\\\"y58ZPuXyQc6ZNh3AdjwsNBu8QPff56QO+q9sVryzjV0\\\\\\\u003d\\\\\\\"}\
\\"}\"},\"type\":\"CARD\",\"info\":{\"cardNetwork\":\"VISA\",\"cardDetails\":\"7059\"}}}",
            "i4go_referrer": "https://ss-myportal.s4-
test.com/index.cfm?action=development.i4mDemo&appReload=vFAowBJyQ2XOqOJA",
            "i4go_cardholderName": "",
            "cardNumber": "",
            "cvv2Code": "",
            "i4go_streetAddress": "",
            "i4go_postalCode": ""
}
```

Response

Use only highlighted fields below from the response and ignore the OTN response.

```
{
        "i4go_cardtype": "VS",
        "i4go_response": "SUCCESS",
        "i4go_responsecode": "1",
        "i4go_extendedcarddata":
"eyJ0aHJlZURTZWN1cmUiOnsiYXV0aGVudGljYXRpb25Tb3VyY2UiOjEsIndhbGxldElEIjoyMTZ9fQ==",
        "otn": {
                "secondaryerrorcode": 0,
                "expirationmonth": 12,
                "validavs": "",
                "preauthorizedtolerance": 0,
                "posttotaltime": 961,
                "cardnumber": "XXXXXXXXXXXX1111",
                "shorterror": "",
                "primaryerrorcode": 0,
                "tagstarttime": 1607713359530,
                "apiformat": 0,
                "invoice": "",
                "options": "",
                "functionrequestcode": "E0",
                "date": 123099,
                "uniqueid": "11118py3z9820f2r",
                "timeout": 25,
                "secondaryamount": 0,
                "customername": " ",
                "responsecode": "",
                "primaryamount": 0,
                "apisignature": "$",
                "preauthorizedamount": 0,
                "cvv2indicator": "",
                "tagtotaltime": 963,
                "poststarttime": 1607713359530,
                "expirationyear": 25,
        "apioptions": "RETURNUTOKEN,ALLDATA,MSRCAPABLE,MANUALENTRYCAPABLE",
        "reference": "WT0000000009",
        "cvv2code": "",
        "verbose": true,
        "cfhttp_statuscode": "200 OK",
        "errortext": "",
        "errorindicator": "N",
        "longerror": "",
        "merchantid": 0,
        "authsource": "E",
        "clerk": 0,
        "zipcode": "",
        "rawline1": "OK",
        "rawline2": "0,\"\",\"VS\",\"\",0.00,\"\",\"\",\"\"",
        "time": "000000",
        "expirationdate": "0000",
        "trackinformation": "MXXXXXXXXXXXX1111=0000?",
        "utoken": "411111-7EC765DA-000788-00001384-173347C849E",
        "cgi_host": "http://127.0.0.1:16448/API/s4tran_action.cfm",
        "streetaddress": "",
        "vendor": "Shift4.i4Go.4.1",
        "tagfinishtime": 1607713360493,
        "cardtype": "VS",
        "postfinishtime": 1607713360491,
        "authorization": "",
        "cvv2valid": "",
        "tranid": 0
    },
    "i4go_utoken": "411111-7EC765DA-000788-00001384-173347C849E",
    "i4go_expirationmonth": "12",
    "i4go_expirationyear": "2025",
```

"i4go_uniqueid": "11118py3z9820f2r" // Use the value returned with i4go_uniqueid when attempting authorization. Do not use the i4go_utoken value.

}

Appendix A – i4Go Response Codes and Messages

Appendix A describes i4Go Response Codes and Messages an end user may see.

Developers can display the Response Message returned by i4Go, or they can process the Response Code and display a customized response message.

The following table describes each Response Code and the corresponding Response Message end users will see (if the developer chose to display the Response Message rather than customizing their own) in the event of a success or an error during the process.

Note: Not all response codes and messages will be applicable as they're based on the implementation method.

Response Code	Response Message	Explanation
1	SUCCESS	The i4Go request was successful.
`authorizeClient` Function Errors		
300	Server does not accept `authorizeClient` function	The `authorizeClient` function failed because the i4Go server neither requires nor accepts the `authorizeClient` function.

Response Code	Response Message	Explanation
301	Not authorized	One of the following errors has occurred with the required and hidden parameter fields on the payment information form: • The i4Go server does not have the value submitted with the `i4go_clientip` parameter in queue. • The required i4Go Entry Parameters and the `i4go_accessblock` parameter were not returned to the specified i4Go server name. • The value submitted with the `i4go_accessblock` parameter is invalid. • The allotted time for using the corresponding values to the `i4go_clientip` and `i4go_accessblock` parameters has expired. • Shift4 Risk Management Services has returned a `risk assessment` value of `D` or `E`.
