Creatio Academy

Business process service

Service that runs business processes

Version 7.16

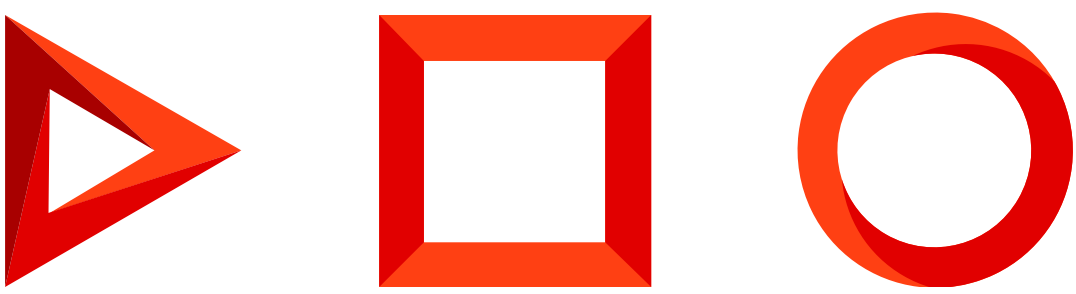


Table of Contents

Service that runs business processes

Advanced

`ProcessEngineService.svc` is a web service implemented in the Creatio service model to run business processes. Use the web service to integrate external apps with Creatio.

Run a business process from an external app

The `ProcessEngineService.svc` web service grants access to Creatio objects.

`ProcessEngineService.svc` service URL

```
https://mycreatio.com/0/ServiceModel/ProcessEngineService.svc
```

The `Terrasoft.Core.Service.Model.ProcessEngineService` class implements the `ProcessEngineService.svc` functionality. The primary **methods** of the `ProcessEngineService.svc` service are as follows:

- `Execute()`. Runs a business process. Passes a set of incoming business process parameters and returns the process execution result.
- `ExecProcElByUId()`. Executes a specific business process element. Can only execute an element of a running process.

Note. View the entire index of the web service methods in the .NET class library.

Run a business process from the front-end

You can run a business process from the front-end in the following **ways**:

- `executeProcess()` method of the `PrcessModuleUtilities` module
- `execute()` method of the `Terrasoft.RunProcessRequest` class

Run a business process using the `executeProcess()` method

To **run a business process from the front-end**, use the `executeProcess()` method of the `ProcessModuleUtilities` module in the `NUI` package. This module provides a convenient interface for executing requests to the `ProcessEngineService.svc` service.

To **run a business process from a client module schema**:

1. Add the `ProcessModuleUtilities` module as a dependency to the module of the page that calls the service.

2. Call the `executeProcess(args)` method of the `ProcessModuleUtilities` module. Set the `args` object as a parameter.
   The **properties** of the `args` object are as follows:

   - `sysProcessName`. The name of the called process (optional if the `sysProcessId` property is used).
   - `sysProcessId`. The ID of the called process (optional if the `sysProcessName` property is used).
   - `parameters`. The object whose properties are the incoming parameters of the called process.

Run a business process using the `execute()` method

1. Create the `RunProcessRequest` class. Set the properties of the process to run and the needed resulting parameters in this class.

   - `schemaName`. The name of the called process (the [*Code*] field of the Process Designer).
   - `schemaUId`. The ID of the called process.
   - `Parameters`. The object whose properties are the incoming parameters of the called process.
   - `resultParameterNames`. The array of parameters whose values must be received when executing the process.

2. Call the `execute()` method of the `RunProcessRequest` class.

Run a business process from the back-end

To **run a business process from the back-end**, call the `Execute()` method of the `Terrasoft.Core.Process.IProcessExecutor` interface. The `IProcessExecutor` interface provides a set of the `Execute()` method overloads for solving custom tasks. View the description of the `IProcessExecutor` interface in the .NET class library.

Run a business process without passing or receiving parameters

View the **ways** to run a business process without passing or receiving parameters in the table below.

Ways to run a business process without passing or receiving parameters

Way	Method	Method parameters
Name of the business process schema	`Execute(string processSchemaName)`	`processSchemaName`. The name of the business process schema.
ID of the business process schema	`Execute(Guid processSchemaUId)`	`processSchemaUId`. The ID of the business process schema.

Example that runs a business process by the name of the business process schema

```
UserConnection userConnection = Get<UserConnection>("UserConnection");
```

```
IProcessEngine processEngine = userConnection.ProcessEngine;
IProcessExecutor processExecutor = processEngine.ProcessExecutor;
processExecutor.Execute("UsrProcess2Custom1");
return true;
```

Run a business process with passing incoming parameters

View the **ways** to run a business process with passing incoming parameters in the table below.

Ways to run a business process with passing incoming parameters

Way	Method	Method parameters
Name of the business process schema	`Execute(string processSchemaName, IReadOnlyDictionary<string, string> parameterValues)`	`processSchemaName`. The name of the business process schema. `parameterValues`. The key-value collection of incoming parameters. In the parameter collection, the key is the name of the business process parameter, the value is the parameter value converted to a string type.
ID of the business process schema	`Execute(Guid processSchemaUId, IReadOnlyDictionary<string, string> parameterValues)`	`processSchemaUId`. The ID of the business process schema. `parameterValues`. The key-value collection of incoming parameters.

View the examples that run a business process below.

Transfer an integer parameter value

```
UserConnection userConnection = Get<UserConnection>("UserConnection");
IProcessEngine processEngine = userConnection.ProcessEngine;
IProcessExecutor processExecutor = processEngine.ProcessExecutor;
var nameValues = new Dictionary<string, string>();
int parameter1Value = 100;
nameValues["parameter1"] = parameter1Value.ToString();
processExecutor.Execute("UsrProcess3Custom1", nameValues);
return true;
```

Transfer a parameter value of the value collection type

```
UserConnection userConnection = Get<UserConnection>("UserConnection");
IProcessEngine processEngine = userConnection.ProcessEngine;
```

```
IProcessExecutor processExecutor = processEngine.ProcessExecutor;
ObjectList<int> values = ObjectList.Create(1, 2, 3, 4);
string serializedValue = BaseSerializableObjectUtilities.SerializeToJson(values);
var parameterNameValues = new Dictionary<string, string>();
parameterNameValues["InputValueList"] = serializedValue;
processExecutor.Execute("ProcessRunsFromScriptTask", parameterNameValues);
return true;
```

Transfer a parameter value of the object collection type with attributes

```
UserConnection userConnection = Get<UserConnection>("UserConnection");
IProcessEngine processEngine = userConnection.ProcessEngine;
IProcessExecutor processExecutor = processEngine.ProcessExecutor;
/* Creates a collection of objects. */
var list = new CompositeObjectList<CompositeObject>();
ar item1 = new CompositeObject();
/* The process parameter object contains [Id] and [Name] fields. */
item1["Id"] = new Guid("94cc536a-71a7-4bfb-87ca-13f53b23c28e");
item1["Name"] = "Name1"; list.Add(item1);
var item2 = new CompositeObject(); item2["Id"] = new Guid("e694d36e-1727-4276-9fbf-b9aa193e4f44"
item2["Name"] = "Name2";
list.Add(item2);
string serializedValue = BaseSerializableObjectUtilities.SerializeToJson(list);
var parameterNameValues = new Dictionary<string, string>();
parameterNameValues["InputObjectList"] = serializedValue;
processExecutor.Execute("ProcessRunsFromScriptTask", parameterNameValues);
return true;
```

Run a business process using a received single outgoing parameter

View **ways** to run a business process using a received single outgoing parameter in the table below.

