Using the widgets for the IPS Assist payments

Using the Apple Pay widget for the IPS Assist payments

General information

The Apple Pay widget enables customers to use the Apple Wallet to pay in the online store without redirecting to external payment pages and without entering bank card details.

Apple Pay is supported on iOS and macOS devices.

By sight, the widget is presented in the form of an Apple Pay button, the process of authorization and payment of the order is started by clicking on this button.

Procedure for payment via widget

1. The customer selects products on the website of the online store.
2. The online store transfers the order data to the widget.
3. The customer clicks the Apple Pay button.
4. A pop-up window appears on the device with connected Apple Pay, where the customer can select one of the cards added to the Apple Pay mobile app before.
5. The customer selects the card and confirms the payment on the device with a password, TouchID or FaceID.
6. Apple Pay creates an encrypted package with a token.
7. The widget receives an Apple Pay token package.
8. The widget transfers an encrypted packet with a token and payment data to the IPS Assist.
9. The IPS Assist decrypts the packet with the token and payment data.
10. The IPS Assist makes payment through the processing of the settlement bank.
11. The IPS Assist returns the results of payment to the online store site.
12. The online store displays the payment result for the customer.

List of actions for organizing of payments using the widget

To organize the payment, you must perform the following preparatory steps:

- fill out a request for connecting Apple Pay payment to IPS Assist;
- register in Apple;
- create a certificate in your Personal account IPS Assist;
- sign the received certificate;
- upload the signed certificate in your Personal account IPS Assist;
- prepare the pages of the online store for payment;
  - use of the https protocol on the page with the widget and support for the TLS protocol version 1.2 are mandatory;
  - the domain must have a valid SSL certificate.

Deployment of the widget on the online store page

To deploy a widget on an online store page, do the following:

- place the widget on the payment page of the online store;
- customize the widget;
- check the possibility of payment using the widget;

- transfer the order parameters;
- receive and process the payment result.

Description of the widget

The widget is an HTML code and a JS script that you need to place and configure on the payment page of your online store.

Widget installation

In that place of the online store page where you plan to place the Apple Pay payment button, you need to add the following code:

```
<button id="apple-pay-button"></button>
```

You can specify the color, type and size of the button. Examples of button types and their description are available on the site https://developer.apple.com/design/human-interface-guidelines/apple-pay/overview/buttons-and-marks/.

To customize the appearance of the button, you must add a description of the style in the tag <head></head>.

```
<style>
  #apple-pay-button {
    display: none;
    background-color: black;
    background-image: -webkit-named-image(apple-pay-logo-white);
    background-size: 100% 100%;
    background-origin: content-box;
    background-repeat: no-repeat;
    width: 100%;
    height: 44px;
    padding: 10px 0;
    border-radius: 10px;
  }
</style>
```

For more information on customizing the appearance of the button, see https://developer.apple.com/documentation/apple_pay_on_the_web/displaying_apple_pay_buttons.

Order parameters transfer

After clicking on the Apple Pay button on the page of the online store, an *applePaySession* session is created in which you need to transfer the order parameters.

The order amount and currency of the order, as well as supported card types, should be passed as order parameters.

```
    Const currency = $('#currency').val(); // currency
    Const paymentRequest = {
      countryCode: region.toUpperCase(),
      currencyCode: currency.toUpperCase(),
      total: {
        label: 'Your label', // payment name
        amount: $('#amount').val() //order amount
      },
      supportedNetworks:['masterCard', 'visa'],
      merchantCapabilities: [ 'supports3DS' ] //supported cards
    };
Const applePaySession = new window.ApplePaySession(1, paymentRequest);
```

Two methods are used to process the session:

- *onvalidatemerchant;*
- *onpaymentauthorizad.*

The *onvalidatemerchant* method verifies the payment session, the method is executed when the Apple Pay pop-up window is displayed.

The *onpaymentauthorizad* method is executed when the payment transaction is confirmed. The method acquires the payment token received from Apple Pay, as well as the order parameters.

