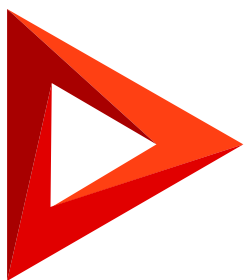


Landings

Version 8.0



This documentation is provided under restrictions on use and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this documentation, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

Table of Contents

Web-To-Object	4
Implementing the Web-to-Object service	4
External API of the Web-to-Object service	4
Using the Web-to-Object service	5
Set up web form for a custom object	5
Case description	6
Source code	6
Case implementation algorithm	6
Implement a handler to create an entity using a web form	17
1. Implement a custom handler	18
2. Register a custom handler in the database	21
The outcome of the example	22
Web-to-Case	22
The logic of the automatic filling of case fields.	23
Recommendations for the execution of project solutions	23
Create Web-to-Case landing pages	24
Steps to create Web-to-Case landing	24

Web-To-Object



Web-to-Object is a mechanism for implementing simple one-way integrations with Creatio. It enables you to create records in Creatio sections (leads, cases, orders, etc.) simply by sending the necessary data to the Web-to-Object service.

The most common cases of using the Web-to-Object service are the following:

- Integrating Creatio with custom landings and web forms. The service call is performed from a landing (a customized custom page with a web form), after the visitor submits the completed web form.
- Integrating with external systems to create Creatio objects.

Using Web-to-Object will enable you to configure the registration of virtually any objects in Creatio (e.g., a lead, an order or a case).

The [*Landings and web-forms*] section is used to work with landing in Creatio. This section is available in all Creatio products, however it might not be enabled in workplaces of certain products (e.g., Sales Creatio). Each record in the [*Landing and web-forms*] section corresponds to a landing page. The record edit page features a [*Landing setup*] tab.

Implementing the Web-to-Object service

The main functionality of the Web-To-Object mechanism is contained in the `WebForm` package and is common to all products. Depending on the product, this functionality may be extended by the Web-to-Lead (the `WebLeadForm` package), Web-to-Order (the `WebOrderForm` package), and [Web-to-Case](#) (the `WebCaseForm` package) mechanisms.

To process data received from a web-form of a landing, the `WebForm` package implements the `GeneratedObjectWebFormService` configuration service (the `Terrasoft.Configuration.GeneratedWebFormService` class). The data of the landing's web-form is accepted as the argument of the `public string SaveWebFormObjectData(FormData formData)` method. They are then passed to the `public void HandleForm(FormData formData)` method of the `Terrasoft.Configuration.WebFormHandler` class, in which the corresponding system object is created.

External API of the Web-to-Object service

To use the service, send a POST request to

```
[Creatio application path]/0/ServiceModel/GeneratedObjectWebFormService.svc/SaveWebFormObjectData
```

For example

```
http://mycreatio.com/0/ServiceModel/GeneratedObjectWebFormService.svc/SaveWebFormObjectData
```

The content type of the request is `application/json`. In addition to the required cookies, the JSON object containing the data of the web-form must be added to the content of the request. JSON object example:

```
{
  "formData": {
    "formId": "d66ebbf6-f588-4130-9f0b-0ba3414dafb8",
    "formFieldsData": [
      { "name": "Name", "value": "John Smith" },
      { "name": "Email", "value": "j.smith@creatio.com" },
      { "name": "Zip", "value": "00000" },
      { "name": "MobilePhone", "value": "0123456789" },
      { "name": "Company", "value": "Creatio" },
      { "name": "Industry", "value": "" },
      { "name": "FullJobTitle", "value": "Sales Manager" },
      { "name": "UseEmail", "value": "" },
      { "name": "City", "value": "Boston" },
      { "name": "Country", "value": "USA" },
      { "name": "Commentary", "value": "" },
      { "name": "BpmHref", "value": "http://localhost/Landing/" },
      { "name": "BpmSessionId", "value": "0ca32d6d-5d60-9444-ec34-5591514b27a3" }
    ]
  }
}
```

Using the Web-to-Object service

Integrating with custom landings and web-forms

Integrating with external systems

To integrate with external systems:

1. Create a new record in the [*Landing and web-forms*] section
2. Get the address to the service (`serviceUrl` property) and the identifier (the `landingId` property) from the configuration object of the created record.
3. Implement the process of sending a POST-request to the Web-to-Object service (at the received address) in the external system. Add the necessary data to the request in form of a JSON object. Set the value of the received identifier to the `formId` property of the JSON object.

Set up web form for a custom object



Create a custom Creatio object via a landing page web form on a third party website. Learn more about landing pages in the “[[Landing pages and web forms](#)] section” article.

The general procedure of creating a custom object via a web form is as follows:

1. Register a new landing page type.
2. Add an edit page for the web form.
3. Map the new landing page type to the created edit page.
4. Update the scripts for the web form record page.
5. Create and configure a landing page in the **[Landing pages and web forms]** section.
6. Deploy and set up a landing page with a web form.

Attention. Tracking website events only works for leads and does not work for custom objects.

Case description

Create a custom **[Contact]** object using a landing page web form.


Source code

You can download the package with an implementation of the case using the following [link](#).

Case implementation algorithm

1. Register a new landing page type

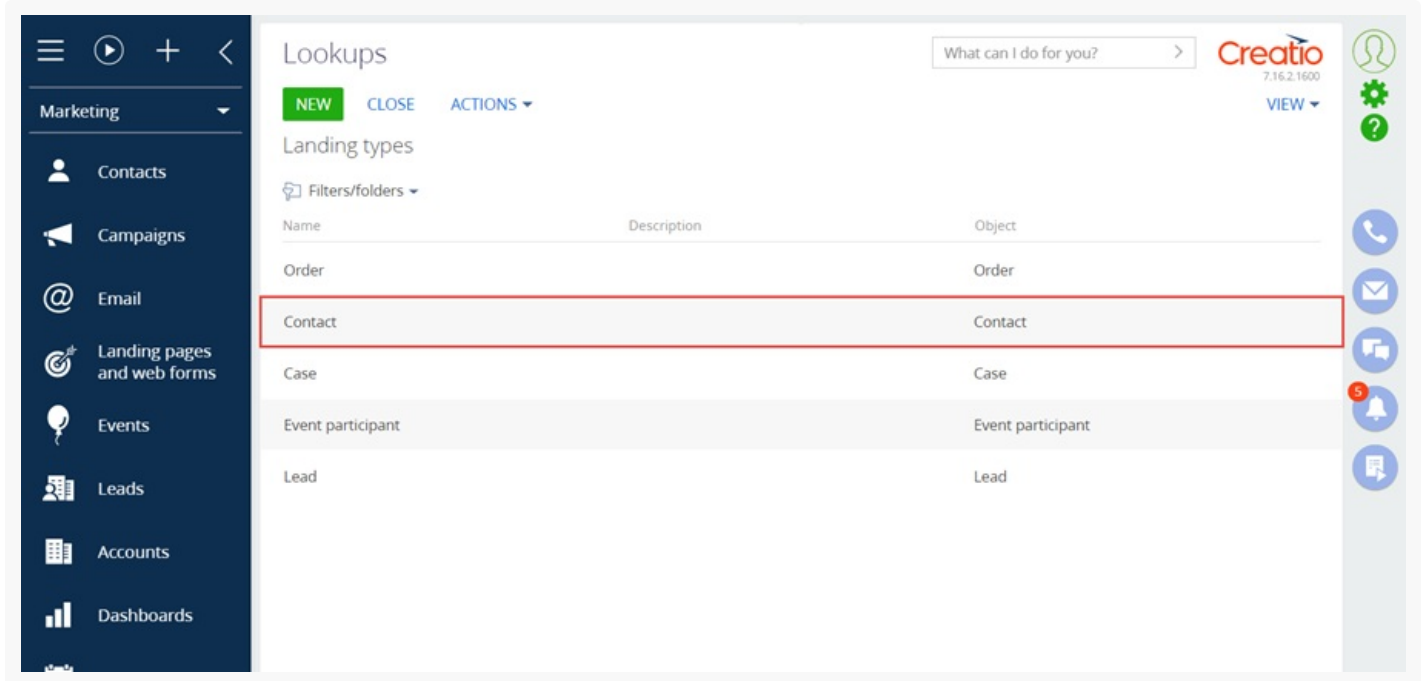
To do this:

1. Open the System Designer by clicking . Go to the **[System setup]** block -> click **[Lookups]**.
2. Select the **[Landing types]** lookup.
3. Add a new record.

In the created record, specify (Fig.1):

- **[Name]** - “Contact”
- **[Object]** - “Contact.”