Run a business process using a received single outgoing parameter

Way	Method	Method parameters
Name of the business process schema	`Execute<TResult>(string processSchemaName, string resultParameterName)`	`processSchemaName`. The name of the business process schema. `resultParameterName`. The name of the outgoing parameter.
Name of the business process schema with passing incoming parameters	`Execute<TResult>(string processSchemaName, string resultParameterName, IReadOnlyDictionary<string, string> parameterValues)`	`processSchemaName`. The name of the business process schema. `resultParameterName`. The name of the outgoing parameter. `parameterValues`. The key-value collection of incoming parameters.
ID of the business process schema	`Execute<TResult>(Guid processSchemaUId, string resultParameterName)`	`processSchemaUId`. The ID of the business process schema. `resultParameterName`. The name of the outgoing parameter.
ID of the business process schema with passing incoming parameters	`Execute<TResult>(Guid processSchemaUId, string resultParameterName, IReadOnlyDictionary<string, string> parameterValues)`	`processSchemaUId`. The ID of the business process schema. `resultParameterName`. The name of the outgoing parameter. `parameterValues`. The key-value collection of incoming parameters.

View the examples that run a business process below.

Run a business process using a received single outgoing parameter

```
UserConnection userConnection = GetUserConnection();
IProcessExecutor processExecutor = userConnection.ProcessEngine.ProcessExecutor;
/* List of incoming parameters. */
var inputParameters = new Dictionary<string, string> {
    ["ProcessSchemaParameter1"] = "Value1",
    ["ProcessSchemaParameter2"] = "Value2"
};
string processSchemaName = "processSchemaName";
Guid processSchemaUId = Guid.Parse("00000000-0000-0000-0000-000000000000");
/* Runs the process via the schema name. Returns the text value of the [ProcessSchemaParameter3]
string resultValue = processExecutor.Execute<string>(processSchemaName, "ProcessSchemaParameter3
/* Runs the process via the schema ID. Returns the text value of the [ProcessSchemaParameter3] p
string resultValue = processExecutor.Execute<string>(processSchemaName, "ProcessSchemaParameter3
/* Runs the process via the schema name using transferred parameters. Returns the text value of
string resultValue = processExecutor.Execute<string>(processSchemaName, "ProcessSchemaParameter3
/* Runs the process via the schema ID using transferred parameters. Returns the text value of th
string resultValue = processExecutor.Execute<string>(processSchemaUId, "ProcessSchemaParameter3"
```

Run a business process with receiving multiple outgoing parameters

View the **ways** to run a business process with receiving multiple outgoing parameters in the table below.

Run a business process with receiving multiple outgoing parameters

Way	Method	Method parameters
Name of the business process schema	`Execute(string processSchemaName, IEnumerable<string> resultParameterNames)`	`processSchemaName`. The name of the business process schema. `resultParameterNames`. The collection of outgoing parameters.
Name of the business process schema with passing incoming parameters	`Execute(string processSchemaName, IReadOnlyDictionary<string, string> parameterValues, IEnumerable<string> resultParameterNames)`	`processSchemaName`. The name of the business process schema. `parameterValues`. The key-value collection of incoming parameters. `resultParameterNames`. The collection of outgoing parameters.
ID of the business process schema	`Execute(Guid processSchemaUId, IEnumerable<string> resultParameterNames)`	`processSchemaUId`. The ID of the business process schema. `resultParameterNames`. The collection of outgoing parameters.
ID of the business process schema with passing incoming parameters	`Execute(Guid processSchemaUId, IReadOnlyDictionary<string, string> parameterValues, IEnumerable<string> resultParameterNames)`	`processSchemaUId`. The ID of the business process schema. `parameterValues`. The key-value collection of incoming parameters. `resultParameterNames`. The collection of outgoing parameters.

View the examples that run a business process below.

Example that runs a business process with receiving outgoing parameters

```
UserConnection userConnection = GetUserConnection();
IProcessExecutor processExecutor = userConnection.ProcessEngine.ProcessExecutor;
/* Collection of incoming parameters.*/
var inputParameters = new Dictionary<string, string> {
    ["ProcessSchemaParameter1"] = "Value1",
    ["ProcessSchemaParameter2"] = "Value2"
};
/* Collection of outgoing parameters.*/
var resultParameterNames = new string[] {
    "ProcessSchemaParameter3",
    "ProcessSchemaParameter4"
};
string processSchemaName = "processSchemaName";
Guid processSchemaUId = Guid.Parse("00000000-0000-0000-0000-000000000000");
/* Runs the process via the schema name using received outgoing parameters.*/
ProcessDescriptor processDescriptor = processExecutor.Execute(processSchemaName, resultParameter
/* Runs the process via the schema ID using received outgoing parameters.*/
ProcessDescriptor processDescriptor = processExecutor.Execute(processSchemaName, resultParameter
/* Runs the process via the schema name using transferred incoming and received outgoing paramet
ProcessDescriptor processDescriptor = processExecutor.Execute(processSchemaName, inputParameters
/* Runs the process via the schema ID using transferred incoming and received outgoing parameter
ProcessDescriptor processDescriptor = processExecutor.Execute(processSchemaUId, inputParameters,
```

Run a business process using a received collection of outgoing parameters returns an object of the `ProcessDescriptor` type. To **receive the values of outgoing parameters**, use the `ResultParameterValues` property of the `IReadOnlyDictionary<string, object>` type in the `Terrasoft.Core.Process.ProcessDescriptor` class. View the description of the `ProcessDescriptor` class in the .NET class library.

Example that receives the outgoing parameter values

```
ProcessDescriptor processDescriptor = processExecutor.Execute("processSchemaName", inputParamete
object parameter3Value = processDescriptor.ResultParameterValues["ProcessSchemaParameter3"];
if (processDescriptor.ResultParameterValues.TryGetValue("ProcessSchemaParameter4", out object pa
    Console.Log(parameter4value);
}
```

Attention. Regardless of the settings, you cannot run the business process with receiving outgoing parameters is the background.