List of order parameters

Parameter	Description
merchant_id	The merchant identifier in IPS Assist
amount	Payment amount, in original currency
currency	Order currency
ordernumber	Order number in the merchant payments system
comment	Order comment
email	Customer's e-mail
firstname	Customer's first name
lastname	Customer's last name
middlename	Customer's middle name

The values may include field names of the completed payment form, for example:

```
var data = {
            token : event.payment.token,
            merchant_id : $('#merchant_id').val(),
            email : $('#email').val(),
            amount : $('#amount').val(),
            currency : $('#currency').val(),
            ordernumber : $('#ordernumber').val(),
            email : $('#email').val(),
            firstname : $('#firstname').val(),
            middlename : $('#middlename').val(),
            lastname : $('#lastname').val(),
            comment : $('#comment').val()
};
```

After payment is completed, the server returns the order number and status as response. To process the response, you must add the appropriate code:

```
$.post("/pay/tokenpay_widget_ap.cfm", JSON.stringify(data)).then(function (result) {
  //payment processing example:
  if (!result.hasOwnProperty('firstcode') && JSON.stringify(result.order.orderstate) == '"Approved"' && JSON.
stringify(result.order.orderstate) == '"Delayed"') {
    applePaySession.completePayment(ApplePaySession.STATUS_SUCCESS);
  } else {
    applePaySession.completePayment(ApplePaySession.STATUS_FAILURE);
  }
});
```

If for some reason the order payment was unsuccessful, then the service will return error messages (with appropriated values of the firstcode and secondcode parameters).

The script uses the Apple Pay API, which is used:

- iOS 10+in the Safari browser and in SFSafariViewController objects, as well as on iPhone models using Secure Element (SE и 6+);
- macOS 10.12+ in Safari on computers with Touch ID (MacBook Pro) or with a connected iPhone or Apple Watch to confirm payments.

Script example

Below you can see an example of a widget script that can be placed on the payment page of an online store.

```
<script type="text/javascript">
document.addEventListener('DOMContentLoaded', function(){
    if (window.ApplePaySession) {
        //checking payment options and displaying the Apple Pay button
        if (ApplePaySession.canMakePayments) {
            document.getElementById('apple-pay-button').style.display = 'block';
            document.getElementById('apple-pay-button').addEventListener('click', applePayButtonClicked);
```

```
        }
    } else {console.log("ApplePaySession not available"); }
});

function applePayButtonClicked() {
    const region = 'RU';
    const currency = $('#currency').val();//order currency
    const paymentRequest = {
        countryCode: region.toUpperCase(),
        currencyCode: currency.toUpperCase(),
        total: {
            label: 'Your label', //payment name
            amount: $('#amount').val()//order amount
        },
        supportedNetworks:['masterCard', 'visa'],
        merchantCapabilities: [ 'supports3DS' ]
    };
    const version = window.ApplePaySession.supportsVersion(3)
                    ? 3
                    : window.ApplePaySession.supportsVersion(2)
                    ? 2 : 1;
    const applePaySession = new window.ApplePaySession(version, paymentRequest);
    console.log("start session");

    //event handler to create merchant session.
    applePaySession.onvalidatemerchant = function (event) {
        console.log("onvalidatemerchant in");
        var data = {
            validationUrl: event.validationURL
        };
        console.log(JSON.stringify(data));

        //sending a request to the merchant server, then an API request to start the session
        $.post("/pay/apple_pay_comm.cfm", data).then(function (result) {
            applePaySession.completeMerchantValidation(result);
        });
    }

  // payment authorization event handler
    applePaySession.onpaymentauthorized = function (event) {
        console.log("onpaymentauthorized in");
        //var email = event.payment.shippingContact.emailAddress; //if the e-mail was requested
        //var phone = event.payment.shippingContact.phoneNumber; //if the phone number was requested
        // all options are on the site https://developer.apple.com/reference/applepayjs/paymentcontact
        // transfer of order parameters
        var data = {
            token : event.payment.token,
            merchant_id : $('#merchant_id').val(),
            email : $('#email').val(),
            amount : $('#amount').val(),
            currency : $('#currency').val(),
            ordernumber : $('#ordernumber').val(),
            email : $('#email').val(),
            firstname : $('#firstname').val(),
            middlename : $('#middlename').val(),
            lastname : $('#lastname').val(),
            comment : $('#comment').val()
        };
        //sending a request to the merchant server, then an API request to payment
        console.log(JSON.stringify(event.payment.token));
        $.post("/pay/tokenpay_widget_ap.cfm", JSON.stringify(data)).then(function (result) {
            if (!result.hasOwnProperty('firstcode') && JSON.stringify(result.order.orderstate) == '"Approved"'
&& JSON.stringify(result.order.orderstate) == '"Delayed"') {
            applePaySession.completePayment(ApplePaySession.STATUS_SUCCESS);
            } else {
            applePaySession.completePayment(ApplePaySession.STATUS_FAILURE);
            }
        });
    };
    applePaySession.begin();
}
```

```
</script>
<style>
  #apple-pay-button {
    display: none;
    background-color: black;
    background-image: -webkit-named-image(apple-pay-logo-white);
    background-size: 100% 100%;
    background-origin: content-box;
    background-repeat: no-repeat;
    width: 100%;
    height: 44px;
    padding: 10px 0;
    border-radius: 10px;
  }
</style>
```