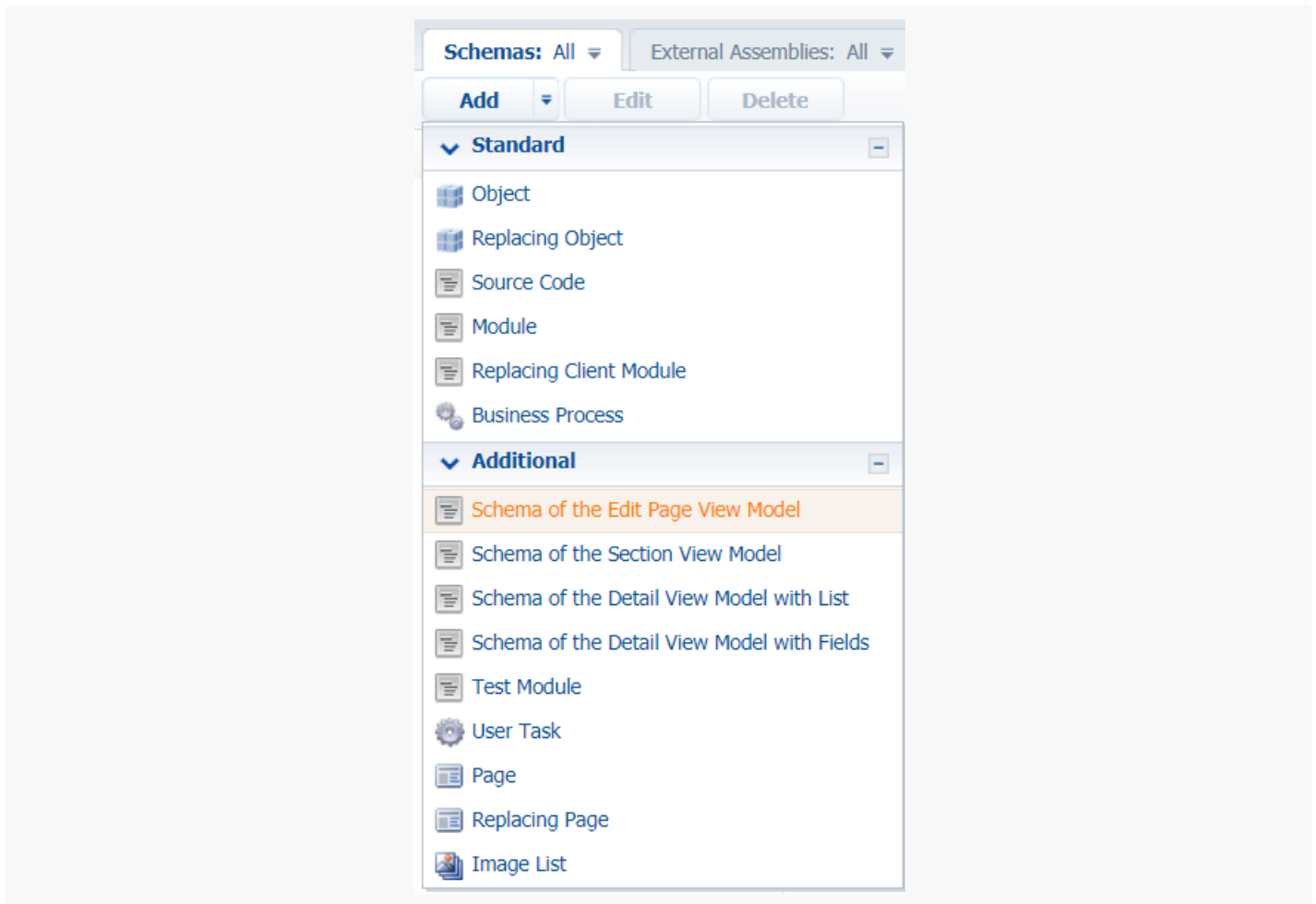
Fig. 1. Setting landing page parameters



2. Add an edit page for the web form

Run the **[Add] -> [Schema of the Edit Page View Model]** menu command on the **[Schemas]** tab in the **[Configuration]** section of the custom package (Fig. 2). The procedure for creating a view model schema of the edit page is covered in the [Create a client schema](#) article.

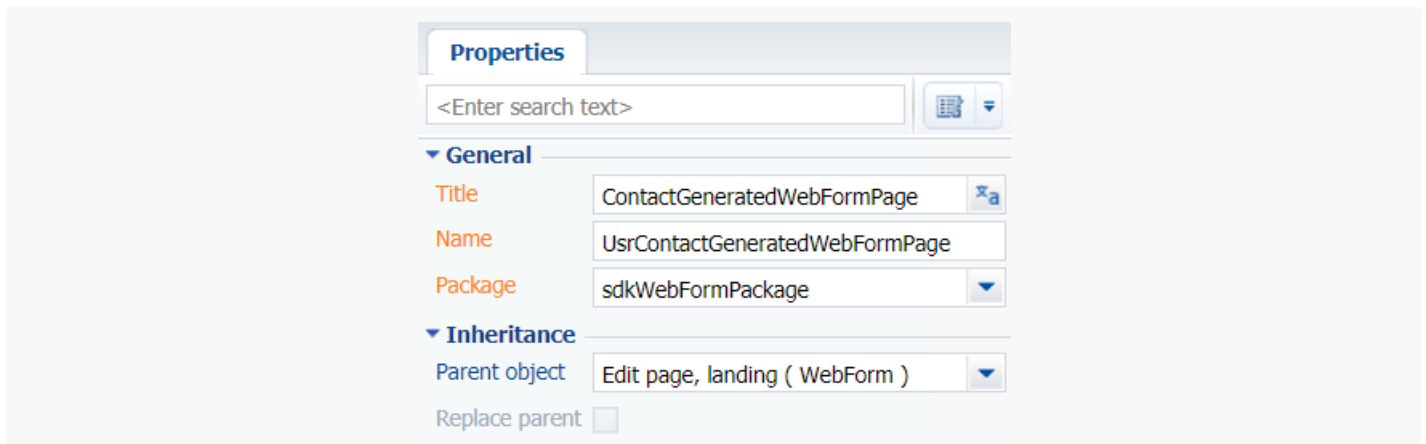
Fig. 2. – Adding a view model schema of the edit page



Specify the following parameters for the created schema of the edit page view model (Fig. 3):

- **[Title]** - "ContactGeneratedWebFormPage"
- **[Name]** - "UsrContactGeneratedWebFormPage"
- **[Parent object]** - "Edit page, landing."

Fig. 3. Setting up the mini-page view model schema



The complete source code of the module is available below:


```

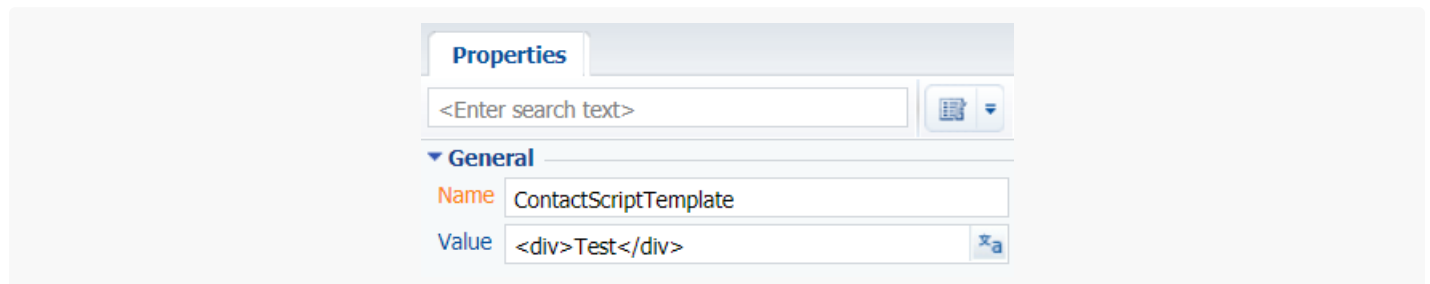
// UsrContactGeneratedWebFormPage - unique schema name.
define("UsrContactGeneratedWebFormPage", ["UsrContactGeneratedWebFormPageResources"],
function() {
return {
details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
methods: {
/**
* @inheritdoc BaseGeneratedWebFormPageV2#getScriptTemplateFromResources
* @overriden
*/
getScriptTemplateFromResources: function() {
var scriptTemplate;
if (this.getIsFeatureEnabled("OutboundCampaign")) {
// ContactScriptTemplate - localizable string name.
scriptTemplate = this.get("Resources.Strings.ContactScriptTemplate");
} else {
scriptTemplate = this.get("Resources.Strings.ScriptTemplate");
}
return scriptTemplate;
}
},
diff: /**SCHEMA_DIFF*/[/**SCHEMA_DIFF*/
];
});

```

Save the schema after making the changes.

Add the *ContactScriptTemplate* localizable string. Specify `<div>Test</div>` as the string value (Fig. 4).

Fig. 4. Setting up a localizable string



Save the schema after making the changes.

3. Map the new landing page type to the created edit page

To do this, add the record to the **[dbo.SysModuleEdit]** DB table. Execute the following SQL script to add the record:

SQL script

MSSQL

```

-- Parameters of new landing page
DECLARE @editPageName nvarchar(250) = N'UsrContactGeneratedWebFormPage'; -- declare the name of
DECLARE @landingTypeName NVARCHAR(250) = N'Contact'; -- the type name of the landing
DECLARE @actionCaption NVARCHAR(250) = N'Contact form'; -- declare the type name for the landing

-- Set system parameters based on new landing page
DECLARE @generatedWebFormEntityUid uniqueidentifier = '41AE7D8D-BEC3-41DF-A6F0-2AB0D08B3967';
DECLARE @cardSchemaUid uniqueidentifier = (select top 1 Uid from SysSchema where Name = @editPageName);
DECLARE @pageCaption nvarchar(250) = (select top 1 Caption from SysSchema where Name = @editPageName);
DECLARE @sysModuleEntityId uniqueidentifier = (select top 1 Id from SysModuleEntity where SysEntitySchemaUid = @generatedWebFormEntityUid);
DECLARE @landingTypeId uniqueidentifier = (SELECT TOP 1 Id FROM LandingType WHERE Name = @landingTypeName);

-- Adding new Landing page variant to application interface
INSERT INTO SysModuleEdit
(Id, SysModuleEntityId, TypeColumnValue, UseModuleDetails, CardSchemaUid, ActionKindCaption, Act
VALUES
(NEWID(), @sysModuleEntityId, @landingTypeId, 1, @cardSchemaUid, @actionCaption, @editPageName,

```

PostgreSQL

```

DO $$
DECLARE
    v_editPageName VARCHAR(250) := N'ContactGeneratedWebFormPageV2';
    v_landingTypeName VARCHAR(250) := N'Contact';
    v_actionCaption VARCHAR(250) := N'Contact form';
    v_pageCaption varchar(250) := (select "Caption" from "SysSchema" where "Name" = v_editPageName);
    v_generatedWebFormEntityUid UUID := '41AE7D8D-BEC3-41DF-A6F0-2AB0D08B3967';
    v_cardSchemaUid UUID := (select "Uid" from "SysSchema" where "Name" = v_editPageName);
    v_sysModuleEntityId UUID := (select "Id" from "SysModuleEntity" where "SysEntitySchemaUid" = v_generatedWebFormEntityUid);
    v_landingTypeId UUID := (SELECT "Id" FROM "LandingType" WHERE "Name" = v_landingTypeName);
BEGIN
    INSERT INTO "SysModuleEdit"
    ("SysModuleEntityId", "TypeColumnValue", "UseModuleDetails", "CardSchemaUid", "ActionKindCaption", "Act
    VALUES
    (v_sysModuleEntityId, v_landingTypeId, true, v_cardSchemaUid, v_actionCaption, v_editPageName);
END;
$$ LANGUAGE plpgsql;

```

Attention. 41AE7D8D-BEC3-41DF-A6F0-2AB0D08B3967 — non-editable identifier of the *GeneratedWebForm* entity schema in the **[dbo.SysSchema]** DB table. The ID is relevant for any case featuring adding a landing page for a custom entity.

