SabreSonic Web v2012.2

Feature Brief – Custom JavaScript

Internal Only

Author: Halpern, Steven Edward

Feature Brief -- Custom Javascript

SabreSonic Web v2012.2

Software version 2012

Document Edition 1.1

DOCUMENT REVISION HISTORY

The following information is to be included with all versions of the document.

Ver #	Rev Date	Originator	Section #	Revision
1	8/31/2012	Steven Halpern	All	First Draft
1.1	9/7/2012	Steven Halpern	5-discussion of placing component definitions has been corrected. 7-function and variable definitions have been expanded.	Second Draft
1.2	5/2014	Steven Halpern	6–code example for Google Tag Manager added. 7–variable definitions for GTM data layer added.	

Table of Contents

1. Functional Description

The Custom JavaScript feature makes it possible for airlines to add functionality to IBE/storefront pages beyond the functionality provided by the standard components. For example, custom JavaScript can be used to add a custom thank you message based on the destination that the user books, to add links to online resources (such as Wikitravel) that can give users more information about their destinations, or to load an analytical package.

For the JavaScript coder, the custom JavaScript feature provides a number of useful functions and variables, such as `WhiteLabel.getIbeData(),` that make it easy to access any data that appears on the page and work with it in custom scripts.

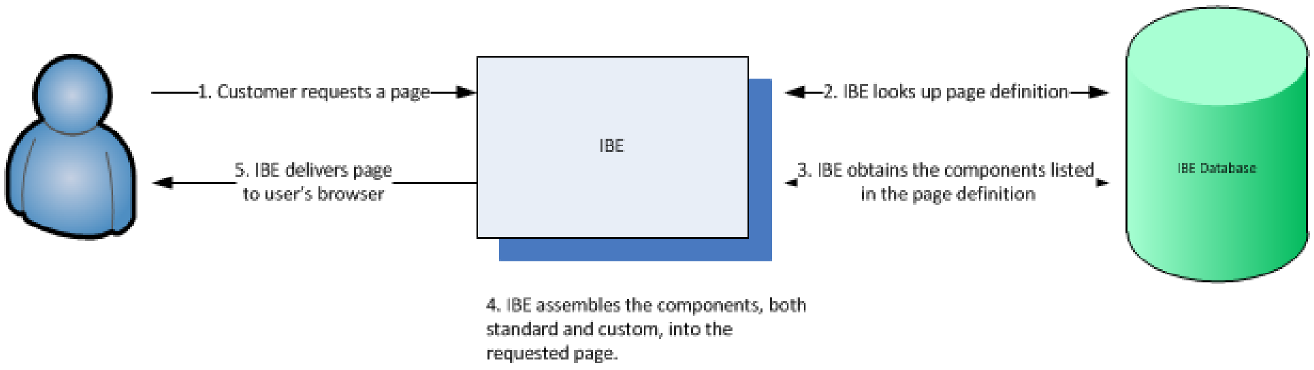
Note Two distinct tasks must be performed to add custom JavaScript to a page, each requiring a different set of skills. The business analyst or delivery manager faced with a request for custom JavaScript support will want to understand the two tasks and the two skill sets required to perform them:

1. Adding a custom JavaScript component to the page's definition: this must be done before the JavaScript is written. This is a fairly technical task that requires a thorough understanding of the IBE's internals (someone who is able to work with the XML representation of the page definition), and will probably be performed by a Sabre Migrations developer.
2. Writing JavaScript code. After the custom component has been added to the page definition, the JavaScript programmer can access the JavaScript file through a code editor in STAN. The primary skill required for this task is JavaScript coding, but some knowledge of the IBE's component model will certainly help.

It is certainly possible for one person to have both skills, but the roles are typically handled by different people: if an airline wants to add some custom JavaScript to a page, the Sabre Migrations team can add the component and the airline can then add, test, edit, and revise the JavaScript code through STAN.

Even when the two tasks are performed by two people, the JavaScript coder should be aware of the relationship between the JavaScript code and the component definition, because the nature of custom JavaScript as a component determines how it is executed when the page is loaded and how it interacts with data on the page. The JavaScript coder will want to be comfortable with these aspects of custom JavaScript components to ensure that the component behaves as expected. Similarly, the business analyst or delivery manager will want to understand the JavaScript coder's needs and communicate them to the XML developer so that the component is defined correctly.

As mentioned already, to implement custom JavaScript code, a custom JavaScript component must be added to the page. Components—and this includes both standard components and custom components-are the way that units of functionality are packaged and included on pages. Creating a custom component simply adds a new unit of functionality, one that is equivalent in a page's structure to the standard components already on the page. In other words, once a custom JavaScript component is added, it is the equivalent of such standard components as the progress bar and the login component. The following illustration shows how a set of components is assembled into a page and delivered to the user, demonstrating why packaging your custom JavaScript as a custom component is necessary to have it delivered as a part of a page:


1. Customer requests a page
5. IBE delivers page to user's browser
IBE
2. IBE looks up page definition
3. IBE obtains the components listed in the page definition
IBE Database
4. IBE assembles the components, both standard and custom, into the requested page.

This document has separate sections for the two main tasks.

2. Feature Activation

There is no activation for this feature. As soon as you define a custom JavaScript component it becomes part of the page.

3. Feature Configuration

There is no overall configuration for this feature. Each custom JavaScript component is configured in the XML tags that define it.