Using the Google Pay widget for the IPS Assist payments

General information

The Google Pay widget enables customers Google to pay in the online store without redirecting to external payment pages and without entering bank card details.

By sight, the widget is presented in the form of a Google Pay button, the process of authorization and payment of the order is started by clicking on this button. The widget independently requests for a token and pays. On the side of the online store, the response is processed and the response is returned to the customer.

Using the Google Pay widget allows customer to pay with tokenized Google Pay cards. On devices without the installed Google Pay application, the customer will be asked to select a stored card from Google.

⚠ Google Pay operates with Visa and MasterCard.

Procedure for payment via widget

1. The customer selects products on the website of the online store.
2. The online store transfers the order data to the widget.
3. The customer clicks the Google Pay button.
4. A special browser dialog opens, in which the customer can select one of the cards stored before.
5. The customer selects the card and confirms the payment on the device.
6. Google Pay application creates an encrypted package with a token.
7. The widget receives a Google Pay token package.
8. The widget transfers an encrypted packet with a token and payment data to the IPS Assist.
9. The IPS Assist decrypts the packet with the token and payment data.
10. The IPS Assist makes payment through the processing of the settlement bank.
11. The IPS Assist returns the results of payment to the online store site.
12. The online store displays the payment result for the customer.

List of actions for organizing of payments using the widget

To organize the payment, you must perform the following preparatory steps:

- fill out a request for connecting Google Pay payment to IPS Assist;
- register in Google and fill out the registration form https://services.google.com/fb/forms/googlepayAPIenable/;
- verify compliance with Google's branding requirements;
- confirm domain on Google;
- prepare the pages of the online store for payment;
    - use of the https protocol on the page with the widget is mandatory;
    - the domain must have a valid SSL certificate.

You must consider the requirements of Google https://payments.developers.google.com/terms/sellertos, including a list of prohibited goods and services https://payments.developers.google.com/terms/aup as well as branding requirements https://developers.google.com/pay/api/web/guides/brand-guidelines.

Deployment of the widget on the online store page

To deploy a widget on an online store page, do the following:

- place the widget on the payment page of the online store;
- customize the widget;
- check the possibility of payment using the widget;
    - transfer the order parameters;
    - receive and process the payment result.

Description of the widget

The widget is an HTML code and a JS script that you need to place and configure on the payment page of your online store.

Widget installation

In that place of the online store page where you plan to place the Apple Pay payment button, you need to add the following code:

Widget installation

In that place of the online store page where you plan to place the Apple Pay payment button, you need to add the following code:

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

The Google Pay payment button will be placed in the *id="container"* tag. A tag identifier can be assigned a different value, if necessary. To do this, you must make the appropriate changes to the *createButton* method of the *addGooglePayButton* function, for example:

```
document.getElementById('mynewcontainer').style.display = 'block'; //mynewcontainer – new tag ID
```

Additionally, on the same page, you must connect the JS script to call the Google Pay API and the JS payment script.

```
<script async src="https://pay.google.com/gp/p/js/pay.js" onload="onGooglePayLoaded()"></script>
```