Clear the browser cache after running the script. As a result, you will be able to add the new **[Contact form]** landing type in the **[Landing pages and web forms]** section. However, the script that must be added to the source code of the landing page will not be immediately available on the landing record page (Fig. 6).

Fig. 5. The record list of the [Landing pages and web forms] section

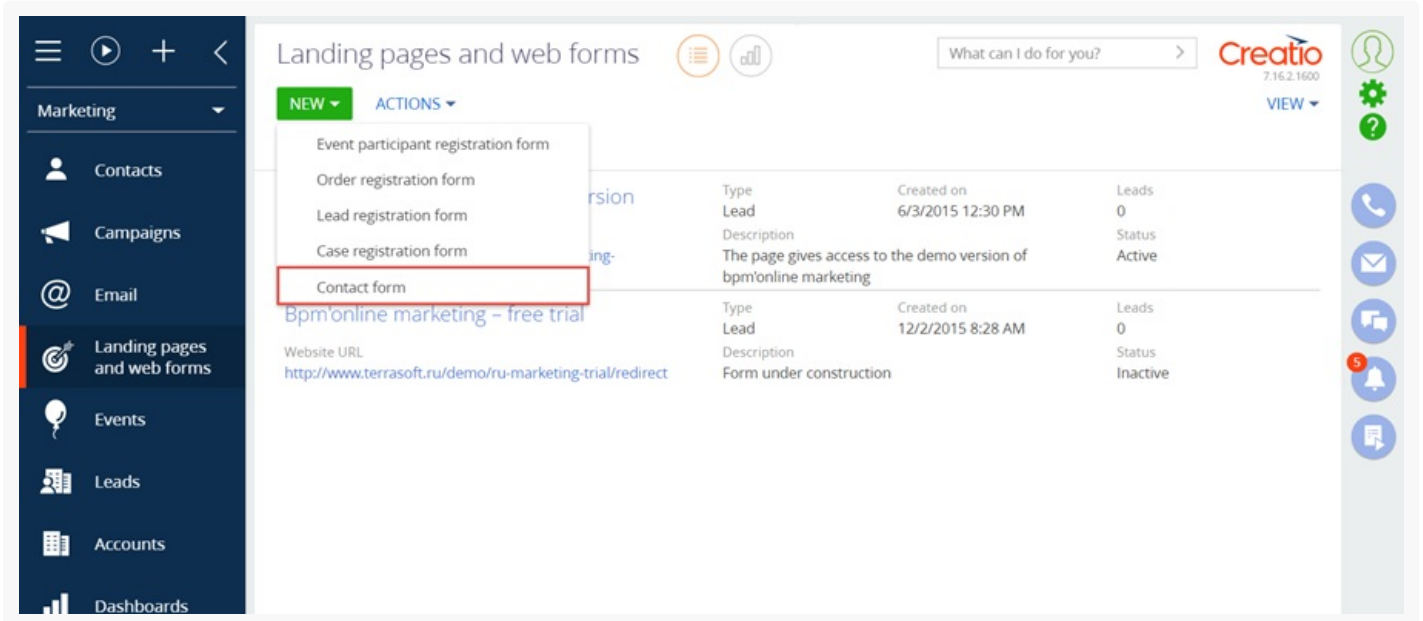
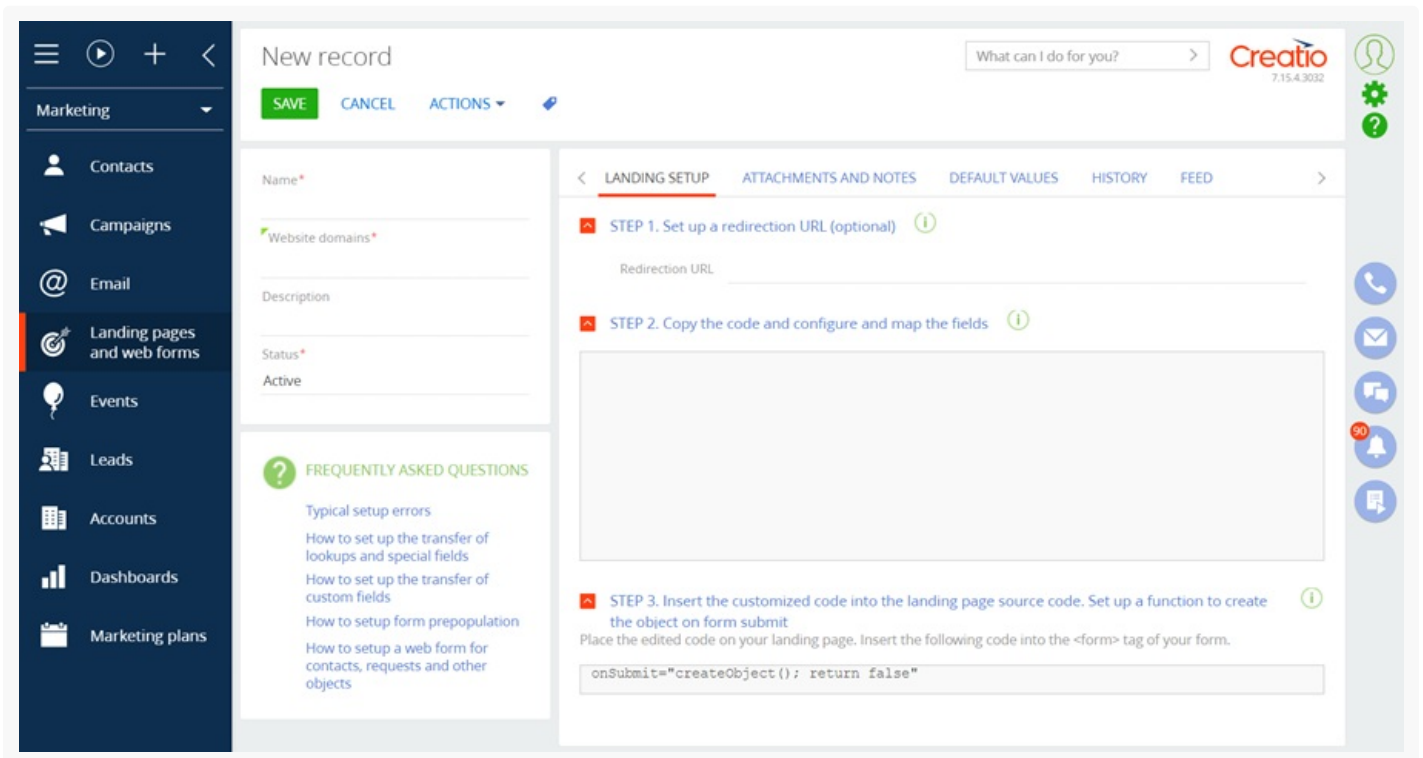


Fig. 6. Landing edit page



4. Update the scripts for the web form record page

The value of the variable contains an escaped HTML string with `<script>` tags and other information for setting up web form field clusters - the columns for the created entity. This value must be localizable. To do so, execute the following SQL script:

```
-- Landing edit page schema name
DECLARE @editPageName nvarchar(250) = N'UsrContactGeneratedWebFormPage'; -- declare the name of

-- region Scripts' structure
DECLARE @sriptPrefix nvarchar(max) = N'<div style="font-family: "Courier New", monospace; font-
DECLARE @sriptDelimiter nvarchar(max) = N'<br>          ';
DECLARE @sriptSuffix nvarchar(max) = N'<br>    },<br>    landingId: ##landingId##,<br>    servi
-- endregion

-- region Scripts' variables
DECLARE @sriptNameColumn nvarchar(max); -- declare column variables to map to the landing
DECLARE @sriptEmailColumn nvarchar(max);
DECLARE @sriptResult nvarchar(max);
-- endregion

-- Adding entity columns.
SET @sriptNameColumn = N'"Name": "css-selector", // Name of a contact';
SET @sriptEmailColumn = N'"Email": "css-selector", // Email';

-- Concat result scripts.
SET @sriptResult = @sriptPrefix + @sriptNameColumn + @sriptDelimiter + @sriptEmailColumn +

-- Set new localizable scripts value for resource with name like '%ScriptTemplate'
UPDATE SysLocalizableValue
SET [Value] = @sriptResult
WHERE SysSchemaId = (SELECT TOP 1 Id FROM SysSchema WHERE [Name] = @editPageName)
and [Key] like '%ScriptTemplate.Value'
```

In the *Adding entity columns* block, add the names of the entity columns to be filled with the values from the web form.

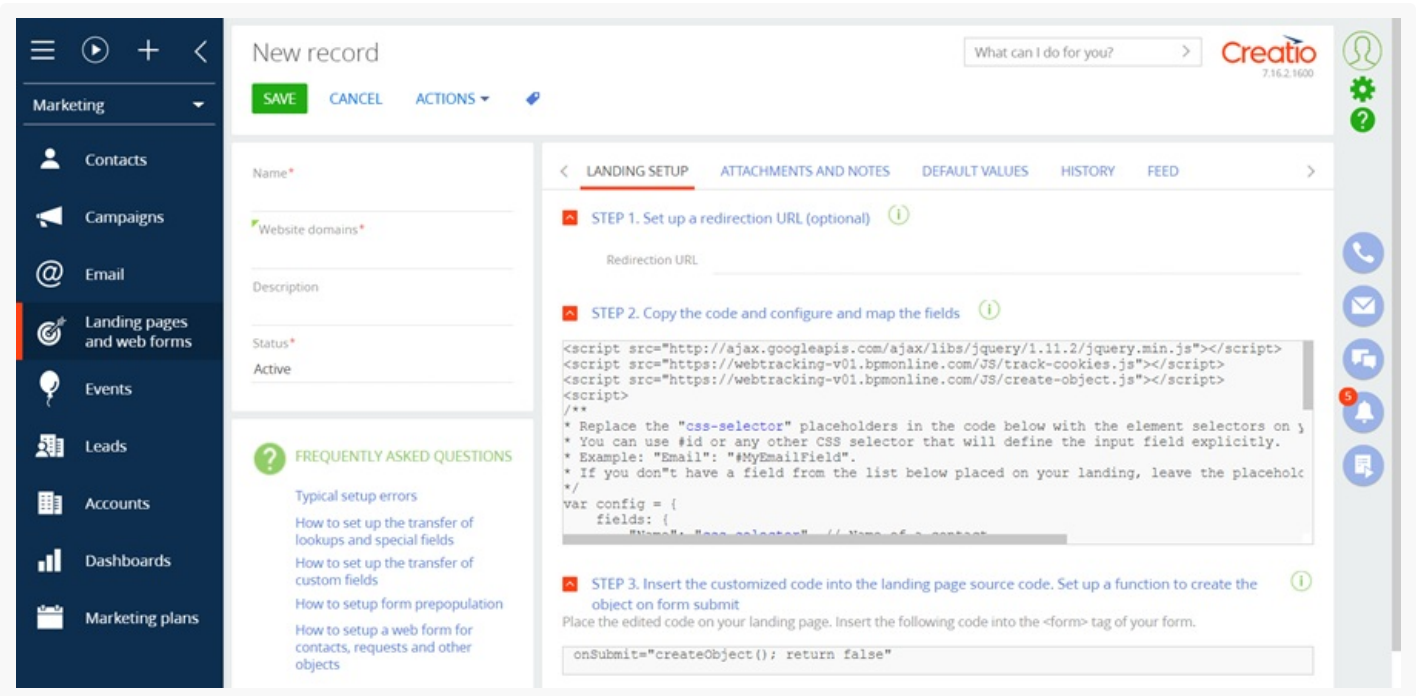
Attention. Replace the double quote (") and space () characters with the `"`; and ` `; HTML character entity references.