4. Feature Translations

N/A

5. Adding a Custom JavaScript Component to a Page

The structure of an IBE storefront can be exported from the database to an XML representation. Part of this XML representation is shown in the following example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ConfigurationRequest storefront="VAVA" create="true">
  <ImportFlows>
    <Flow clode="BOOKING">
      <Page rows="4" columns="2" name="AIR SEARCH PAGE">
       ...
      <Page rows="4" columns="2" name="CALENDAR PAGE">
       ...
      <Page rows="4" columns="2" name="AIR SELECT PAGE">
        ...
      <Page rows="4" columns="2" name="PASSENGERS PAGE">
        <Placeholder topRightY="4" topRightX="2" placeholderId="cnt 1" form="false" bottomLeftY="3"
                                                    bottomLeftX="0">
          <ComponentInstance initialized="true" initialState="header" form="false"
                                                    componentId="null 1" componentCode="scc"/>
        </Placeholder>
        <Placeholder topRightY="3" topRightX="1" placeholderId="cnt 2" form="true" bottomLeftY="1"
                                                    bottomLeftX="0">
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                                    componentId="prbar 1" componentCode="prbar"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                                    componentId="flomes 1" componentCode="flomes"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                                    componentId="psng 1" componentCode="psng"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                                    componentId="cic 1" componentCode="cic"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                                    componentId="cac 1" componentCode="cac"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                                    componentId="sbmt 1" componentCode="sbmt">
            <Property key="component.sbmt.type">passengers</Property>
          </ComponentInstance>
        </Placeholder>
        <Placeholder topRightY="3" topRightX="2" placeholderId="cnt 3" form="false" bottomLeftY="2"
                                                    bottomLeftX="1">
          <ComponentInstance initialized="true" initialState="initialized" form="true"
                                                    componentId="login 1" componentCode="login"/>
            <Property key="component.login.logout.redirect.force">false</Property>
          </ComponentInstance>
        </Placeholder>
        ...
      </Page>
      …
    </Flow>
  </ImportFlows>
</ConfigurationRequest>
</xml>
```

The example shows the definition of the Passengers page and several of the standard components that appear on it, including the progress bar (`component id=prbar`) and the login component (`component id=login_1`). Notice the hierarchical nature of these definitions: the storefront contains flows, flows contain pages, pages contain placeholders, and placeholders contain component instances.

To add a custom JavaScript component to a page you add tags that describe the new component. In most cases, custom JavaScript components should be the last components on the page, inside the last placeholder on the page. Custom JavaScript components typically access data from other components on the page, so you want your custom JavaScript to execute after the data-holding components on the page have been rendered, when their data will actually be available to your script.

The next example shows the same storefront, with tags added to define a custom JavaScript component:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ConfigurationRequest storefront="VAVA" create="true">
  <ImportFlows>
    <Flow clode="BOOKING">
      <Page rows="4" columns="2" name="AIR SEARCH PAGE">
       …
      <Page rows="4" columns="2" name="CALENDAR_PAGE">
       …
      <Page rows="4" columns="2" name="AIR SELECT PAGE">
        …
      <Page rows="4" columns="2" name="PASSENGERS PAGE">
        <Placeholder topRightY="4" topRightX="2" placeholderId="cnt 1" form="false" bottomLeftY="3"
                                                    bottomLeftX="0">
          <ComponentInstance initialized="true" initialState="header" form="false"
                                                    componentId="null 1" componentCode="scc"/>
        </Placeholder>
        <Placeholder topRightY="3" topRightX="1" placeholderId="cnt 2" form="true" bottomLeftY="1"
                                             bottomLeftX="0">
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="prbar 1" componentCode="prbar"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="flomes 1" componentCode="flomes"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="psng 1" componentCode="psng"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="cic 1" componentCode="cic"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="cac 1" componentCode="cac"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="sbmt 1" componentCode="sbmt">
            <Property key="component.sbmt.type">passengers</Property>
          </ComponentInstance>
          …
        </Placeholder>
        <Placeholder topRightY="3" topRightX="2" placeholderId="cnt 3" form="false" bottomLeftY="2"
                                             bottomLeftX="1">
          <ComponentInstance initialized="true" initialState="initialized" form="true"
                                             componentId="login 1" componentCode="login"/>
            <Property key="component.login.logout.redirect.force">false</Property>
          </ComponentInstance>
        …
          <ComponentInstance initialized="true" initialState="script" form="false"
                                             componentId="scc 1" componentCode="scc">
            <Property key="component.scc.script.path">script.js</Property>
            <Property key="component.scc.script.functionName">customScript</Property>
          </ComponentInstance>
        </Placeholder>
      </Page>
      …
    </Flow>
  </ImportFlows>
</ConfigurationRequest>
</xml>
```

Looking at the new tags, we see that it took two tags to define the custom JavaScript component, a `<ComponentInstance>` tag and two instances of the `<Property>` tag. Among the attribute values for these tags, the following are the most significant:

- The component instance's `initialState` attribute; for a custom JavaScript component this must be `script`.
- The component instance's `componentId` attribute; this is a unique ID for the new component. Try to use a meaningful name.
- The component instance's `componentCode` attribute; for a custom JavaScript component this must be `scc` (for “static content component”).
- The first instance of the `<property>` tag sets the `component.scc.script.path` key. This is the name that will be used for the component's JavaScript file. The default name is `script.js`.
- The second instance of the `<property>` tag sets the `component.scc.script.functionName` key. This is the name of the function that will be called when the component is loaded. The default name is `customScript`.

For information on the other attributes see the Tag Reference section.

Note that this example used the default values for the two property keys. It is possible to add multiple custom JavaScript components and use the default values for all of them. The result is a set of components that share one JavaScript file and one called function, which requires logic that branches according to the current page. For an example of this, see Code Example 1.