Widget customization

To configure the environment, you must set the *environment* parameter in the *getGooglePaymentsClient* function. To operate with real data, you must specify the value '*PRODUCTION*', and for testing - the value '*TEST*'.

```
function getGooglePaymentsClient() {
  if ( paymentsClient === null ) {
  paymentsClient = new google.payments.api.PaymentsClient({environment: 'TEST'});
  }
  Return paymentsClient;
}
```

Next, you need to specify the type of authentication cards that will be accepted for payment in the online store. For this, the *allowedCardAuthMethods* parameter is used, which can take the following values:

"PAN_ONLY" - authentication of cards stored in a Google account;

"CRYPTOGRAM_3DS" - authentication of cards stored as Google Pay tokens is used only on mobile devices with the installed Google Pay application.

```
const allowedCardAuthMethods = ["PAN_ONLY", "CRYPTOGRAM_3DS"];
```

You should also determine which cards of payment systems will be accepted for payment in the online store. The *allowedCardNetworks* parameter is used for this:

```
const allowedCardNetworks = ["MASTERCARD", "VISA"];
```

Now you need to insert the online store identifier into the widget script (*gatewayMerchantId* parameter):

```
const tokenizationSpecification = {
  type: 'PAYMENT_GATEWAY',
  parameters: {
    'gateway': 'assist',
    'gatewayMerchantId': '02510116604241796260'
  }
}
```

The connected Google Pay API JS script launches a handler that checks the device and displays a button:

```
function onGooglePayLoaded() {
  const paymentsClient = getGooglePaymentsClient()
  paymentsClient.isReadyToPay(getGoogleIsReadyToPayRequest()) //device check
    .then(funtion(response) {
      If (response.result) {
      addGooglePayButton(); //button display
       prefetchGooglePaymentData();
       }
    })
    .catch(function(err) { //error processing
      console.error(err); //error output to the console
    })
}
```

To display the button, the script calls the *addGooglePayButton* function. In this case, you can customize the appearance of the Google Pay button. To do this, make changes to the createButton method:

- *buttonColor* – sets the color of the button; possible values are black and white;
- *buttonType* - sets the button type, possible values are long and short.

For more information on adding a button, see the Google Pay API documentation https://developers.google.com/pay/api/web/reference/object?hl=ru#ButtonOptions.

An example of the *addGooglePayButton* function that adds a large black button:

```
function addGooglePayButton() {
  const paymentsClient = getGooglePaymentsClient();
  const button =
     paymentsClient.createButton({onClick: onGooglePaymentButtonClicked, buttonColor:'black',
buttonType:'long'}); //button creation
        document.getElementById('container').appendChild(button); // adding a button to a page
        document.getElementById('container').style.display = 'block';
}
```

When setting up the button, you must consider the requirements for branding Google.

If the device supports payment using Google Pay, a button will be displayed in accordance with the settings.

Clicking the button initiates the transfer of operation information to Google Pay using the *getGoogleTransactionInfo* function.

The following parameters should be passed for the operation:

- *currencyCode* — order currency;
- *totalPrice* — order amount;
- *totalPriceStatus* — order amount status.

The *totalPriceStatus* parameter can take the following values:

- *NOT_CURRENTLY_KNOWN* — is used to check payment options;
- *ESTIMATED* — order amount can be changed later, depending on the response;
- *FINAL* — the order amount is final, will not change in the process.

Example of *getGoogleTransactionInfo* function:

```
function getGoogleTransactionInfo() {
  return {
    currencyCode: $('#currency').val(), //order currency
    totalPriceStatus: 'FINAL', //order amount status
    totalPrice: $('#amount').val() //order amount
  };
}
```

After the payment is confirmed by the customer, the result will be returned to the *processPayment* function.

To make a payment, order data should be transferred to the widget (*processPayment* function).

List of the *ProcessPayment* function parameters

Parameter	**Description**
merchant_id	The merchant identifier in IPS Assist
amount	Payment amount, in original currency
currency	Order currency
ordernumber	Order number in the merchant payments system
comment	Order comment
email	Customer's e-mail
firstname	Customer's first name
lastname	Customer's last name
middlename	Customer's middle name

The values may include field names of the completed payment form.