Attention. Add the (`@scriptVariableNameColumn`) variable and [concatenate](#) it to `scriptResult` for adding a field.

If the values of other fields (except for *Name* and *Email*) are required after completing the setup, re-run the script from this paragraph after registering all of the required columns, including the existing ones, in the *Adding entity columns* block. When the script is re-run, the settings created earlier are updated.

After the script execution completes, open the schema created in the configuration and re-save it to re-save the resources as well. As a result, when selecting **[Contact form]** in the **[Landing pages and web forms]** section, Creatio will display the landing edit page (Fig. 6) combined with the script to copy to the landing page source code.

Fig. 7. Landing edit page



The script contains the *config* configuration object, which has the following properties defined:

- *fields* - contains the *Name* and *Email* properties. Their values must match the selectors of the *id* attributes of the corresponding field of the web form.
- *landingId* - the landing page ID in the database.
- *serviceUrl* - the URL of the service to which the web form data will be sent.
- *onSuccess* - the handler function to process a successful contact creation. Optional property.
- *onError* - the handler function to process a contact creation error. Optional property.

The configuration to be formed is displayed below.

```
var config = {
  fields: {
    "Name": "css-selector", // Contact name
    "Email": "css-selector", // Email name
  },
  landingId: "b73790ab-acb1-4806-baea-4342a1f3b2a8",
  serviceUrl: "http://localhost:85/0/ServiceModel/GeneratedObjectWebFormService.svc/SaveWebForm",
  redirectUrl: ""
};
```

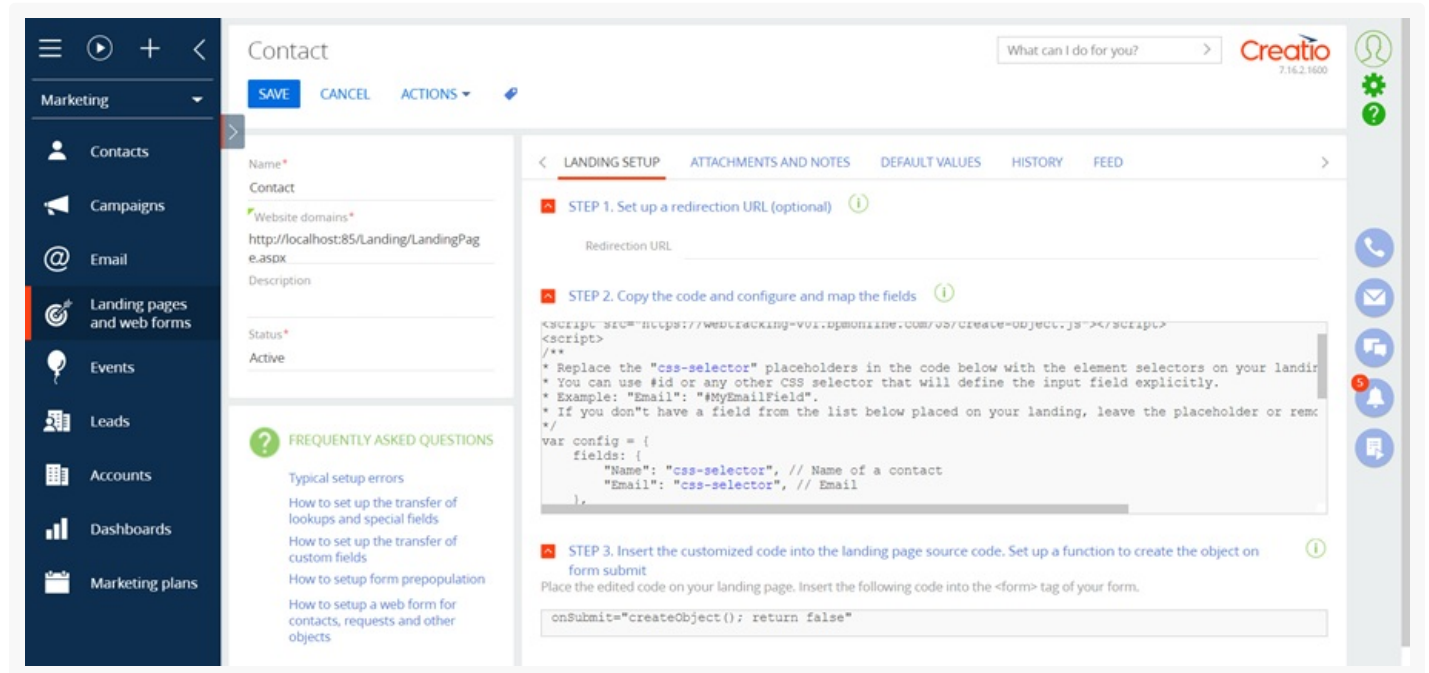
5. Create and configure a landing page in the [Landing pages and web forms] section

To do this, go the **[Landing pages and web forms]** section and click **[Contact form]**.

In the created record, specify (Fig. 8):

- **[Name]** - "Contact"
- **[Website domains]** - "http://localhost:85/Landing/LandingPage.aspx"
- **[Status]** - "Active".

Fig. 8. Landing edit page



Save the page to apply the changes.

6. Deploy and set up a landing page with a web form

Create a landing page with a web form using HTML markup in any text editor. Learn more about creating a landing page and adding the landing script in the ["How to connect your website landing page to Creatio"](#) article.

To register contact data sent via the web form in Creatio, add the following fields (`<input>` HTML element) to the source code of the landing page:

- Contact name.
- Contact email.

Specify the *name* and *id* attributes for each field.

To create a new **[Contact]** object when sending web form data to Creatio, add the JavaScript code to the landing page. Copy the source code from the **[STEP 2. Copy the code and configure and map the fields]** field on the landing record page (fig 8.).

The *config* object is passed as an argument of the *createObject()* function, which is executed upon submitting

the web form.

Ensure that the `createObject()` function is called upon submitting the web form. To do this, add the `onSubmit="createObject(); return false"` attribute to the `<form>` element. You can retrieve the required attribute value from the **[STEP 3. Insert the customized code into the landing page source code. Set up a function to create the object on form submit]** field on the landing edit page (Fig. 8).

The complete source code of the landing page is available below:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"><!--STEP 2--> <!--Copy this part from the STEP 2 field of the landing
  <script src="https://webtracking-v01.bpmonline.com/JS/track-cookies.js"></script>
  <script src="https://webtracking-v01.bpmonline.com/JS/create-object.js"></script>
  <script>

  /**
  * Replace the "css-selector" placeholders in the code below with the element selectors on yc
  * You can use #id or any other CSS selector that will define the input field explicitly.
  * Example: "Email": "#MyEmailField".
  * If you don't have a field from the list below placed on your landing, leave the placeholde
  */
  var config = {
    fields: {
      "Name": "#name-field", // Name of a contact
      "Email": "#email-field", // Email
    },
    landingId: "b73790ab-acb1-4806-baea-4342a1f3b2a8",
    serviceUrl: "http://localhost:85/0/ServiceModel/GeneratedObjectWebFormService.svc/SaveWe
    redirectUrl: "",
    onSuccess: function(response) {
      window.alert(response.resultMessage);
    },
    onError: function(response) {
      window.alert(response.resultMessage);
    }
  };
  /**
  * The function below creates a object from the submitted data.
  * Bind this function call to the "onSubmit" event of the form or any other elements events.
  * Example: <form class="mainForm" name="landingForm" onSubmit="createObject(); return false"
  */
  function createObject() {
    landing.createObjectFromLanding(config)
  }
  /**
  * The function below inits landing page using URL parameters.
  */
```