A more efficient (and generally recommended) practice is shown in the next example. It has two separate custom JavaScript components, and the two components not only have different component IDs (which is required), but they have different value for the `script.path` and `script.functionName` keys. The result is two different JavaScript files, each with its own function. In a storefront with several custom JavaScript components, which can appear on multiple pages in different combinations, this method minimizes the amount of JavaScript code that must be loaded for each page.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ConfigurationRequest storefront="VAVA" create="true">
  <ImportFlows>
    <Flow clode="BOOKING">
      <Page rows="4" columns="2" name="AIR SEARCH PAGE">
       ...
      <Page rows="4" columns="2" name="CALENDAR PAGE">
       ...
      <Page rows="4" columns="2" name="AIR SELECT PAGE">
        ...
      <Page rows="4" columns="2" name="PASSENGERS PAGE">
        <Placeholder topRightY="4" topRightX="2" placeholderId="cnt 1" form="false" bottomLeftY="3"
                                             bottomLeftX="0">
          <ComponentInstance initialized="true" initialState="header" form="false"
                                             componentId="null 1" componentCode="scc"/>
        </Placeholder>
        <Placeholder topRightY="3" topRightX="1" placeholderId="cnt 2" form="true" bottomLeftY="1"
                                             bottomLeftX="0">
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="prbar 1" componentCode="prbar"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="flomes 1" componentCode="flomes"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="psng 1" componentCode="psng"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="cic 1" componentCode="cic"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="cac 1" componentCode="cac"/>
          <ComponentInstance initialized="true" initialState="initialized" form="false"
                                             componentId="sbmt 1" componentCode="sbmt">
            <Property key="component.sbmt.type">passengers</Property>
          </ComponentInstance>
          …
        </Placeholder>
        <Placeholder topRightY="3" topRightX="2" placeholderId="cnt 3" form="false" bottomLeftY="2"
                                             bottomLeftX="1">
          <ComponentInstance initialized="true" initialState="initialized" form="true"
                                             componentId="login_1" componentCode="login"/>
            <Property key="component.login.logout.redirect.force">false</Property>
          </ComponentInstance>
        …
          <ComponentInstance initialized="true" initialState="script" form="false"
                                             componentId="scc 1" componentCode="scc">
            <Property key="component.scc.script.path">scc1.js</Property>
            <Property key="component.scc.script.functionName">customScript1</Property>
          </ComponentInstance>
          <ComponentInstance initialized="true" initialState="script" form="false"
                                          componentId="scc 2" componentCode="scc">
            <Property key="component.scc.script.path">scc2.js</Property>
            <Property key="component.scc.script.functionName">customScript2</Property>
          </ComponentInstance>
        </Placeholder>
      </Page>
      …
    </Flow>
  </ImportFlows>
</ConfigurationRequest>
</xml>
```

Summarizing the steps for this task, we have:

1. Determine the number of custom components for a page, and their component IDs.
2. Decide whether you are going to use a separate `.js` file for each component (recommended), or a single `.js` file with logic that branches according to the page it appears on. Decide on the names to be used for the `.js` files.
3. Export the current XML definition for the storefront you are working with.
4. Add the tags that define the custom JavaScript component, normally as the last component instances on the page.
5. Re-import the storefront definition. After re-importing, the custom JavaScript component will be visible in STAN.

Tag Reference

This section provides detailed information about the XML tags that define a custom JavaScript component.