Example of the *processPayment* funktion:

```
function processPayment(paymentData) {
  var data = {
        paymentData : paymentData,
        merchant_id : $('#merchantId').val(), //merchant ID
        amount : $('#amount').val(), //order amount
        currency : $('#currency').val(), //currency
        ordernumber : $('#ordernumber').val(), //order number
        email : $('#email').val(), //customer’s email
        firstname : $('#firstname').val(), //Customer's first name
        middlename : $('#middlename').val(),//Customer's middle name
        lastname : $('#lastname').val() //Customer's last name
        comment : $('#comment').val() //
  };
```

After payment using Google Pay is completed, the server returns the order number and status in response. To process the response, you must add the appropriate code to this *processPayment* function:

```
$.post("/pay/tokenpay_widget_gp.cfm", JSON.stringify(data)).then(function (result) {
   //here should be processing the response from the payment service
   // this is an response example {"order":{"ordernumber":"2019.03.11-664","orderstate":"Approved"}}
});
```

If for some reason the order payment was unsuccessful, then the service will return error messages (with appropriated values of the firstcode and secondcode parameters).

The script uses the Google Pay API, which is used:

- on all mobile devices, regardless of operating system;
- in browsers Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge, Opera or UCWeb UC Browser.

Google Pay does not have the ability to store a special test card, so when testing it will display a real card. However, in the test environment of Google this card will be replaced by a test one and test card data will be returned to the script. In this way you can use a stored real card without debiting from it.