```

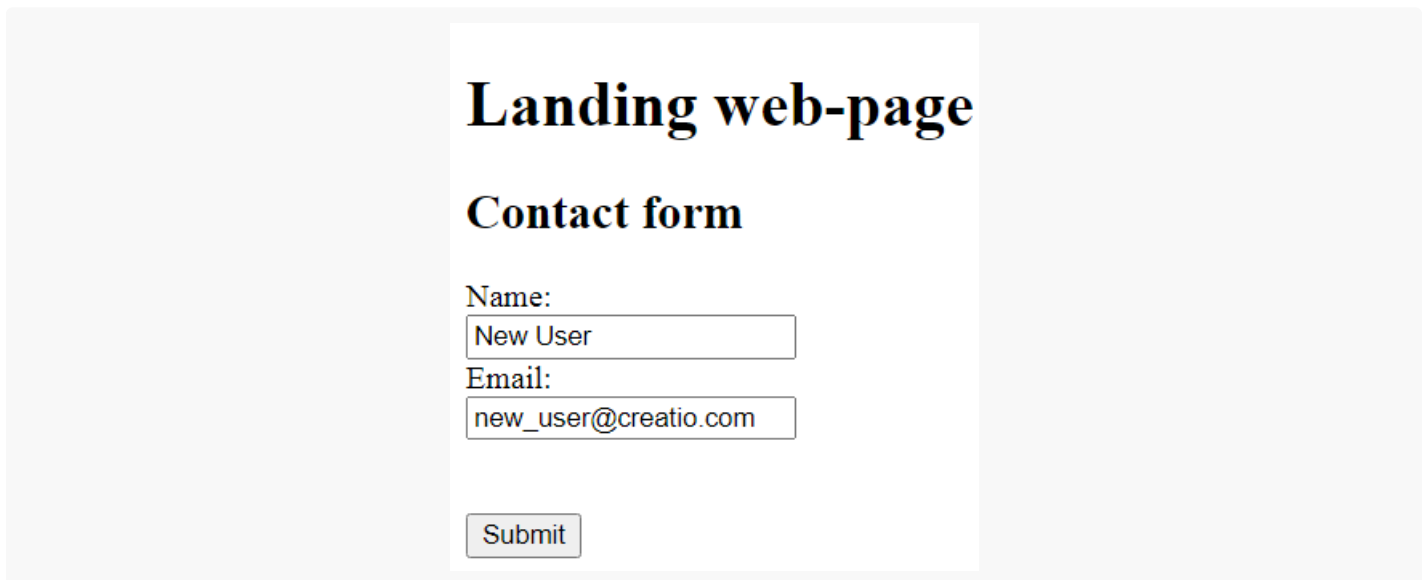
function initLanding() {
    landing.initLanding(config)
}
jQuery(document).ready(initLanding)
</script><!--STEP 2--></head>
<body>
<h1>Landing web-page</h1>
<div>
    <h2>Contact form</h2>
    <form method="POST" class="mainForm" name="landingForm" onSubmit="createObject(); return false">
        <font>
            </font>
        </form>
    </div>
</body>
</html>

```

Open the landing page Specify the values for the created contact (Fig. 9):

- **[Name]** — "New User"
- **[Email]** — "new_user@creatio.com".

Fig. 9. Landing page



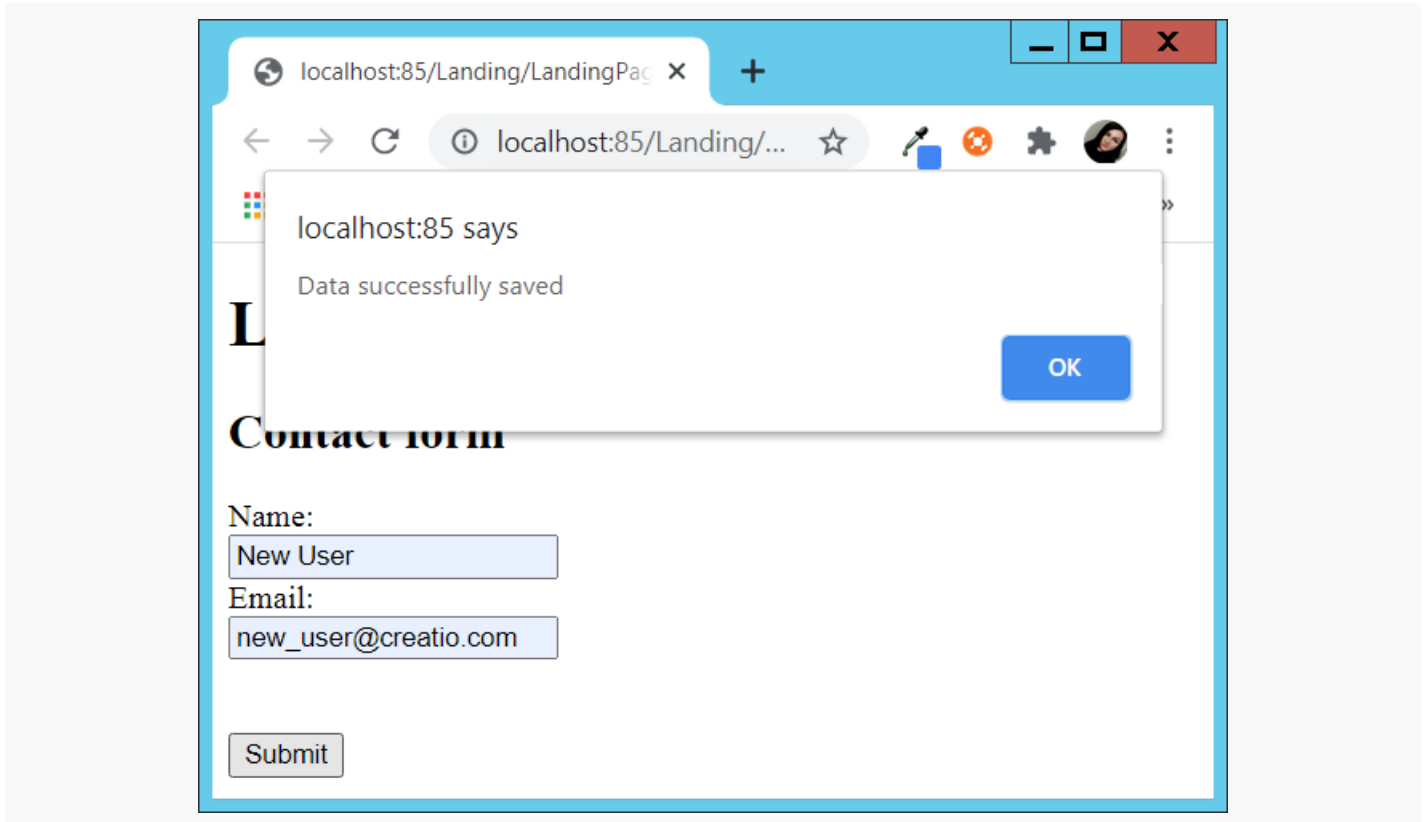
The screenshot shows a web page with a white background and a light gray border. At the top, the text "Landing web-page" is displayed in a large, bold, black serif font. Below this, the text "Contact form" is displayed in a slightly smaller, bold, black serif font. Underneath, there are two input fields. The first is labeled "Name:" and contains the text "New User". The second is labeled "Email:" and contains the text "new_user@creatio.com". At the bottom of the form area, there is a rectangular button with the text "Submit".

Click **[Submit]** to create the contact.

Attention. The contact from the landing page will only be added if the domain of the landing page is specified in the **[Website domains]** field on the landing record page.

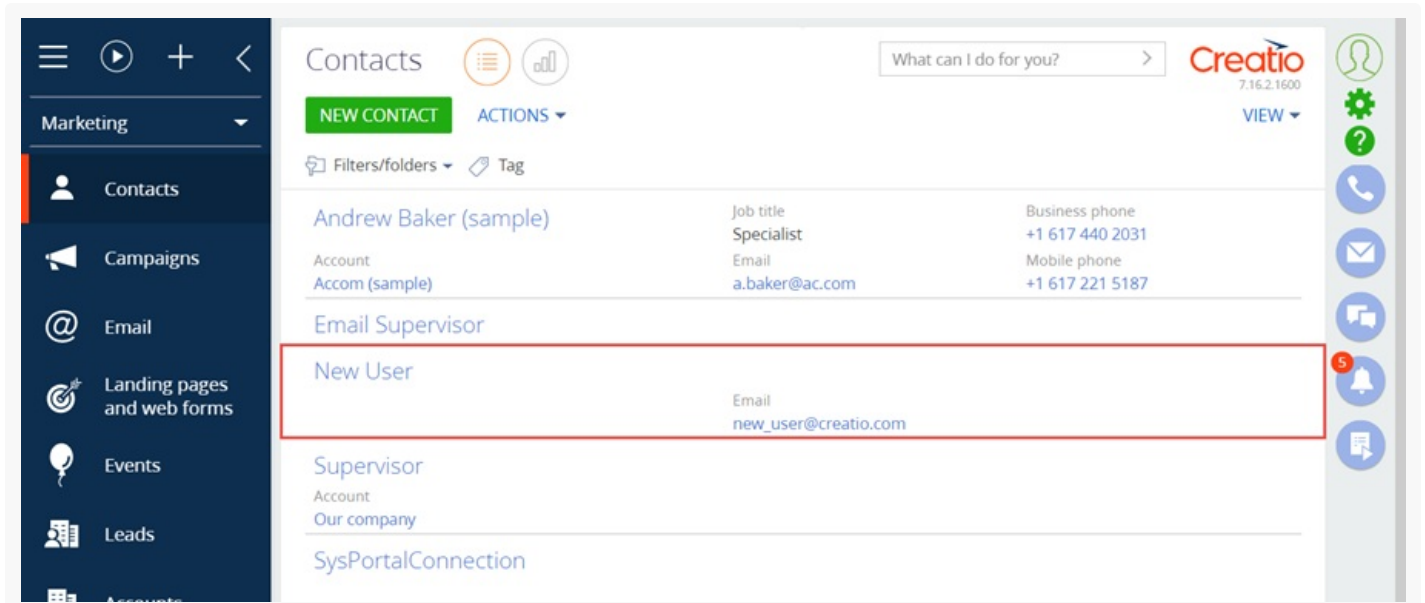
If you place the page on the local server available via the ["localhost" reserved domain name](#) (as specified in the landing page setup, Fig. 8), then the script for creating a contact from the landing page will work correctly (Fig. 10).

Fig. 10. A message about successful data addition



As a result, Creatio will create contact with the specified parameters (Fig. 11).

Fig. 11. Automatically created contact



Implement a handler to create an entity using a web form

 **Advanced**

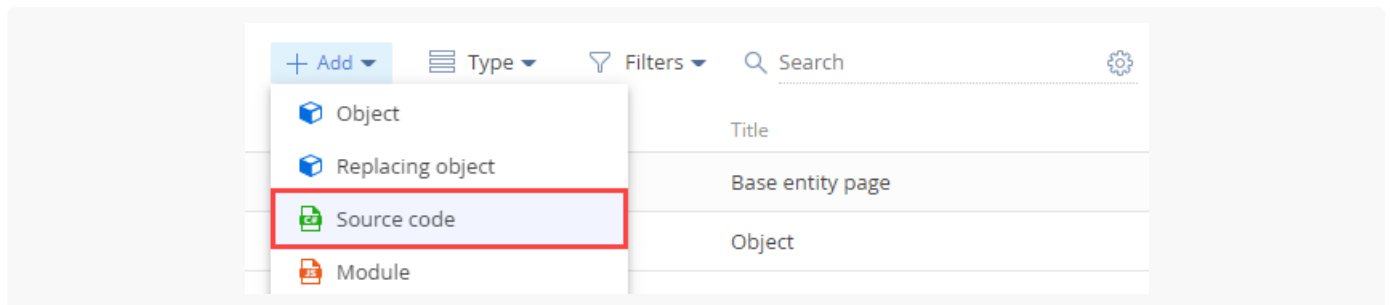
The `Contact` entity contains the `CustomRequiredTextColumn` required text column. When an event participant (`EventTarget`) is created via a web form, Creatio searches for the corresponding contact. By default, if Creatio cannot find the contact, a new contact is created. Saving the contact leads to an error since the `CustomRequiredTextColumn` required field is not populated. To ensure the contact is **saved successfully**, implement a custom handler and call it before an event participant is created.

Example. Implement a custom handler that runs before an event participant is created.

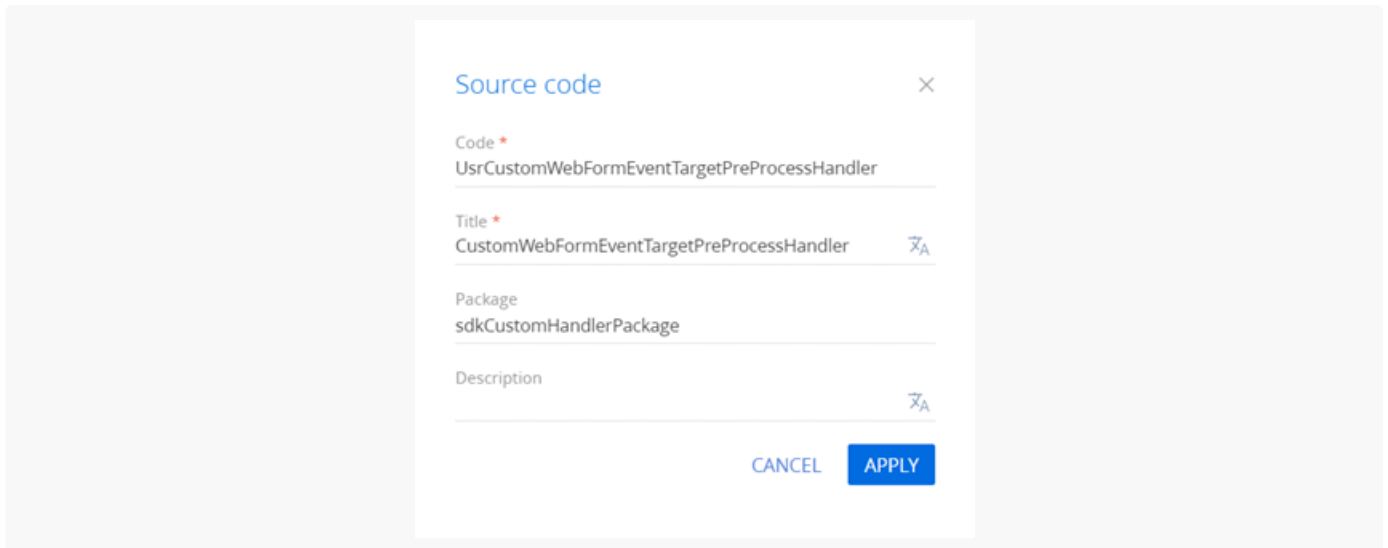
Before you implement the example, set up a web form that creates a custom object. Add a `CustomRequiredTextColumn` required custom field to the web form. To do this, follow the instructions in a separate article: [Set up web form for a custom object](#).

1. Implement a custom handler

1. [Go to the \[Configuration \] section](#) and select a custom [package](#) to add the schema.
2. Click [*Add*] → [*Source code*] on the section list toolbar.



3. Go to the Schema Designer and fill out the schema properties:
 - Set [*Code*] to "UsrCustomWebFormEventTargetPreProcessHandler".
 - Set [*Title*] to "CustomWebFormEventTargetPreProcessHandler".



Click [*Apply*] to apply the properties.

4. Implement a **custom handler that runs before an event participant is created**.
 - a. Add a `Terrasoft.Configuration` namespace in the Schema Designer.
 - b. Add namespaces whose data types to utilize in the class using the `using` directive.
 - c. Add a class name that matches the schema name (the [*Code*] property).
 - d. Specify the `WebFormEventTargetPreProcessHandler` class as a parent class.

View the source code of the `UsrCustomWebFormEventTargetPreProcessHandler` schema of the [*Source code*] type below.

UsrCustomWebFormEventTargetPreProcessHandler