The Placeholder Tag

Components, including custom JavaScript components, are defined inside placeholder tags, which locate a component or group of components on a page. You typically place custom JavaScript components as the last component instances in the last placeholder on the page, in which case you don't change any of the placeholder's attributes.

The placeholder tag has the following attributes:

<Placeholder> Tag Attribute	**Meaning**	**Values**
bottomLeftX	Locates the placeholder on the page, relative to the page's bottom left	
bottomLeftY	Locates the placeholder on the page, relative to the page's bottom left	
form	Indicates whether the placeholder contains a form	true\\false

placeHolderId	A unique Id for the placeholder.	string
topRightX	Locates the placeholder on the page, relative to the page's top right	
topRightY	Locates the placeholder on the page, relative to the page's top right	

The ComponentInstance Tag

The componentInstance tag defines a component. The table shows the attributes and values to use for custom JavaScript components.

<Component Instance> Tag Attribute	**Meaning**	**Values**
componentCode	Identifies the type of the component.	For custom JavaScript components, this should always be `scc`, for "static content component."
componentId	A unique identifier for each component. For custom JavaScript components, use meaningful IDs, such as `scc_1`.	string
initialized	For custom JavaScript components, this should always be `true`	true\\false
initialState	For custom JavaScript components, this should always be `script`.	`script`
form	Indicates whether the component is a form. For custom JavaScript components, this should always be `false`.	true\\false

The Property Tag

The property tag provides additional information about a components. A custom JavaScript component requires two instances of this tag, to define the two configuration keys:

<Property> Tag Key Attribute	**Meaning**	**Values**	
component.scc.script.path	The name of the file for the component's JavaScript code.	The default value is `script.js`; as discussed above, if you supply different filenames for each component, each component will have its own `.js` file. Or you can supply a single file name for all of your custom JavaScript, and then branch the logic inside that file according to the context.	*`xxxx`*`.js`
component.scc.script.functionName	The name of the JavaScript function to be executed when the component is rendered.	The default value is `customScript`	A valid JavaScript function name

6. Writing JavaScript Code

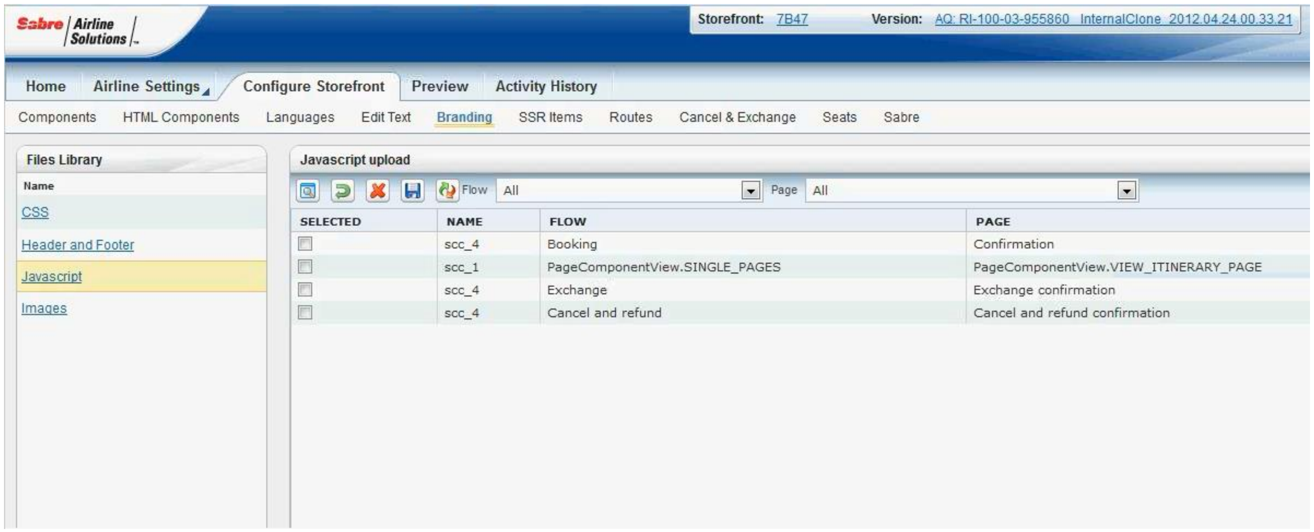
Once you have defined a component, the component will become visible in STAN, and you can use STAN to add and edit the JavaScript code.

Using the JavaScript Code Editor

To add or edit custom JavaScript code:

1. In STAN, click the *Configure Storefront* tab.
2. In the pop-up menu, click *Branding*.
3. In the *Files Library* pane, click *Javascript*. At this point, the right pane will become the *Javascript Upload* pane, and all of the custom JavaScript component instances for the storefront will be listed. If the list is long, you can use the *Flow* and *Page* controls to filter the list.
4. The component instances are identified by their component IDs (the IDs assigned in the XML). These IDs are listed in the *Name* column.
5. Notice the buttons at the top of the *Javascript Upload* pane. These buttons perform basic editing functions: *View, Revert, Clear, Download,* and *Replace.* The buttons give you access to the file specified in the XML with the `component.scc.script.path` key.

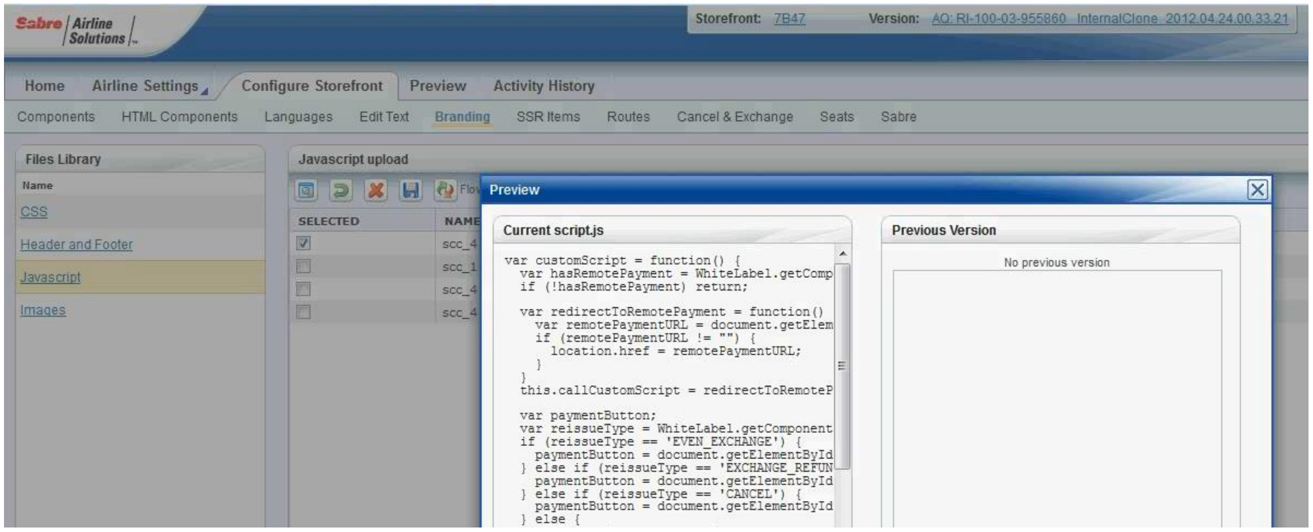
STAN: List of JavaScript Components



SELECTED	NAME	FLOW	PAGE
☐	scc_4	Booking	Confirmation
☐	scc_1	PageComponentView.SINGLE_PAGES	PageComponentView.VIEW_ITINERARY_PAGE
☐	scc_4	Exchange	Exchange confirmation
☐	scc_4	Cancel and refund	Cancel and refund confirmation

6. To work with a JavaScript file, locate the component instance you want to work with. Click the check box in the *Selected* column and click one of the editing function buttons.
7. For example, clicking the *View* button opens the JavaScript file for review:

STAN: JavaScript in open JavaScript Editor

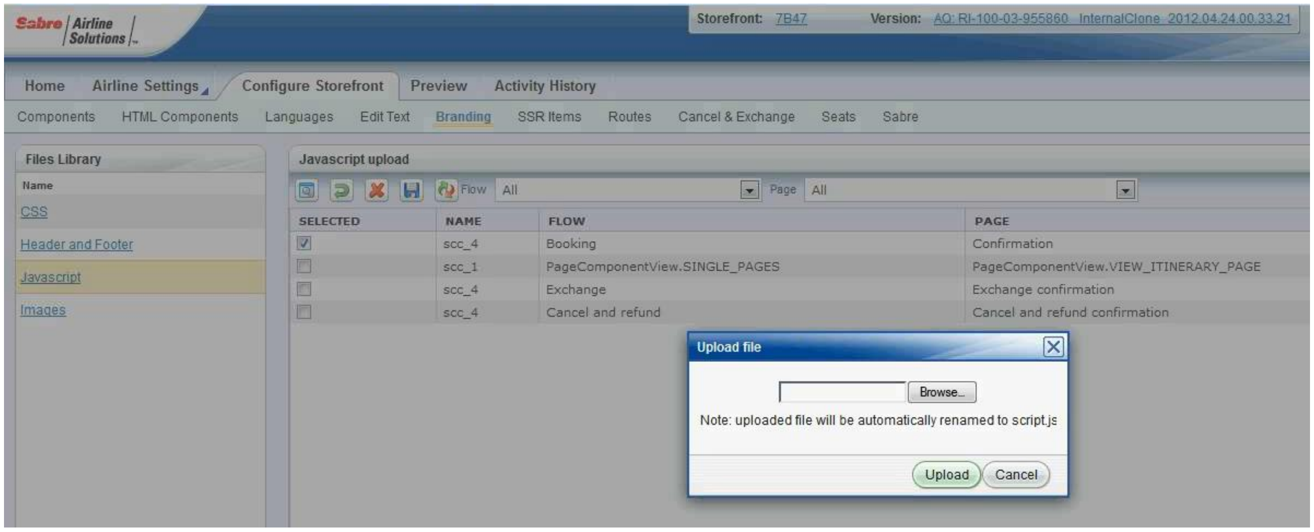