Script example

Below you can see an example of a widget script that can be placed on the payment page of an online store.

```
<script type="text/javascript">
const baseRequest = {
  apiVersion: 2,
  apiVersionMinor: 0
};

const allowedCardNetworks = ["MASTERCARD", "VISA"];
const allowedCardAuthMethods = ["PAN_ONLY", "CRYPTOGRAM_3DS"];

const tokenizationSpecification = {
  type: 'PAYMENT_GATEWAY',
  parameters: {
    'gateway': 'assist',
    'gatewayMerchantId': '02510116604241796260'
  }
}

const baseCardPaymentMethod = {
  type: 'CARD',
  parameters: {
    allowedAuthMethods: allowedCardAuthMethods,
    allowedCardNetworks: allowedCardNetworks,
    billingAddressRequired: true,
    billingAddressParameters: {"format": "MIN"}
   }
}

const cardPaymentMethod = Object.assign(
  {},
  baseCardPaymentMethod,
  {
    tokenizationSpecification: tokenizationSpecification
  }
);

let paymentsClient = null;

function getGoogleIsReadyToPayRequest() {
  return Object.assign(
      {},
      baseRequest,
      {
        allowedPaymentMethods: [baseCardPaymentMethod]
      }
  );
}

/**
* Configure support for the Google Pay API
*
* @see {@link https://developers.google.com/pay/api/web/reference/object#PaymentDataRequest|PaymentDataRequest}
* @returns {object} PaymentDataRequest fields
*/
function getGooglePaymentDataRequest() {
  const paymentDataRequest = Object.assign({}, baseRequest);
  paymentDataRequest.allowedPaymentMethods = [cardPaymentMethod];
  paymentDataRequest.transactionInfo = getGoogleTransactionInfo();
  paymentDataRequest.merchantInfo = {
    // @todo a merchant ID is available for a production environment after approval by Google
    // See {@link https://developers.google.com/pay/api/web/guides/test-and-deploy/integration-
```

```
checklist|Integration checklist}
    merchantId: '16590966430175452581',
    merchantOrigin: 'www.assist.ru',
    merchantName: 'ASSIST Merchant'
  };
  return paymentDataRequest;
}

/**
* Return an active PaymentsClient or initialize
*
* @see {@link https://developers.google.com/pay/api/web/reference/client#PaymentsClient|PaymentsClient
constructor}
* @returns {google.payments.api.PaymentsClient} Google Pay API client
*/
function getGooglePaymentsClient() {
  if ( paymentsClient === null ) {
    paymentsClient = new google.payments.api.PaymentsClient({environment: 'TEST'});
  }
  return paymentsClient;
}

/**
* Initialize Google PaymentsClient after Google-hosted JavaScript has loaded
*
* Display a Google Pay payment button after confirmation of the viewer's
* ability to pay.
*/
function onGooglePayLoaded() {
  const paymentsClient = getGooglePaymentsClient();
  paymentsClient.isReadyToPay(getGoogleIsReadyToPayRequest())
      .then(function(response) {
        if (response.result) {
          addGooglePayButton();
          // @todo prefetch payment data to improve performance after confirming site functionality
           prefetchGooglePaymentData();
        }
      })
      .catch(function(err) {
        // show error in developer console for debugging
        console.error(err);
      });
}

/**
* Add a Google Pay purchase button alongside an existing checkout button
*
* @see {@link https://developers.google.com/pay/api/web/reference/object#ButtonOptions|Button options}
* @see {@link https://developers.google.com/pay/api/web/guides/brand-guidelines|Google Pay brand guidelines}
*/
function addGooglePayButton() {
  const paymentsClient = getGooglePaymentsClient();
  const button =
      paymentsClient.createButton({onClick: onGooglePaymentButtonClicked, buttonColor:'black',
buttonType:'long'});
      document.getElementById('container').appendChild(button);
      document.getElementById('container').style.display = 'block';
}

/**
* Provide Google Pay API with a payment amount, currency, and amount status
*
* @see {@link https://developers.google.com/pay/api/web/reference/object#TransactionInfo|TransactionInfo}
* @returns {object} transaction info, suitable for use as transactionInfo property of PaymentDataRequest
*/

function getGoogleTransactionInfo() {
  return {
    currencyCode: $('#currency').val(),
    totalPriceStatus: 'FINAL',
    // set to cart total
```

```
    totalPrice: $('#amount').val()
  };
}

/**
* g payment data to improve performance
*
* @see {@link https://developers.google.com/pay/api/web/reference
/client#prefetchPaymentData|prefetchPaymentData()}
*/
function prefetchGooglePaymentData() {
  const paymentDataRequest = getGooglePaymentDataRequest();
  // transactionInfo must be set but does not affect cache
  paymentDataRequest.transactionInfo = {
    totalPriceStatus: 'NOT_CURRENTLY_KNOWN',
    currencyCode: $('#currency').val()
  };
  const paymentsClient = getGooglePaymentsClient();
  paymentsClient.prefetchPaymentData(paymentDataRequest);
}

/**
* Show Google Pay payment sheet when Google Pay payment button is clicked
*/
function onGooglePaymentButtonClicked() {
  const paymentDataRequest = getGooglePaymentDataRequest();
  paymentDataRequest.transactionInfo = getGoogleTransactionInfo();

  const paymentsClient = getGooglePaymentsClient();
  paymentsClient.loadPaymentData(paymentDataRequest)
      .then(function(paymentData) {
        // handle the response
        processPayment(paymentData);
      })
      .catch(function(err) {
        // show error in developer console for debugging
        console.error(err);
      });
}

/**
* Process payment data returned by the Google Pay API
*
* @param {object} paymentData response from Google Pay API after user approves payment
* @see {@link https://developers.google.com/pay/api/web/reference/object#PaymentData|PaymentData object
reference}
*/
function processPayment(paymentData) {
  var data = {
          paymentData : paymentData,
          merchant_id : $('#merchantId').val(),
          amount : $('#amount').val(),
          currency : $('#currency').val(),
          ordernumber : $('#ordernumber').val(),
          email : $('#email').val(),
          firstname : $('#firstname').val(),
          middlename : $('#middlename').val(),
          lastname : $('#lastname').val(),
          comment : $('#comment').val()
      };
// For debug
console.log(data);

  $.post("/pay/tokenpay_widget_gp.cfm", JSON.stringify(data)).then(function (result) {
      // For debug
          console.log(result);
          // Example result:
          // {"order":{"ordernumber":"2019.03.11-664","orderstate":"Approved"}}

  });
}
```

```
</script>
<script async src="https://pay.google.com/gp/p/js/pay.js" onload="onGooglePayLoaded()"></script>
```

Back on top