302	`authorizeClient` Error: Please try again.	Unable to complete the `authorizeClient` request.
302	`authorizeClient` Error: Communication error - #cfhttp.statusCode#. Please try again.	A Shift4 communication service is not responding.
302	`authorizeClient` Error: Invalid XML response - no children. Please try again.	A Shift4 communication service is responding in an incorrect format.
302	`authorizeClient` Error #val(xmlDoc.response.xmlAttributes.code)#: Please try again.	A Shift4 communication service responded with an error.
302	`authorizeClient` Error: No notes returned. Please try again.	A Shift4 communication service is not returning expected data.

Response Code	Response Message	Explanation
302	`authorizeClient` Error: No server returned. Please try again.	A Shift4 communication service is not returning expected data.
503	USAGE ERROR: i4go_accessToken is required (or i4Go_accountId and i4Go_siteId)	Valid credentials, either an AccessToken or an AccountID and SiteID, were not supplied.
503	USAGE ERROR: Invalid i4go_accessToken format	The supplied AccessToken is improperly formatted.
505	USAGE ERROR: i4go_clientIP is required	The ClientIP address was not supplied.
505	USAGE ERROR: Invalid i4go_clientIP format (IPv4 format required)	The supplied ClientIP address is improperly formatted.
Configuration Errors		
	Note: Configuration Errors alert the developer to problems during the development and approval process. The end user should never see these messages.	
503	USAGE ERROR: i4Go_AccessToken is required	The developer must modify the payment information form to include the merchant's Access Token. For additional information, see the *Prior to Implementation* section.

Response Code	Response Message	Explanation
504	CONFIGURATION ISSUE: Access block invalid or missing	The value submitted with the `i4go_accessblock` parameter field is invalid (due to length) or missing. The developer must contact Shift4 to enable this function. Once enabled, i4Go returns an access block when a successful `authorizeClient` request has been received. The developer must modify the payment information form to return this value with the `i4go_accessblock` parameter (which includes the merchant's Access Token).
505	USAGE ERROR: Invalid i4Go_ClientIP format (IPv4 format required)	The value submitted with the `i4go_clientip` parameter field is invalid.
506	USAGE ERROR: i4Go_ClientIP is required	The developer must modify the payment information form to post the end user's IP address to i4Go.
Informational Messages		
601	i4Go down for maintenance - try again later	The i4Go system is down for maintenance.
602	Lighthouse Transaction Manager down for maintenance - try again later	Lighthouse Transaction Manager is temporarily unavailable due to maintenance.
603	Lighthouse Transaction Manager merchant database down for maintenance - try again later	Lighthouse Transaction Manager is operational, but the merchant database is temporarily unavailable due to maintenance.

Response Code	Response Message	Explanation
Blacklist Errors		
9827	Country [country code] is blacklisted	The end user is attempting to process a transaction from a country that has been blacklisted for security reasons. i4Go will not allow connections from this country.
9828	IP Address [IP address] is blacklisted	The end user is attempting to process a transaction from an IP address that has been blacklisted for security reasons. i4Go will not allow connections from this IP address.
Data Entry Errors		
101	Invalid or missing card type indicator	One of the following errors has occurred: • A card type was not submitted with the `i4go_cardtype` parameter field on the payment information form. • A card type was submitted with the `i4go_cardtype` parameter field on the payment information form, but it was not a valid Shift4 card type. • A valid card type was submitted with the `i4go_cardtype` parameter field on the payment information form, but the card type is not in use for the merchant based on LTM API information. The end user must correct the `i4go_cardtype` parameter field on the payment information form.

Response Code	Response Message	Explanation
102	Card type indicator does not match card number	A card number was submitted with the `i4go_cardnumber` parameter field on the payment information form, but it was not a valid card number based on the card type submitted with the `i4go_cardtype` parameter field on the payment information form. For example, a Visa payment card number was submitted and an American Express card type was submitted on the payment information form. The end user must correct the `i4go_cardtype`, `i4go_cardnumber`, or both parameter fields on the payment information form. (Refer to Response Code 101 for additional information about card types in i4Go.)