Run the business process using a web

Run the business process using a web service

Advanced

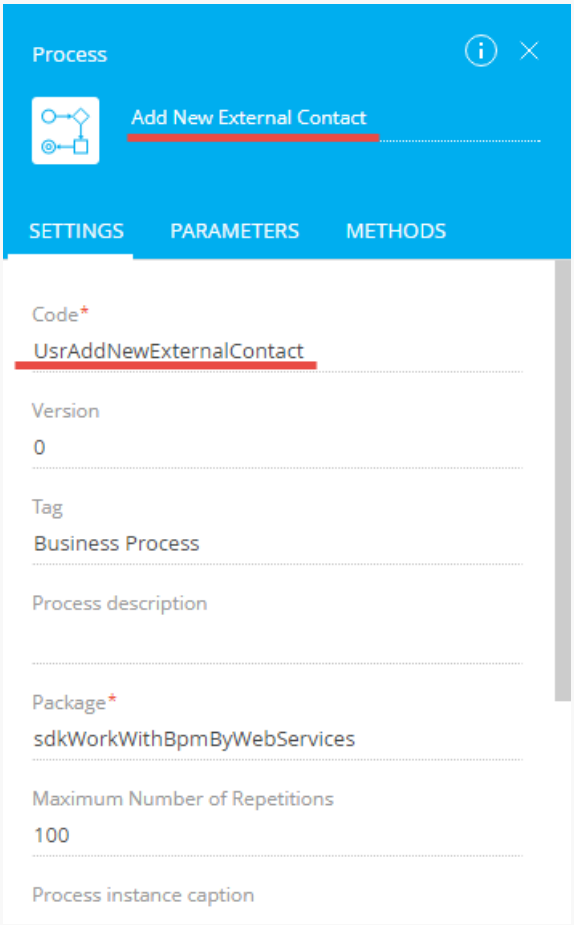
Example. Run custom business processes that create a new contact and read all Creatio contacts. Run the processes from the browser address bar and console app. Use the `ProcessEngineService.svc` web service to run the processes.

1. Create a process that adds a contact

Note. Learn more about the specifics of and best practices for creating business processes in Creatio in the user documentation: BPM tools.

1. Open the [*Configuration*] section and select a custom package to add the business process.
2. Open the Process Designer.
3. Fill out the **business process properties**.
   - Set [*Name*] to "`Add New External Contact`."
   - Set [*Code*] to "`UsrAddNewExternalContact`."

   Use default values for other properties.



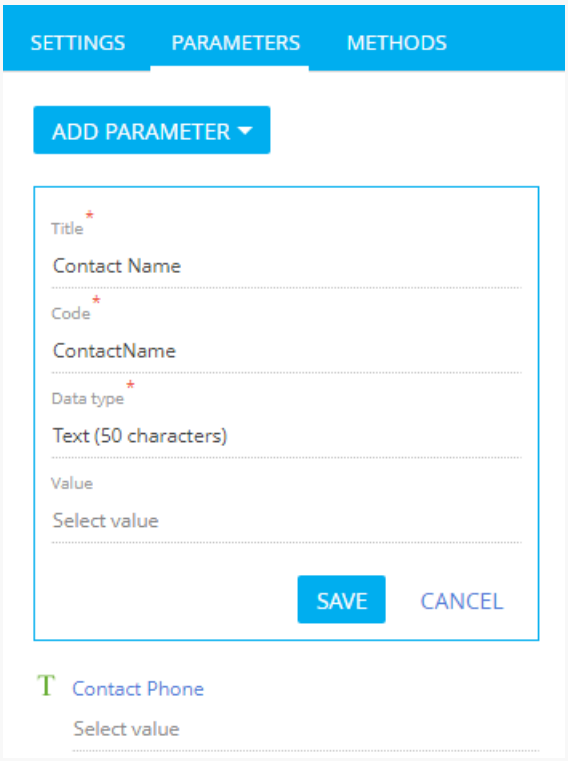
4. Add business process parameters.

   The process uses parameters to pass the data of the new contact: name and phone number.

   View the process parameter values to fill out in the table below.

   Process parameter values

[*Title*]	[*Code*]	[*Data type*]
`ContactName` **parameter**		
"Contact Name"	"ContactName"	"Text (50 characters)"
`ContactPhone` **parameter**		
"Contact Phone"	"ContactPhone"	"Text (50 characters)"



5. Add the [*ScriptTask*] element.

   a. Fill out the **element properties**.

   - Set [*Name*] to "`Add contact`."
   - Set [*Code*] to "`ScriptTaskAddContact`."

   d. Implement the mechanism that adds a new contact.

   To edit the script code, double-click a diagram element. This opens a window on the element setup area. Enter and edit the source code in the window.

`ScriptTaskAddContact`

```
/* Creates an instance of the [Contact] object schema. */
var schema = UserConnection.EntitySchemaManager.GetInstanceByName("Contact");
/* Creates an instance of the new object. */
var entity = schema.CreateEntity(UserConnection);
/* Sets the default values for the object columns. */
entity.SetDefColumnValues();
string contactName = Get<string>("ContactName");
string contactPhone = Get<string>("ContactPhone");
/* Sets the [Name] column values from the process parameter. */
entity.SetColumnValue("Name", contactName);
/* Sets the [Phone] column values from the process parameter. */
entity.SetColumnValue("Phone", contactPhone);
/* Saves the new contact. */
entity.Save();
```

```
return true;
```

6. Click [*Save*].

2. Create a process that reads all Creatio contacts

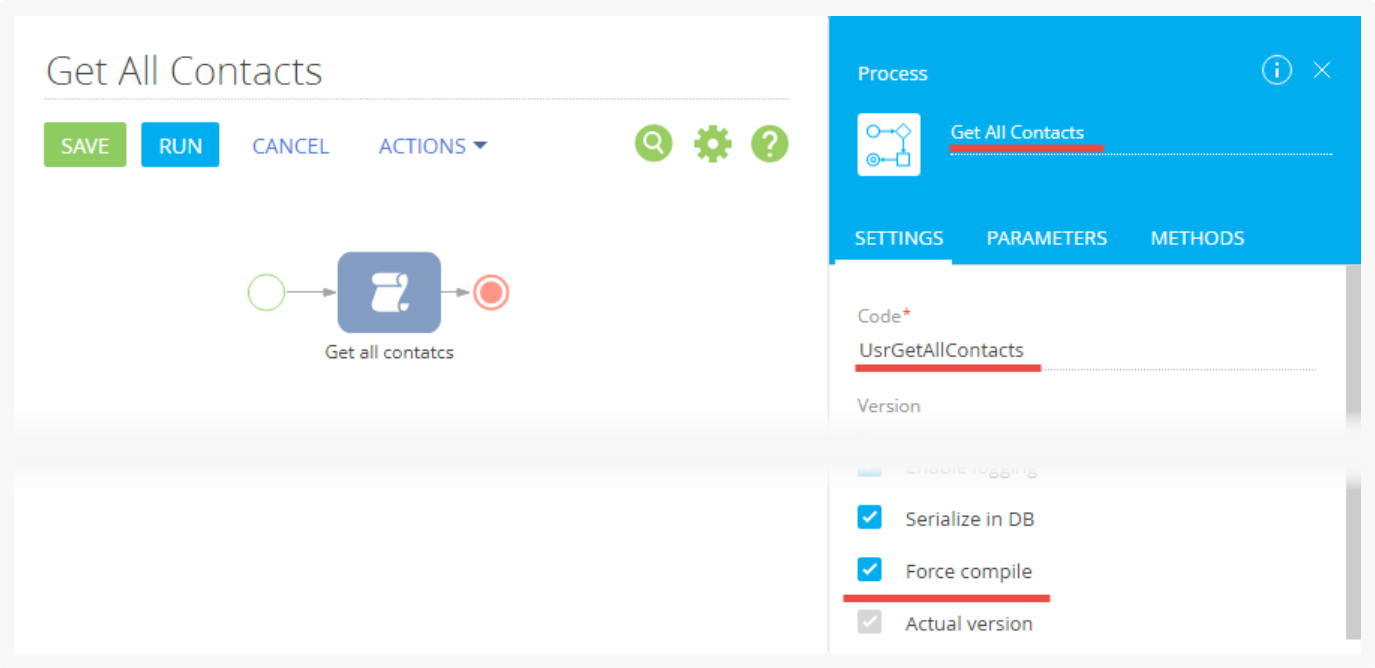
The business process that creates the index of all Creatio contacts includes a single [*ScriptTask*] element as well.