```
var customScript = function() {
  var hasRemotePayment = WhiteLabel.getComp
  if (!hasRemotePayment) return;

  var redirectToRemotePayment = function()
    var remotePaymentURL = document.getElem
    if (remotePaymentURL != "") {
      location.href = remotePaymentURL;
    }
  }
  this.callCustomScript = redirectToRemoteP

  var paymentButton;
  var reissueType = WhiteLabel.getComponent
  if (reissueType == 'EVEN_EXCHANGE') {
    paymentButton = document.getElementById
  } else if (reissueType == 'EXCHANGE_REFUN
    paymentButton = document.getElementById
  } else if (reissueType == 'CANCEL') {
    paymentButton = document.getElementById
  } else {
```

8. Or, clicking the *Replace* button opens this dialog which lets you upload a local file. You can develop locally with your preferred text editor, then use this feature to upload and test. (In this example, the XML definition specified the file name for this component instance as `script.js`, so the upload action will automatically rename any file you upload to `script.js`.)



SELECTED	NAME	FLOW	PAGE
✓	scc_4	Booking	Confirmation
	scc_1	PageComponentView.SINGLE_PAGES	PageComponentView.VIEW_ITINERARY_PAGE
	scc_4	Exchange	Exchange confirmation
	scc_4	Cancel and refund	Cancel and refund confirmation

Note: uploaded file will be automatically renamed to script.js

Code Example 1

This example shows you how to write the called function if you are using a single JavaScript file to support multiple component instances. The logic in this example uses the `sabre.config.pageCode` to get the current page, and then branches based on the current page.

```
var customScript = function (node) {
    var pageCode = sabre.config.pageCode;
    if (pageCode === 'PURCHASE PAGE') {
        purchasePageScript(node);
    } else if (pageCode === 'CONFIRMATION PAGE') {
        confirmationPageScript(node);
    }
};

function purchasePageScript(node) {
    /* custom logic only for purchase page */
}

function confirmationPageScript(node) {
    /* custom logic only for confirmation page */
}
```

Code Example 2

This example shows the called function from a JavaScript file that supports only a single component instance.

```
var customScript = function (node) {
    var journeySpan = WhiteLabel.getIbeData().journeySpan;
    var content = '';
    if (journeySpan === 'ONE WAY') {
        content = '<a class="translate" wl:translate="" href="' + sabre.config.global.applicationUrl
                      + '">Why fly only one way? Book return journey</a>';
    }
    node.innerHTML = content;
};
```

Code Example 3

This example shows code that collects a set of data about passenger activity on the current IBE page and passes it back to the airline's Google Tag Manager account. The airline then accesses the accumulated data through its GTM account for analytical purposes. To use this code, create a custom JavaScript component and add it to each IBE page from which the airline wants to collect data. (This will typically be all pages.)

Note that the developer implementing this code should replace the variable *GTM-XXXX* in the snippet with the specific airline's GTM account number before the code is deployed.