104	Invalid track information	The encoded track information for the swiped payment card submitted with the `i4go_cardnumber` parameter field on the payment information form is invalid, possibly because the magnetic strip is worn. The end user can try entering the payment card number manually in the `i4go_cardnumber` parameter field on the payment information form to resolve the error. If entering the payment card number manually does not resolve the error, or if the payment card number cannot be entered manually, a different payment card must be used. If you are using IT'S YOUR CARD gift cards, refer to the *Card Manufacturing Tips* document for more information about track encoding.

Response Code	Response Message	Explanation
105	Invalid or missing card number	A payment card number was not submitted with the required `i4go_cardnumber` parameter field on the payment information form, or i4Go could not validate the payment card number submitted. i4Go validates payment card numbers using the Luhn mod 10 check configured in the dedicated UTG running in the Shift4 Data Center. i4Go validates both IT'S YOUR CARD and third-party gift card numbers against LTM, assuming third-party gift card numbers have been imported. The end user must correct the `i4go_cardnumber` parameter field on the payment information form.

Response Code	Response Message	Explanation
106	Invalid, missing or expired expiration date	One of the following errors has occurred: • An expiration month or year was not submitted with the `i4go_expirationmonth` or `i4go_expirationyear` parameter field on the payment information form. • An expiration month or year was submitted with the `i4go_expirationmonth` or `i4go_expirationyear` parameter field on the payment information form, but it was not a valid expiration month or year (for example, month 13 or year 2110) or it was submitted in an incorrect format. • An expiration month or year was submitted with the `i4go_expirationmonth` or `i4go_expirationyear` parameter field on the payment information form, but the expiration month or year has already passed. The end user must correct the `i4go_expirationmonth`, `i4go_expirationyear`, or both parameter fields on the payment information form.

Response Code	Response Message	Explanation
107	Invalid or missing CVV2 information	One of the following errors has occurred: • A card security code (CVV2) was not submitted with the `i4go_cvv2` parameter field on the payment information form. • A card security code (CVV2) was submitted with the `i4go_cvv2` parameter field on the payment information form, but it was invalid because it contained non-numeric characters. The end user must correct the `i4go_cvv2` parameter field on the payment information form.
108	Invalid CVV2 indicator	A card security code (CVV2) was submitted with the `i4go_cvv2` parameter field on the payment information form, but it was invalid because it did not match the card security code (CVV2) on file (with the card issuer) for the card. The end user must correct the `i4go_cvv2` parameter field on the payment information form.
109	Invalid or missing cardholder name	One of the following errors has occurred: • A cardholder name was not submitted with the `i4go_cardholdername` parameter field on the payment information form. • A cardholder name was submitted with the `i4go_cardholdername` parameter field on the payment information form, but it was invalid because it did not match the cardholder name on file for the card. The end user must correct the `i4go_cardholdername` parameter field on the payment information form.

Response Code	Response Message	Explanation
110	Invalid or missing street address for AVS	One of the following errors has occurred: • A street address was not submitted with the `i4go_streetaddress` parameter field on the payment information form. • A street address was submitted with the `i4go_streetaddress` parameter field on the payment information form, but it was an invalid submission for the Address Verification System (AVS). The end user must correct the `i4go_streetaddress` parameter field on the payment information form. AVS requires the numerical portion of the street address that corresponds to the payment card.
111	Invalid or missing postal code for AVS	One of the following errors has occurred: • A postal/ZIP code was not submitted with the `i4go_postalcode` parameter field on the payment information form. • A postal/ZIP code was submitted with the `i4go_postalcode` parameter field on the payment information form, but it was an invalid submission for the Address Verification System (AVS). The end user must correct the `i4go_postalcode` parameter field on the payment information form. AVS requires the postal/ZIP code from the address that corresponds to the payment card.

Appendix B – Additional Resources

Appendix B contains a list of additional resources that may be helpful and provide additional guidance. Direct any specific questions about these documents to their respective publishers.

Product Support

For assistance with this and any other Shift4 product, visit www.shift4.com/support/.

Live Support

Information about troubleshooting techniques and handling special problems that may occur during installation, configuration, or use can be obtained by contacting the Shift4 Customer Support team.

Development/Integration Support

For assistance handling problems that may occur when developing an application programming interface (API) and integrating Shift4 products with your system, contact your Shift4 API analyst.

Shift4 Guides and Documentation

The following Shift4 guides and documentation may provide additional helpful information and are located here:

- *Securing the Merchant's Site That Uses i4Go*
- *Account Administrator Guide*
- *Card Manufacturing Tips*

Industry Websites

The following sites provide additional industry guidelines and standards:

- https://www.pcisecuritystandards.org
- http://usa.visa.com/merchants/protect-your-business/index.jsp