To **create a process that reads all Creatio contacts**:

1. Open the [*Configuration*] section and select a custom package to add the business process.
2. Open the Process Designer.
3. Fill out the **business process properties**.

   - Set [*Name*] to "`Get All Contacts`."
   - Set [*Code*] to "`UsrGetAllContacts`."

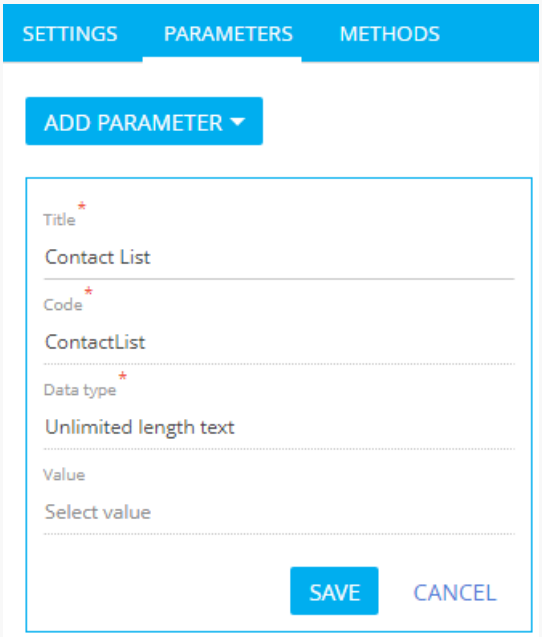
   Use default values for other properties.



4. Add the business process parameter.

   Fill out the **values of the business process parameter**.

   - Set [*Title*] to "`Contact List`."
   - Set [*Code*] to "`ContactList`."
   - Set [*Data type*] to "`Unlimited length text`."



The parameter returns the JSON index of all Creatio contacts.

5. Add the [*ScriptTask*] element.

   a. Fill out the **element properties**.

   - Set [*Name*] to "`Get all contacts`."
   - Set [*Code*] to "`ScriptTaskGetAllContacts`."

   d. Implement the mechanism that reads the Creatio contacts.

   To edit the script code, double-click the diagram. This opens a window on the element setup area. Enter and edit the source code in the window.

ScriptTaskGetAllContacts

```
/* Creates an EntitySchemaQuery instance. */
EntitySchemaQuery query = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Contac
/* Flag the [Id] primary column as required to select. */
query.PrimaryQueryColumn.IsAlwaysSelect = true;
/* Adds columns to the query. */
query.AddColumn("Name");
query.AddColumn("Phone");
/* Retrieve the resulting collection. */
var entities = query.GetEntityCollection(UserConnection);
/* Creates the contact list to serialize in JSON. */
List<object> contacts = new List<object>();
foreach (var item in entities) {
    var contact = new {
        Id = item.GetTypedColumnValue<Guid>("Id"),
        Name = item.GetTypedColumnValue<string>("Name"),
```

```
            Phone = item.GetTypedColumnValue<string>("Phone")
        };
        contacts.Add(contact);
    }
    /* Saves the contact collection that is serialized in JSON to the ContactList parameter. */
    string contactList = JsonConvert.SerializeObject(contacts);
    Set<string>("ContactList", contactList);
    return true;
```

6. Click [*Save*].

Outcome of the example

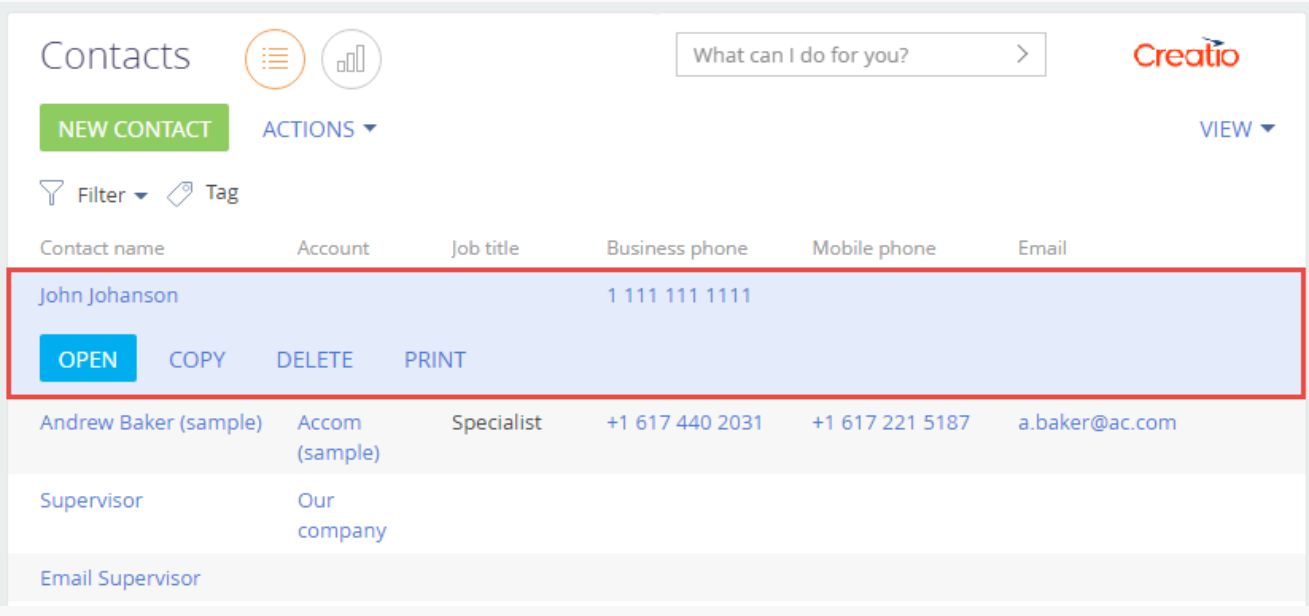
1. **Run business processes from the browser address bar**.

   a. Run the process that adds a contact. To do this, enter the following URL in the browser address bar.