```
namespace Terrasoft.Configuration
{
    using System;
    using System.Linq;
    using Core.Entities;
    using Core;
    using GeneratedWebFormService;

    #region Class: UsrCustomWebFormEventTargetPreProcessHandler
    /// <summary>
    /// Call the custom handler before the event participant is saved.
    /// </summary>
    /// <seealso cref="Terrasoft.Configuration.IGeneratedWebFormPreProcessHandler" />
    public class CustomWebFormEventTargetPreProcessHandler: WebFormEventTargetPreProcessHandler
    {

        #region Properties: Private
        private UserConnection _userConnection { get; set; }
        private FormData _formData { get; set; }
    }
}
```

```

#endregion

#region Methods: Private
private string GetCustomRequiredColumnValue(string customColumnName) {
    var customFormField = this._formData.formFieldsData
        .FirstOrDefault(x => x.name == customColumnName);
    if (customFormField == null) {
        throw new Exception($"There is no required form field {customColumnName}");
    }
    if (string.IsNullOrEmpty(customFormField?.value)) {
        throw new Exception($"Required value is empty for field {customColumnName}");
    }
    return customFormField.value;
}
#endregion

#region Methods: Protected
/// <summary>
/// Create a contact entity whose custom required text column is populated with the f
/// </summary>
/// <param name="contactId">The unique contact ID.</param>
/// <param name="contactNameField">The required contact name field of the form.</para
protected override void CreateContactEntity(Guid contactId, FormFieldsData contactNam
    EntitySchema contactSchema = _userConnection.EntitySchemaManager.GetInstanceByNam
    Entity contact = contactSchema.CreateEntity(_userConnection);
    contact.SetDefColumnValues();
    contact.SetColumnValue("Id", contactId);
    contact.SetColumnValue("Name", contactNameField.value);

    // Set the value of the custom required column.
    var customRequiredColumnName = nameof(Contact.CustomRequiredTextColumn);
    var customRequiredColumnValue = GetCustomRequiredColumnValue(customRequiredColumnn
    contact.SetColumnValue(customRequiredColumnName, customRequiredColumnValue);

    contact.Save(false);
}
#endregion

#region Methods: Public
/// <inheritdoc/>
/// Overload the inherited method so that it initiates the <see cref="UserConnection"
public new FormData Execute(UserConnection userConnection, FormData formData,
    IWebFormImportParamsGenerator paramsGenerator) {
    _userConnection = userConnection;
    _formData = formData;
    return base.Execute(userConnection, formData, paramsGenerator);
}
#endregion

```

```

    }
    #endregion
}

```

5. Click [*Save*] then [*Publish*] on the Designer's toolbar.


2. Register a custom handler in the database

To implement the custom handler, register it in the [WebFormProcessHandlers] database table.

You can register the custom handler in the database in several **ways**:

- using a lookup

To register the custom handler in the database **using a lookup**:

- Click  to open the System Designer.
- Go to the [*System setup*] block → [*Lookups*].
- Add a new handler record to the [*Web form process handlers*] entity lookup. By default, this lookup is absent from the index of Creatio lookups. To add the [*Web form process handlers*] lookup to Creatio, create a lookup and select the [*Web form process handlers*] object as the lookup object.
- Fill out the **lookup fields**:
 - Set [*Entity name*] to "EventTarget".
 - Set [*FullClassName*] to "Terrasoft.Configuration.CustomWebFormEventTargetPreProcessHandler, Terrasoft.Configuration".
 - Select the [*Is active*] checkbox.
- using an SQL query

To register the custom handler in the database **using an SQL query**, execute the following SQL query.

SQL query

```

INSERT INTO WebFormProcessHandlers (Id, EntityName, FullClassName, IsActive)
VALUES (NEWID(), N'EventTarget', 'Terrasoft.Configuration.CustomWebFormEventTargetPreProc


```

Since the schema inherits from the out-of-the-box handler and the custom schema calls the base logic, you must disable the out-of-the-box handler.

You can disable the out-of-the-box handler in the following **ways**:

- using a lookup

To disable the out-of-the-box handler **using a lookup**:

- Click  to open the System Designer.
- Go to the [*System setup*] block → [*Lookups*].
- Open the lookup and clear the [*Is active*] checkbox for the [*EventTarget*] entity that has the "Terrasoft.Configuration.CustomWebFormEventTargetPreProcessHandler, Terrasoft.Configuration"

value in the [*FullClassName*] field.

- using an SQL query

To disable the out-of-the-box handler **using an SQL query**, execute the following SQL query.

SQL query