```
<!-- Google Tag Manager -->


<noscript><iframe src="//www.googletagmanager.com/ns.html?id=GTM-XXXX"
height="0" width="0" style="display:none;visibility:hidden"></iframe></noscript>
<script>
    (function(w,d,s,l,i){w[l]=w[l]||[];
     w[l].push({'gtm.start':new Date().getTime(),event:'gtm.js'});
     var f=d.getElementsByTagName(s)[0],j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';
     j.async=true;j.src='//www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j,f);
     })
    (window,document,'script','dataLayer','GTM-XXXX');
</script>
<!-- End Google Tag Manager -->
```

At runtime, the snippet acts as follows:

1. Makes a call to the airlines Google Tag Manager account and downloads several macros stored there that define the data to be collected.
2. Executes the macros to collect the data.
3. Returns the collected data to the airline's GTM account.

The data defined in the macros and returned to the airline's account is represented by the *dataLayer* variable in the snippet. The data items collected and returned in the data layer are listed in the JavaScript Code Reference section.

7. JavaScript Code Reference

This section provides detail about the variables and functions introduced with the custom JavaScript feature.

Custom JavaScript Functions

Useful functions that can be called in custom JavaScript

Function Name	Description	Syntax

WhiteLabel.getIbeData()	Returns any of the Sabre exposed variables listed in the following	`var name = WhiteLabel.getIbeData()`
		
		or
		
		`var name =`

	table	`WhiteLabel.getIbeData().`*`sabreExposedVariable`*

Sabre Exposed Variables	Descriptio n	Variable Name	Format	Example
Pax Info (logged in user):	User's unique sign-in data	loggedUser	Object	
prefix	Title	prefix	Alpha	MR
First name	First name	firstName	Alpha	JOHN
last name	Last name	lastName	Alpha	DOE
tier level	Frequent traveler tier	tierLevel	Alpha-numeric	
FF Number	Passenger frequent traveler number	ffNumber	Alpha-numeric	121ab31
emails	Passenger email	emails		ABC@SABRE.COM
Journey span	Type of booking	journeySpan	Predefined: ONE_WAY ROUND_TRIP MULTI_CITY	ONE_WAY
Cabin Class	Seated cabin	cabinClass	Predefined: ECONOMY PREMIUM_ECONO MY BUSINESS FIRST	ECONOMY
Promo code	Promotional code	promoCode	Predefined	
Air search itinerary parts:	Itinerary	itineraryParts	Object	
departure airport	Departure city	departureAirport	3 Letter Airport Code	MEL
arrival airport	Arrival city	arrivalAirport	3 Letter Airport Code	JFK

date	Travel date	date	mm/dd/yyyy hh:mm:ss	2012/09/04 00:00:00
Redemption flag	Redemption code	redemption	Predefined	Redemption
Passenger type map	Type of passenger	passengers	Object	{ ADT: 2, CHD: 1 }
Currency	Currency of reserved fare	currency	Predefined	USD
Language	Language and country code (examples: en_US, en_GB, fr_FR, ar_AE, etc.)	language	Predefined	en_US
Selected offers:	Optional offers	selectedOffers	Object	
fare amount (monetary)	Base fare amount	fareAmount	Numeric	108.180
Branded fare itinerary part:				
Segments:	Itinerary segment(s)	segments	Predefined	
Departure date	Departure date	departure	mm/dd/yyyy hh:mm:ss	2012/09/04 06:00:00
arrival date	Arrival date	arrival	mm/dd/yyyy hh:mm:ss	2012/09/04 07:30:00
departure airport	Departure city	departureAirport	3 Letter Airport Code	JFK
arrival airport	Arrival city	arrivalAirport	3 Letter Airport Code	MEL
flight number	Flight number	flightNumber	Numeric	800
airline code	Marketing carrier	airlineCode	Alpha-numeric	VA

operating airline code	Operating carrier	operatingAirlineCode	Alpha-numeric	VA
cabin class	Seated cabin	cabinClass	Predefined	ECONOMY
brand ID	Name of Brand	brandId	Predefined	EP
booking class	Fare class	bookingClass	Alpha	T
fare basis	Fare code	fareBasis	Alpha-numeric	TZDSV
next day indicator	Change of day during flight	nextDayIndicator	Boolean	false
Total (monetary)	Total amount	total	Numeric	108.180
Passengers info:	Passenger details	passengersInfo	Object	
prefix	Title	prefix	Alpha	MR
first name	First name	firstName	Alpha	JOHN
last name	name	lastName	Alpha	DOE
tier level	Frequent traveler tier	tierLevel	Alpha-numeric	
FF Number	Passenger frequent traveler number	ffNumber	Alpha-numeric	121ab31
emails	Passenger email	emails		ABC@SABRE.COM
Insurance code	Travel insurance	insuranceCode	Alpha-numeric	NO
Selected ancillaries map	Optional items for sale	selectedAncillariesPerPaxIndex	Object	
code		code	Predefined	0CC
prices		prices	Object	

Travel part:		travelPart	Object	
origin	Origination	origin	3 Letter Airport Code	JFK
destination	Destination	destination	3 Letter Airport Code	MEL
type		type	Predefined: (SEGMENT / ITINERARY_PART)	
PNR Number	Record locator	pnrNumber	Alpha-numeric	ABC23D
Remote payment indicator		remotePayment	Predefined	
Reissue type		reissueType	Predefined	

Custom JavaScript Variables

This section lists useful variables that can be accessed in a custom JavaScript.

Variable Name	Description	Values
node	Identifies the custom JavaScript component's container on the rendered page. You can use this to access and manipulate the container's HTML. For an example, see Code Example 2. (When a page is are rendered, custom JavaScript components on the page are rendered as `<div>` tags that have `id` attributes equal to the custom component IDs.)	String
sabre.config.pageCode	Identifies the current pages, using one of the codes listed in the next column	AIR_SEARCH_PAGE AIR_SELECT_PAGE PASSENGERS_PAGE SEATS_PAGE PURCHASE_PAGE CONFIRMATION_PAGE SANDBOX_PAGE COMPONENT_PREVIEW_PAGE OFFSITE_PAYMENT_PAGE ERROR_PAGE

		MAINTENANCE_PAGE MY_BOOKING_PAGE MY_TRIPS_PAGE MY_ACCOUNT_PAGE ACCOUNT_CREATE_PAGE VIEW_ITINERARY_PAGE FLIGHT_EXCHANGE_PAGE FLIGHT_STATUS_PAGE EXCHANGE_AIR_SEARCH_PAGE EXCHANGE_AIR_SELECT_PAGE EXCHANGE_REVIEW_PAGE EXCHANGE_PAYMENT_PAGE EXCHANGE_PASSENGERS_PAGE EXCHANGE_SEATS_PAGE EXCHANGE_CONFIRMATION_PAGE CANCEL_REFUND_REFUND_PAGE CANCEL_REFUND_CONFIRMATION_PAGE EXCHANGE_ERROR_REDIRECT ANCILLARY_PAGE CALENDAR_PAGE CHANGE_PAX_DETAILS_PAGE EXCHANGE_ANCILLARY_PAGE UPGRADE_ANCILLARY_PAGE UPGRADE_SEATS_PAGE UPGRADE_PURCHASE_PAGE UPGRADE_CONFIRMATION_PAGE

Data Layer Variables

The first table lists items that are collected on all IBE pages.

Data Layer Variable Name	Variable Description	Data Format/Example
PageName	Name of the SSW/IBE page from which the data was collected.	
SiteLanguage	Language used on the page from which the data was collected.	
FlowType	The IBE booking flow from which the data was collected.	Values: BOOKING REDEMPTION EXCHANGE UPGRADE ANCILLARIES_MTO SINGLE_PAGES CHECK_IN
Storefront	Name of the storefront from which the data was collected	Depends on storefront codes established by airline.

Data Layer Variable Name	Variable Description	Data Format/Example
BuildNumber	Build number of the page from which the data was collected	
PromoCode	CAT5 or promo code entered by passenger into Promotional Code field.	