   URL that runs the process that adds a contact

   ```
   http[s]://[CreatioURL]/0/ServiceModel/ProcessEngineService.svc/UsrAddNewExternalContact/
   Execute?ContactName=John Johanson&ContactPhone=1 111 111 1111
   ```

   As a result, the process will add a contact to Creatio.



Attention. A contact is added every time the service receives a request successfully. If you run multiple queries that contain the same parameters, the service adds multiple duplicate contacts.

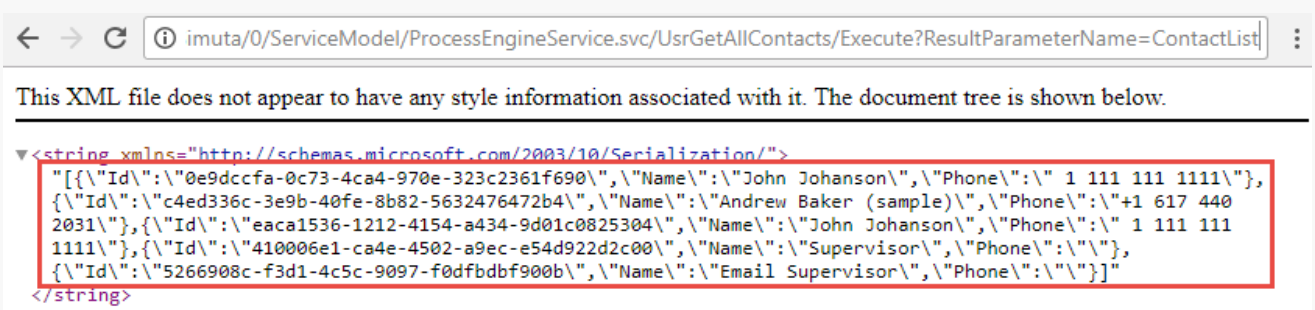
   b. Run the process that reads all Creatio contacts. To do this, enter the following URL in the browser address

bar.

URL that runs the process that reads all Creatio contacts

```
http[s]://[Creatio URL]/0/ServiceModel/ProcessEngineService.svc/UsrGetAllContacts/
Execute?ResultParameterName=ContactList
```

As a result, the browser window will display the JSON object that contains the contact collection.



2. **Run business processes from the console app**.

   View the complete source code of the custom console app that runs business processes via the `ProcessEngineService.svc` service on GitHub.

   a. Perform the **authentication**. To do this, use the `AuthService.svc` authentication service. Use the example available in a separate article: Implement authentication using C#.

   b. Add a string field that contains the base service URL to the source code of the `Program` class.

   URL that creates requests to the `ProcessEngineService.svc` service

   ```
   private const string processServiceUri = baseUri + @"/0/ServiceModel/ProcessEngineService.s
   ```

   c. Add a `GET` method that runs the business process that creates a contact to the source code of the `Program` class.

   `GET` method that runs the business process

   ```
   public static void AddContact(string contactName, string contactPhone) {
       /* Generates a request URL. */
       string requestString = string.Format(processServiceUri + "UsrAddNewExternalContact/Exe
       /* Creates an HTTP request. */
       HttpWebRequest request = HttpWebRequest.Create(requestString) as HttpWebRequest;
       request.Method = "GET";
       request.CookieContainer = AuthCookie;
       /* Executes the request and analyzes the HTTP response. */
       using (var response = request.GetResponse()) {
   ```

```
        /* Displays the properties of the HTTP response if the service returns an empty str
        Console.WriteLine(response.ContentLength);
        Console.WriteLine(response.Headers.Count);
    }
}
```

d. Add `GET` method that runs the business process that reads all Creatio contacts to the `Program` class source code.

`GET` method that runs the business process

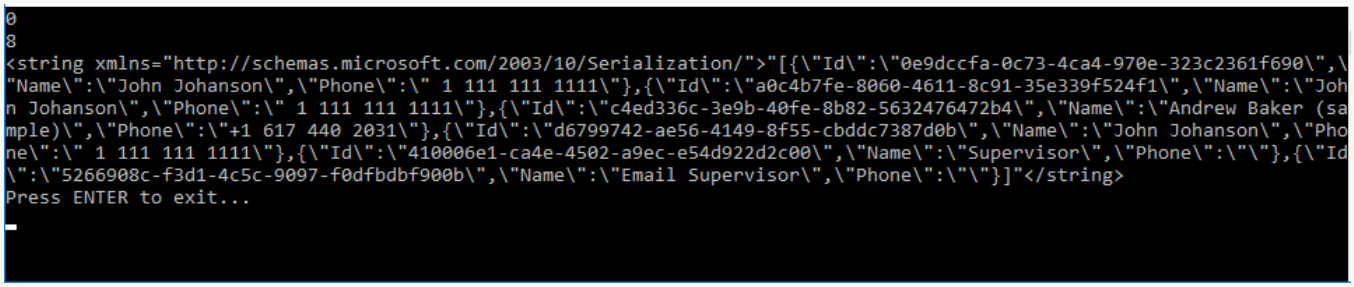
```
public static void GetAllContacts() {
    /* Generates a request URL. */
    string requestString = processServiceUri + "UsrGetAllContacts/Execute?ResultParameterNa
    HttpWebRequest request = HttpWebRequest.Create(requestString) as HttpWebRequest;
    request.Method = "GET";
    request.CookieContainer = AuthCookie;
    /* Creates an HTTP request. */
    using (var response = request.GetResponse()) {
        /* Executes the request and displays the result. */
        using (var reader = new StreamReader(response.GetResponseStream())) {
            string responseText = reader.ReadToEnd();
            Console.WriteLine(responseText);
        }
    }
}
```

e. Call the added methods in the main app method after a successful authentication.