```
UPDATE WebFormProcessHandlers
SET IsActive = 0
WHERE FullClassName = 'Terrasoft.Configuration.WebFormEventTargetPreProcessHandler, Terrasoft
```

The outcome of the example

To **view the outcome of the example**, restart the application pool.

As a result, Creatio will add a new contact when a form is submitted with required fields populated, including the `CustomRequiredTextColumn` field.

Web-to-Case



Advanced

Web-to-Case functionality implements the ability to create cases in the Creatio by filling the required form fields embedded in a third-party site - landing.

The `ProductCore` package depends on the `WebForms` package, that contains Web-to-Case functionality. This means that landings can be used in all products. Pre-configured base functionality is implemented in the service enterprise, customer center, marketing products and all bundles that these products are part of.

More information about landings can be found in the [\[Landings \] section](#) articles of the corresponding products (such as Marketing Creatio).

Web-to-Case configuration can be done in the system interface. To implement generated JavaScript to a third-party site, you need the basic Web development skills.

The Web-to-Case base functionality allows to configure the following features without programming (using minor improvements on a third-party site):

- The form interface and styles.
- List of the additionally passed fields.
- List of default values for the fields that are not displayed in the form.
- The list of domains from which the case registration for each landing will be possible.
- The address to which the user will be redirected after submitting the form.
- JavaScript event handlers of successful/unsuccessful case registration.
- Additional landings, that can be configured in different way. That makes it possible to distinguish cases created from different sites.

You can modify the project to set up a preliminary handler of case registration through the Web-to-Case with the

data validation, correction, creation of related entities and etc. The automatic creation of contact for the registered case is configured in the Creatio base configuration in the handler of case registration through the Web form.

The logic of the automatic filling of case fields.

In the process of case registration through the Web form, the following fields are recommended for filling: [*Name*], [*Email*], [*Phone*], [*Case subject*]. The [*Case subject*] value will be passed to the new case.

The Creatio will identify the contact by [*Name*], [*Email*] and [*Phone*] fields. The search is performed in a following way:

1. If contact fields matches the [Name], [Email] and [Phone] fields from the filled form, they will be added to the created case.
2. If contact fields matches only the [Name] and [Email] fields from the filled form, they will be added to the created case.
3. If contact fields matches only the [Email] field from the filled form, it will be added to the created case.
4. Otherwise, a new contact is created and the [Name], [Email] and [Phone] fields will be filled in. The created contact is added to the registered case.

If more than one contact are found, then the first contact will be used as contact of the case. Also the case registration date (`RegisteredOn` column) will be automatically filled with the current date and time.

Recommendations for the execution of project solutions

If you need to customize the Web-to-Case, use its base functionality as an example.

To execute the project solution:

1. Create a page schema that is inherited from the `CaseGeneratedWebFormPageV2`. The page should not be a replacement page.
2. Add a record of the new type of landing to the `LandingType` table and localization to the `SysLandingTypeLcz` table.
3. Register the typed page created in the first step (the value of the type is new).
4. If you need preliminary processing of the form data before saving the record in the database, you need to create a class that implements the `IGeneratedWebFormPreProcessHandler` interface. This class is a preliminary handler for case registration. Implement the `Execute()` method. This method is the entry point to the handler. Additional actions are implemented in this method. You can take the `WebFormCasePreProcessHandler` schema as an example.
5. If you need to perform actions after saving the record in the database, you need to create a class that implements the `IGeneratedWebFormPreProcessHandler` interface. This class is a preliminary handler for case registration. Implement the `Execute()` method and perform necessary actions.
6. If you created the registration handlers of the case, register them in the `WebFormProcessHandlers` table. Use an existing record as an example of registration.
7. Edit the script template that forms the configuration JavaScript object of the landing, and place it in the `ScriptTemplate` localized string of the created page. Specify the similar script for all localizations used. You can

find an example of the script in the `CaseGeneratedWebFormPageV2` schema.

8. Bind all created data to the package.

Create Web-to-Case landing pages

 Advanced

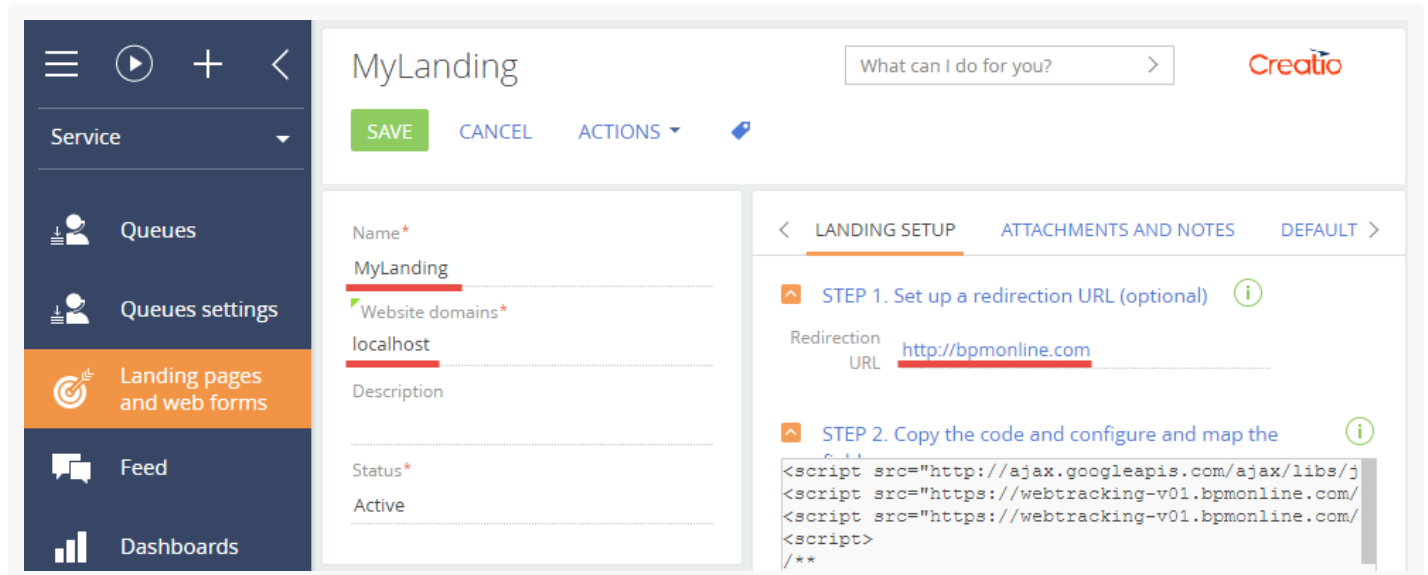
Steps to create Web-to-Case landing

1. Create new landing record in the Creatio

To create a new landing record, execute the [*Add*] action in the [*Landing pages and web forms*] section. Fill in the following fields on the opened page (Fig. 1):

- [Name] – landing page name in Creatio.
- [Website domains] – your landing page URL.
- [Status] – landing status.
- [Redirection URL] – the URL that is opened after the landing page form is completed.

Fig. 1 Landing edit page



The screenshot shows the 'MyLanding' edit page in Creatio. The interface includes a top navigation bar with a search box and the Creatio logo. Below the title, there are 'SAVE', 'CANCEL', and 'ACTIONS' buttons. The form fields are as follows:

- Name***: MyLanding
- Website domains***: localhost
- Description**: (empty)
- Status***: Active
- Redirection URL**: http://bpmonline.com

The right-hand side of the page displays the 'LANDING SETUP' section with two steps:

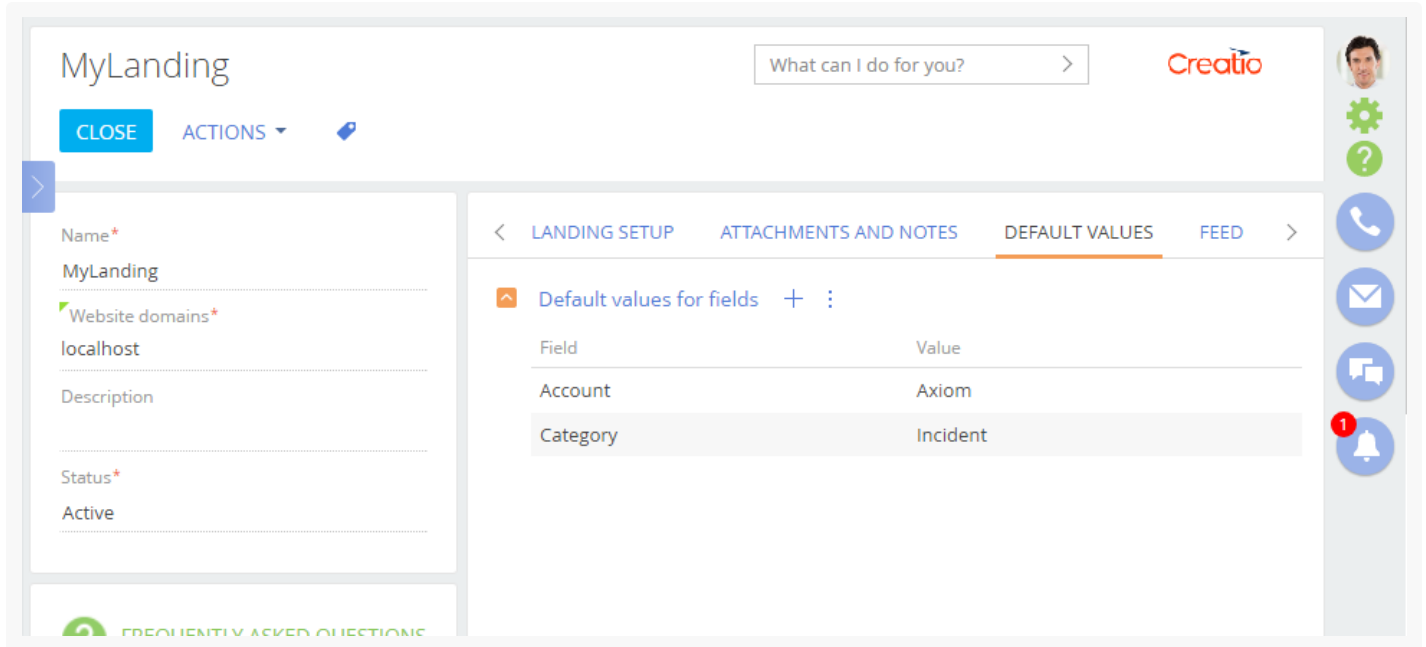
- STEP 1. Set up a redirection URL (optional)**: This step is completed, with the URL 'http://bpmonline.com' entered.
- STEP 2. Copy the code and configure and map the**: This step is partially completed. It shows a code block with the following content:


```
<script src="http://ajax.googleapis.com/ajax/libs/j
<script src="https://webtracking-v01.bpmonline.com/
<script src="https://webtracking-v01.bpmonline.com/
<script>
/**
```

Attention.

When creating a case, you can receive only four fields ("Subject", "Email", "Name" and "Phone") from the landing page. Therefore, you must set the default values for the new landing record (Fig. 2).

Fig. 2 Values by default



Save the page to apply the changes.

2. Create a landing page

To create landing page, you need to create a standard HTML page containing a Web form in any text editor using HTML markup.

To register the data sent via the web-form, add four fields to the form (using `<input>` element) that define the case:

- Case subject
- Contact email
- Contact name
- Contact phone

Specify the `name` and `id` attributes for each field.

To send a form data to Creatio when creating a new [Case] object, you need to add a JavaScript script to the HTML page. Copy the script source code from the [*STEP 2. Copy the code and configure and map the fields*] field of the landing edit page (Fig. 1).

Note.

The script must be copied from the already saved landing.

The script contains the `config` configuration object that has following properties:

- `fields` – contains the object with "Subject", "Email", "Name" and "Phone" values that must match the `id` attribute selectors of the corresponding web form fields.

- `landingId` - contains the landing Id in the database.
- `serviceUrl` - contains URL of the service to which the form data will be sent.
- `redirectUrl` - contains redirection URL specified in the [Redirection URL] field of the landing.
- `onSuccess` - contains a function that handles the successful creation of a case. Optional property.
- `onError` - contains a function that handles the error of the case creation. Optional property.

The `config` configuration object is passed as an argument of the `createObject()` function that must be executed when the form is submitted.

To call the `createObject()` function when sending a form, add the `onSubmit = "createObject(); return false"` attribute to the

tag of the HTML page of the Landing page (see STEP 3, Fig. 1).

An example of the complete landing page source code for the case registration:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <!--STEP 2-->
  <!--This part needs to be copied from the STEP 2 field of the landing edit page-->
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
  <script src="https://webtracking-v01.creatio.com/JS/track-cookies.js"></script>
  <script src="https://webtracking-v01.creatio.com/JS/create-object.js"></script>
  <script>
    /**
     * Replace the "css-selector" placeholders in the code below with the element selectors
     * You can use #id or any other CSS selector that will define the input field explicitly
     * Example: "Email": "#MyEmailField".
     * If you don't have a field from the list below placed on your landing, leave the place
     */
    var config = {
      fields: {
        "Subject": "#subject-field", // Case subject
        "Email": "#email-field", // Visitor's email
        "Name": "#name-field", // Visitor's name code
        "Phone": "#phone-field", // Visitor's phone number
      },
      landingId: "8ab71187-0428-4372-b81c-fd05b141a2e7",
      serviceUrl: "http://localhost/creatioservice710/0/ServiceModel/GeneratedObjectWebFor
      redirectUrl: "http://creatio.com",
      onSuccess: function(response) {
        window.alert(response.resultMessage);
      },
      onError: function(response) {
```

```

        window.alert(response.resultMessage);
    }

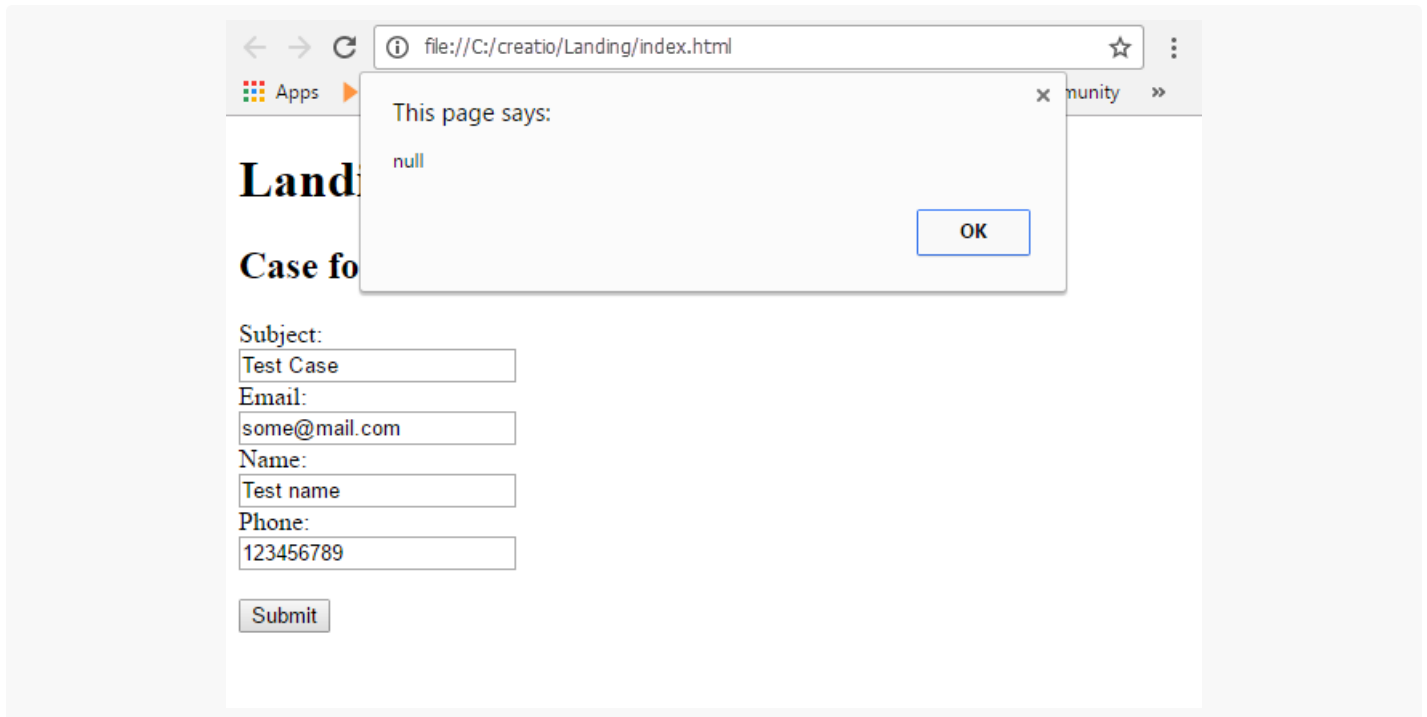
};
/**
 * The function below creates a object from the submitted data.
 * Bind this function call to the "onSubmit" event of the form or any other elements eve
 * Example: <form class="mainForm" name="landingForm" onSubmit="createObject(); return f
 */
function createObject() {
    landing.createObjectFromLanding(config)
}
</script>
<!--STEP 2-->
</head>
<body>
<h1>Landing web-page</h1>
<div>
    <h2>Case form</h2>
    <form class="mainForm" name="landingForm" onSubmit="createObject(); return false">
        Subject:<br>
        <input type="text" name="subject" id="subject-field"><br>
        Email:<br>
        <input type="text" name="Email" id="email-field"><br>
        Name:<br>
        <input type="text" name="Name" id="name-field"><br>
        Phone:<br>
        <input type="text" name="Phone" id="phone-field"><br><br>
        <input type="submit" value="Submit">
    </font>
    </form>
</div>
</body>
</html>

```

3. Add the page to the website.

A case from the landing page will be added to the Creatio only if the page is hosted on the site whose name is listed in the [*Website domains*] field of the landing page record in Creatio. If you open the page in the browser locally, then an empty message will be displayed when the case is created.

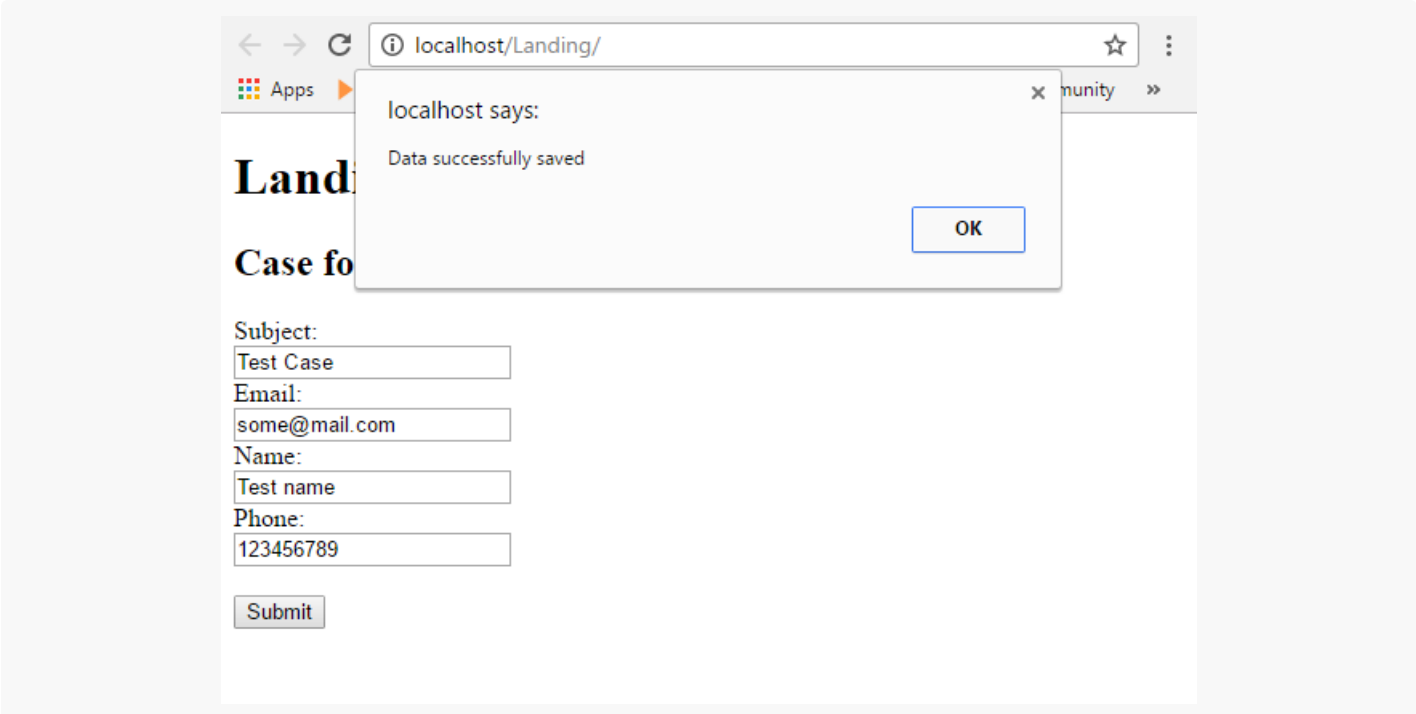
Fig. 3 Empty message

**Note.**

The output of an empty message is configured in the `onError()` method of the configuration object.

If you place the page on the local server of the computer that serves as the [reserved domain name localhost](#) (as specified in the landing setting, Fig. 1), then the script that adds the address from the web page of the landing will work correctly (Fig. 4)

Fig. 4 The correct adding of data



As a result, a case with specified parameters will be automatically created.

Fig. 5 Automatically created case