ErrorTextKey	Any text key(s) displayed to the passenger, such as error messages, on the page from which the data was collected.	
FFPLoggedIn	Frequent Flyer account number, if the passenger is logged in.	
FFPFields	Values entered into FFP fields on passenger details page separated by | if more than one is entered.	Values separated by the | character.
DateUTC	The server date when the data was collected in UTC.	YYYYMMDD
TimeUTC	The server time when that data was collected in UTC.	HHMMSS
ClientDateUTC	The client (the passenger's browser) date when the data was collected in UTC.	YYYYMMDD
ClientTimeUTC	The client (the passenger's browser) time when the data was collected in UTC.	HHMMSS
PNR	PNR	The PNR's ID code
Currency	The ISO 4217 3 letter currency code for the currency of the transaction.	AED, USD, etc.
TotalAmount	The total amount in the passenger's shopping cart.	Integer value
Product Level		
FlightOND	The origin and destination of the passenger's itinerary (for multicity or open jaw itineraries there will be several pairs). Each pair is separated from the others by -.	Each origin destination pair is represented by colon separated string: O:D To describe multi-city or open jaw itineraries, multiple pairs are used: O1:D1-D2:O1, or O1:D1-O2:D2-O3:D3-O4:D4 For example: LHR:AUH or LHR:AUH-MCT:SYD, or LHR:SYD-BNE:SIN-SIN:AUH
FlightCabin	The cabin of the passenger's flight.	Values: ECONOMY BUSINESS FIRST
FlightType	The type of the passenger's	RETURN

Data Layer Variable Name	Variable Description	Data Format/Example
	flight.	ONEWAY MULTICITY
FlightTripDates	The date(s) of the passenger's flight(s).	For one-way: YYYYMMDD For roundtrip: YYYYMMDD:YYYYMMDD For multi-city: YYYYMMDD:YYYYMMDD: YYYYMMDD
FlightTripDuration	Number of days between first departure date and last arrival date.	Integer value representing the number of days
FlightPace	Number of days between day data was collected and first departure date.	Integer value representing the number of days
FlightADTPax	Number of adult passengers in the booking.	Integer value from 0-9.
FlightCHDPax	Number of child passengers in the booking.	Integer value from 0-9.
FlightINFPax	Number of infant passengers in the bookings.	Integer value from 0-9.
FlightPaxTypes	Number of adults, children and infants.	#ADT:#CHD:#INF
FlightSegmentNumbers	Number of segments of a flight. (Segments are to be defined as separated by point of turnaround or ARNK.)	Integer value from 1 to 4.
FlightSegment1RBD	The RBD codes of the first segment in the itinerary.	Alpha values from A to Z, separated by (:) for each sector of the segment. For example: Y, or Y:W, or U:V:Y.
FlightSegment1OND	Airport codes of the first flight segment.	Three-letter airport codes, separated by colon (:). For example: O:D or O1:O2:D or O1:O2:O3:D
FlightSegment1OperatingAirline	Operating airlines of the first flight segment.	Two-letter airline codes, separated by colon (:) for each sector of the segment. For example: SU
FlightSegment1FlightNumber	Flight numbers of the first flight segment.	Flight numbers, separated by colon (:) for each sector of the segment. For example: 123
FlightSegment1DepartDateTime	Departure dates and times of the first flight segment.	Date time values, separated by colon (:) for each sector of the segment. For example: YYYYMMDDHHMM
FlightSegment1ArriveDateTime	Arrival dates and times of the first flight segment.	Date time values, separated by colon (:) for each sector of the segment. For example:

Data Layer Variable Name	Variable Description	Data Format/Example
		YYYYMMDDHHMM
FlightSegment2RBD	The RBD codes of the second segment in the itinerary.	Alpha values from A to Z, separated by (:) for each sector of the segment. For example: Y, or Y:W, or U:V:Y.
FlightSegment2OND	Airport codes of the second flight segment.	Three-letter airport codes, separated by colon (:). For example: O:D or O1:O2:D or O1:O2:O3:D
FlightSegment2OperatingAirline	Operating airlines of the second flight segment.	Two-letter airline codes, separated by colon (:) for each sector of the segment. For example: SU
FlightSegment2FlightNumber	Flight numbers of the second flight segment.	Flight numbers, separated by colon (:) for each sector of the segment. For example: 123
FlightSegment2DepartDateTime	Departure dates and times of the second flight segment.	Date time values, separated by colon (:) for each sector of the segment. For example: YYYYMMDDHHMM
FlightSegment2ArriveDateTime	Arrival dates and times of the second flight segment.	Date time values, separated by colon (:) for each sector of the segment. For example: YYYYMMDDHHMM
FlightSegment3RBD	RBDs of the third flight segment.	Alpha values from A to Z, separated by (:) for each sector of the segment. For example: Y, or Y:W, or U:V:Y.
FlightSegment3OND	Airport codes of the third flight segment.	Three-letter airport codes, separated by colon (:). For example: O:D or O1:O2:D or O1:O2:O3:D
FlightSegment3OperatingAirline	Operating airlines of the third flight segment.	Two-letter airline codes, separated by colon (:) for each sector of the segment. For example: SU
FlightSegment3FlightNumber	Flight numbers of the third flight segment.	Flight numbers, separated by colon (:) for each sector of the segment. For example: 123
FlightSegment3DepartDateTime	Departure dates and times of the third flight segment.	Date time values, separated by colon (:) for each sector of the segment. For example: YYYYMMDDHHMM
FlightSegment3ArriveDateTime	Arrival dates and times of the third flight segment.	Date time values, separated by colon (:) for each sector of the segment. For example:

Data Layer Variable Name	Variable Description	Data Format/Example
		YYYYMMDDHHMM
FlightSegment4RBD	RBDs of the fourth flight segment.	Alpha values from A to Z, separated by (:) for each sector of the segment. For example: Y, or Y:W, or U:V:Y.
FlightSegment4OND	Airport codes of the fourth flight segment.	Three-letter airport codes, separated by colon (:). For example: O:D or O1:O2:D or O1:O2:O3:D
FlightSegment4OperatingAirline	Operating airlines of the fourth flight segment.	Two-letter airline codes, separated by colon (:) for each sector of the segment. For example: SU
FlightSegment4FlightNumber	Flight numbers of the fourth flight segment.	Flight numbers, separated by colon (:) for each sector of the segment. For example: 123
FlightSegment4DepartDateTime	Departure dates and times of the fourth flight segment.	Date time values, separated by colon (:) for each sector of the segment. For example: YYYYMMDDHHMM
FlightSegment4ArriveDateTime	Arrival dates and times of the fourth flight segment.	Date time values, separated by colon (:) for each sector of the segment. For example: YYYYMMDDHHMM
Product Flight		