Call the methods

```
static void Main(string[] args) {
    if (!TryLogin("Supervisor", "Supervisor")) {
        Console.WriteLine("Wrong login or password. Application will be terminated.");
    }
    else {
        try {
            /* Calls methods that runs business processes. */
            AddContact("John Johanson", "+1 111 111 1111");
            GetAllContacts();
        }
        catch (Exception) {
            /* Handle exceptions. */
            throw;
        }
    };
```

```
        Console.WriteLine("Press ENTER to exit...");
        Console.ReadLine();
    }
```



Run the business process from a client module

Advanced

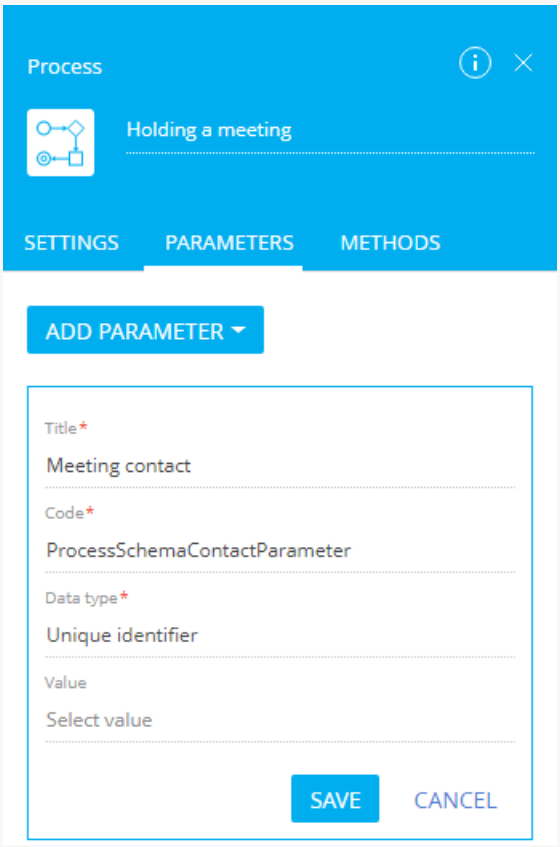
Example. Add the [*Schedule a meeting*] custom action to the account page and [*Accounts*] section list. The action must run the [*Holding a meeting*] custom business process. The action must be active if the account has a primary contact, i. e., the [*Primary contact*] field of the account page is filled out. Pass the primary contact of the account to the business process as a parameter.

1. Implement the [*Holding a meeting*] custom business process

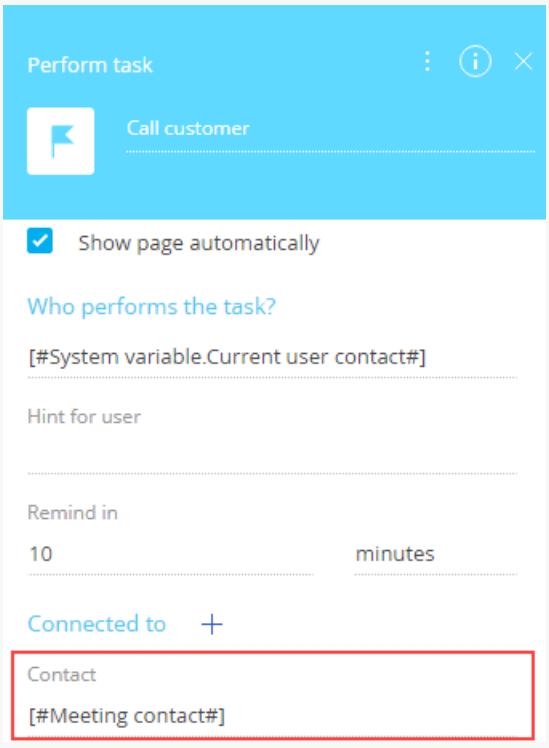
1. Create the [*Holding a meeting*] business process. To do this, implement the example from a separate article: Send emails via business processes.
2. Add the business process parameter.

   Fill out the values of the **business process parameter**.

   - Set [*Title*] to "`Meeting contact`."
   - Set [*Code*] to "`ProcessSchemaContactParameter`."
   - Set [*Data type*] to "`Unique identifier`."



3. Add the "`[#Meeting contact#]`" incoming parameter to the [*Contact*] property of the [*Call customer*] process action.



2. Implement the custom action on the account page

1. Create a replacing view model schema of the account page To do this, follow the guide in a separate article: Add an action to the record page.
2. Add a **localizable string** that contains the menu item caption.
   a. Click the + button in the context menu of the [*Localizable strings*] node.
   b. Fill out the **localizable string properties**.
      - Set [*Code*] to "`CallProcessCaption`."
      - Set [*Value*] to "`Schedule a meeting`."

   e. Click [*Add*] to add a localizable string.
3. Implement the **menu item behavior**.
   a. Add the `ProcessModuleUtilities` module as a dependency.
   b. Implement the following **methods** in the `methods` property:
      - `callCustomProcess()`. Action handler method. Implements the `executeProcess()` method of the `ProcessModuleUtilities` module. Pass the name of the business process and object that contains the initialized incoming process parameters to the module as a parameter.
      - `isAccountPrimaryContactSet()`. Checks whether the primary contact of the account is filled out.
      - `getActions()`. An overloaded base method. Returns the action collection of the replacing page.

   View the source code of the replacing view model schema of the account page below.

AccountPageV2

```
define("AccountPageV2", ["ProcessModuleUtilities"], function(ProcessModuleUtilities) {
    return {
        /* The name of the record page object's schema. */
        entitySchemaName: "Account",
        /* The methods of the record page's view model. */
        methods: {
            /* Verifies the [Primary contact] field of the record page is filled in. */
            isAccountPrimaryContactSet: function() {
                return this.get("PrimaryContact") ? true : false;
            },
            /* Overrides the base virtual method that returns the action collection of the re
            getActions: function() {
                /* Calls the parent method implementation to retrieve the collection of the b
                var actionMenuItems = this.callParent(arguments);
                /* Adds a separator line. */
                actionMenuItems.addItem(this.getActionsMenuItem({
```

```
                    Type: "Terrasoft.MenuSeparator",
                    Caption: ""
                }));
                /* Adds a menu item to the action list of the record page. */
                actionMenuItems.addItem(this.getActionsMenuItem({
                    /* Binds the menu item caption to the localizable schema string. */
                    "Caption": {
                        bindTo: "Resources.Strings.CallProcessCaption"
                    },
                    /* Binds the action handler method. */
                    "Tag": "callCustomProcess",
                    /* Binds the property of the menu item visibility to the value returned b
                    "Visible": {
                        bindTo: "isAccountPrimaryContactSet"
                    }
                }));
                return actionMenuItems;
            },
            /* The action handler method. */
            callCustomProcess: function() {
                /* Receives the ID of the account primary contact. */
                var contactParameter = this.get("PrimaryContact");
                /* The object that transferred to the executeProcess() method as a parameter.
                var args = {
                    /* The name of the process that needs to be run. */
                    sysProcessName: "UsrCustomProcess",
                    /* The object with the ContactParameter incoming parameter value for the
                    parameters: {
                        ProcessSchemaContactParameter: contactParameter.value
                    }
                };
                /* Runs the custom business process. */
                ProcessModuleUtilities.executeProcess(args);
            }
        }
    };
});
```

4. Click [*Save*] on the Module Designer's toolbar.

3. Implement the custom action of the section page

1. Create a replacing view model schema of the section. To do this, follow the guide in a separate article: Add an action to the record page.
2. Add a **localizable string** that contains the menu item caption. To do this, take step 2 of the procedure to implement a custom action of the account page.
3. Implement the **menu item behavior**. To do this, implement the `isAccountPrimaryContactSet()` method in the

`methods` property. The method checks whether the primary contact of the account is filled out.