ProductFlightSKU	Constructed by concatenating the following: the number of pax types, the RBD codes, operating airline codes and flight numbers, flight dates and times and the origins and destinations of each segment.	Sequence of codes. Main items separated by |, segments separated by -, and within a segment, sectors are separated by : For example: One-way, for two adults and one child, AUH to LHR: 2ADT:1CHD:0INF|W-W|WW 123|201308310235|AUH:LHR Round-trip, for one adult and one infant, AUH to LHR and back: 1ADT:0CHD:1INF|W-W|WW 123-WW 124|201308311335-201309130915|AUH:LHR-LHR:AUH One way, for two adults and one child, DFW to MCT: 2ADT:1CHD:0INF|V:U:Y|WW001:WW002:WW003|201310271630:201310272100:201310290220|DFW:ORD:AUH:MCT
ProductFlightName	Flight O&D and Cabin.	Concatenation of flight O and D and cabin class. For example: BUSINESS LHR-AUH ECONOMY LHR-AUH-MCT-SYD

Data Layer Variable Name	Variable Description	Data Format/Example
ProductFlightCategory	Flight	Flight
ProductFlightAmount	Total amount of flight product, including taxes but excluding all ancillaries.	Numeric amount, in the currency specified in the Currency variable.
Product Extra Baggage		
ProductBaggageSKU	Merchandising codes for any baggage ancillaries selected by the passenger.	Concatenation of baggage merchandising codes r, separated by **|**. For example: BagCode1|BagCode2
ProductBaggageName	Identifier for baggage products	A string literal that identifies the baggage product: BAGGAGE
ProductBaggageCategory	Identifier for the ancillary product category	A string literal that identifies the ancillary product category: ANCILLARY
ProductBaggageAmount	Total amount for the selected baggage product.	Total for products in the baggage category. A numeric amount, in the currency specified in the Currency variable.
Product Seat		
ProductSeatSKU	SKU for the seat/seats selected by the passenger	Concatenation of the selected seat numbers separated by the **|** character. For example: SeatNumber1|SeatNumber2
ProductSeatName	Identifier for seat products	A string literal that identifies the seat product: SEAT
ProductSeatCategory	Identifier for the ancillary product category	A string literal that identifies the ancillary product category: ANCILLARY
ProductSeatAmount	Total amount for the selected seat product.	Total amount for products in the seat category selected by the passenger. A numeric value, in the currency specified in the Currency variable.
Product Carbon Offset		
ProductCarbonOffsetSKU	SKU for the carbon offset/offsets selected by the passenger	Concatenation of Carbon Offset codes separated by the **|** character. For example: *CarbonOffsetCode1*|*CarbonOffsetCode2*
ProductCarbonOffsetName	Carbon Offset	A string literal that identifies the carbon offset product: CARBON OFFSET
ProductCarbonOffsetCategory	Identifier for the ancillary product category	A string literal that identifies the ancillary product category: ANCILLARY
ProductCarbonOffsetAmount	Total amount of carbon offset product.	Total amount for products in the carbon offset category selected by the passenger. A numeric value, in the currency specified in the Currency variable.

The second table lists additional variables collected on the Confirmation pages (CONFIRMATION_PAGE, EXCHANGE_CONFIRMATION_PAGE, UPGRADE_CONFIRMATION_PAGE, and ANCILLARIES_MTO_CONFIRMATION_PAGE).

Variable Name	Description	Data Format/Example
Transaction Data		
transactionId	Unique transaction identifier	A string literal: PNR
transactionDate	Date of transaction	UTC date
transactionAffiliation	Partner or store	Storefront code, depends on storefronts set up by airline.
transactionTotal	Total value of the transaction	Numeric amount
transactionTax	Tax amount for the transaction	Numeric amount
transactionPaymentType	Payment type	A string identifying the FOP used by the passenger.
transactionCurrency	Currency of the transaction	A currency code.
transactionPromoCode	Discount or promotion codes used by the passenger	A string or strings (entered by the passenger, from promo codes created by the airline).
transactionProducts	List of items purchased in the transaction	Array of product identifiers.
TransactionProduct Data		
sku	Product SKU	A string identifying one of the following, as defined above, depending on the product: the ProductFlightSKU the ProductBaggageSKU the ProductSeatSKU the ProductCarbonOffsetSKU
name	Product name	A string specifying one of the following, as defined above, depending on the product: the ProductFlightName the ProductBaggageName the ProductSeatName the ProductCarbonOffsetName
category	Product category	A string specifying one of the following, as defined above, depending on the product: the ProductFlightCategory the ProductBaggageCategory the ProductSeatCategory the ProductCarbonOffsetCategory
price	Unit price	A numeric value corresponding to one of the following, as defined above, depending on the product: ProductFlightAmount ProductBaggageAmount ProductSeatAmount ProductCarbonOffsetAmount

Variable Name	Description	Data Format/Example
quantity	Number of items	Always 1

8. JavaScript Code Guidelines and Hints

Custom JavaScript Component Placement

The placement of custom JavaScript components on a page flow can affect the custom script's ability to access page variables. If a custom script is loaded before some other component that holds data value, the script will not be able to access the data. The most reliable way to avoid this problem is to place the custom JavaScript component as the last component in the page flow.

Custom JavaScript and jQuery

If you are writing custom JavaScript that uses jQuery, you need to explicitly load jQuery. There are two possible ways of doing this. The first loads both jQuery and the custom script from some external source, similar to what is done in the following example:

```
var customScript = function(node) {
//jQuery
    var jquery = document.createElement('script'); jquery.type = 'text/javascript';
    jquery.src = 'https://www.domain.com/…/jquery-1.7.2.min.js';
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(jquery, s);
//external script
    var customScript = document.createElement('script'); customScript.type = 'text/javascript';
    customScript.src = 'https://www.domain.com/…/custom.js';
    var s = document.getElementsByTagName('script')[0]; s.parentNode.appendChild(customScript, s);

};
```

The second approach, which is recommended, is to load jQuery and execute the custom JavaScript in a callback after jQuery has been loaded:

```
var customScript = function(node) {

                function loadScript(url, callback)
{

   var head = document.getElementsByTagName('head')[0];
   var script = document.createElement('script');
   script.type = 'text/javascript';
   script.src = url;
   script.onreadystatechange = callback;
   script.onload = callback;
   head.appendChild(script);
}

var customCode = function() {
  // ****************** insert code here **************************
  function someCustomFunction() {
  …
}
$(document).ready(someCustomFunction);
  //*************************************************************
};

loadScript("https://www.domain.com/.../jquery-1.7.2.min.js", customCode);
};
```