View the source code of the replacing view model schema of the section page below.

AccountSectionV2

```
define("AccountSectionV2", [], function() {
    return {
        /* The name of the section page schema. */
        entitySchemaName: "Account",
        /* The methods of the section page's view model. */
        methods: {
            /* Verifies the [Primary contact] field of the selected account is filled in. */
            isAccountPrimaryContactSet: function() {
                /* Defines the active record. */
                var activeRowId = this.get("ActiveRow");
                if (!activeRowId) {
                    return false;
                }
                /* Retrieves the data collection of the section list's list view. Retrieves t
                var selectedAccount = this.get("GridData").get(activeRowId);
                if (selectedAccount) {
                    /* Receives the model property – the primary contact. */
                    var selectedPrimaryContact = selectedAccount.get("PrimaryContact");
                    /* Returns true if the primary contact is filled in, return false otherwi
                    return selectedPrimaryContact ? true : false;
                }
                return false;
            }
        }
    };
});
```

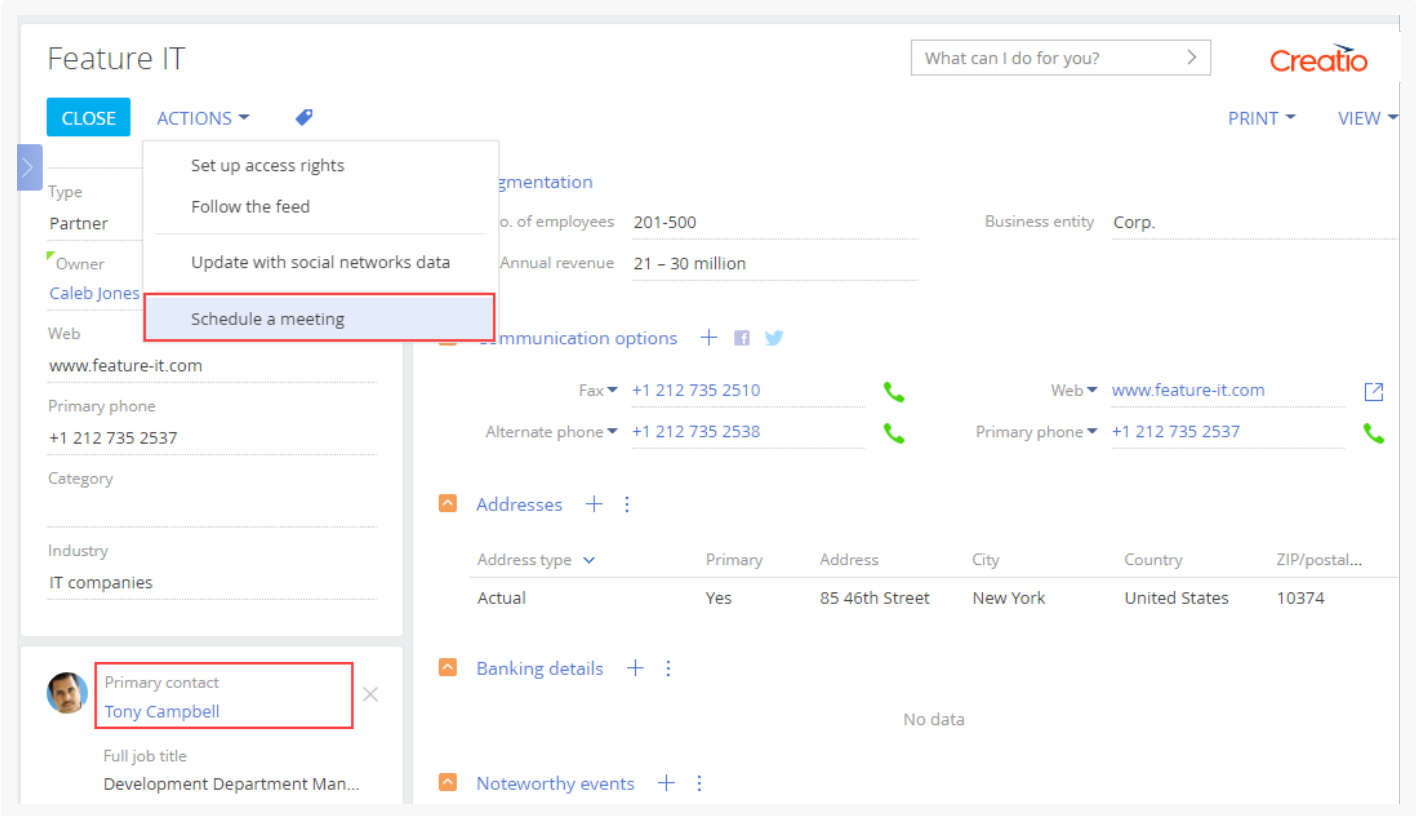
4. Click [*Save*] on the Module Designer's toolbar.

Outcome of the example

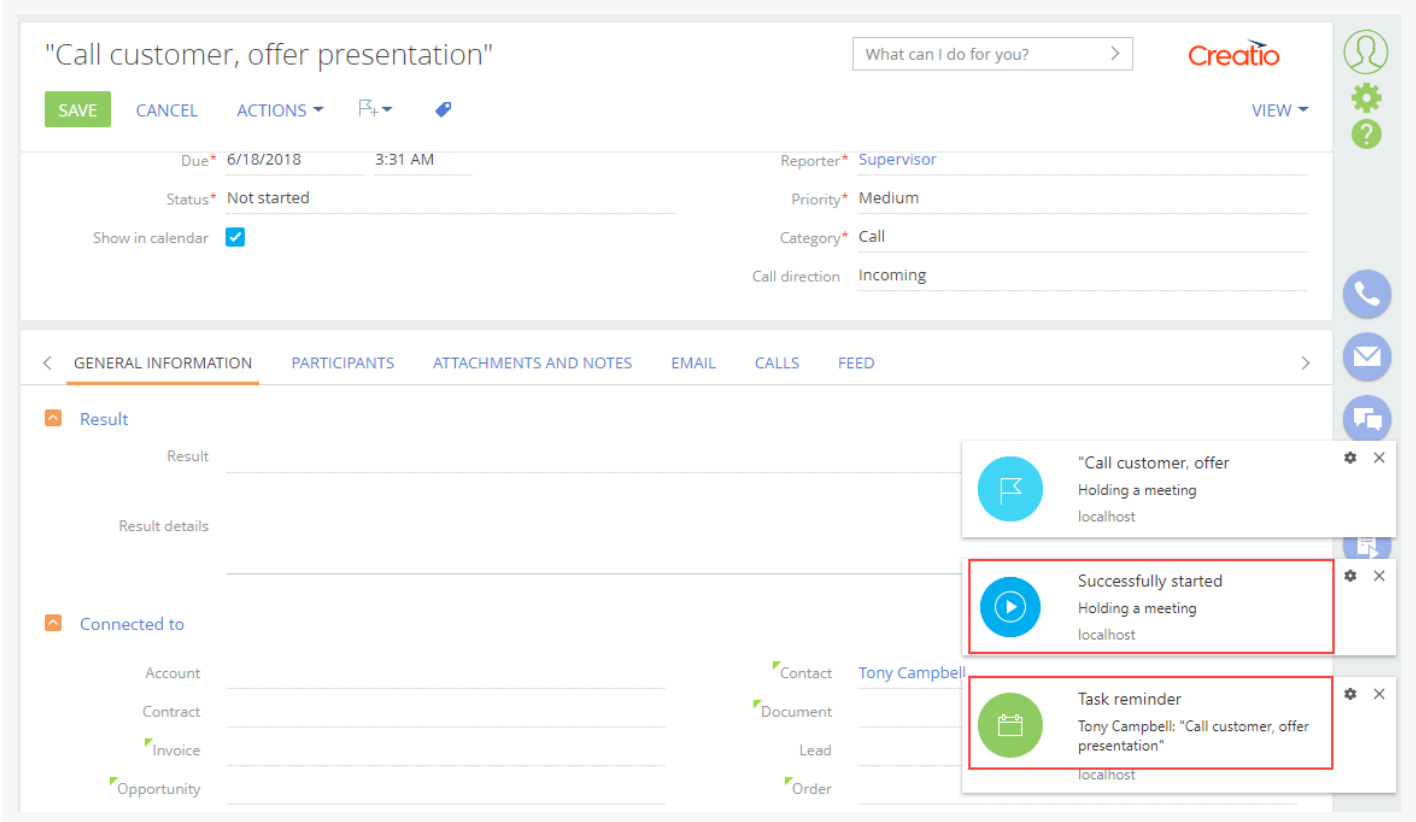
To **view the outcome of the example**:

1. Clear the browser cache.
2. Refresh the [*Accounts*] section page.

As a result, Creatio will add the [*Schedule a meeting*] action to the account page. This action will be active if the primary contact of the account is filled out.



Select the action to run the [*Holding a meeting*] custom business process . The primary contact of the account will be passed to the business process as a parameter.



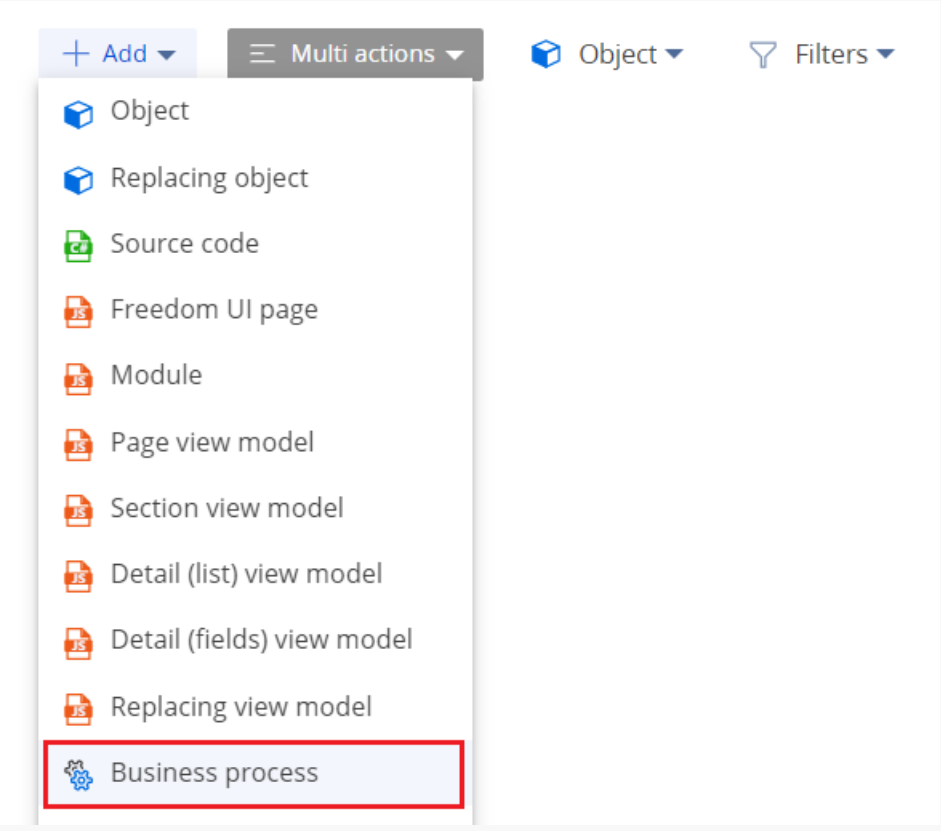
Run the business process that receives parameters from a client module

Medium

Example. Add the [*Receive parameters*] custom action to the account page and [*Accounts*] section list. The action must run the [*Result parameters*] custom business process. The action must be active if the account has a primary contact, i. e., the [*Primary contact*] field of the account page is filled out. Pass the primary contact of the account to the business process as a parameter. Display the job title and age of the account's primary contact as resulting parameters in the business process dialog.

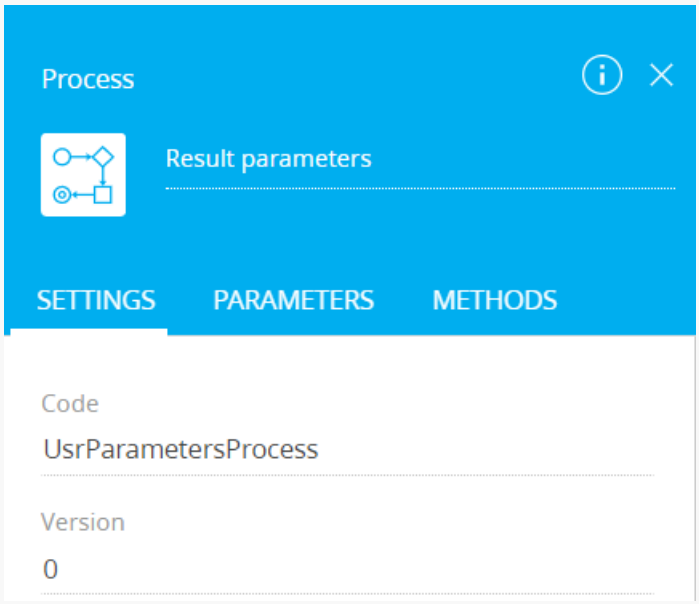
1. Implement the [*Result parameters*] custom business process

1. Open the [*Configuration*] section and select a custom package to add the business process.
2. Click [*Add*] → [*Business process*] on the section list toolbar.



3. Fill out the **business process properties**.
   - Set [*Title*] to "`Result parameters`."
   - Set [*Code*] to "`UsrParametersProcess`."

Use default values for the other properties.

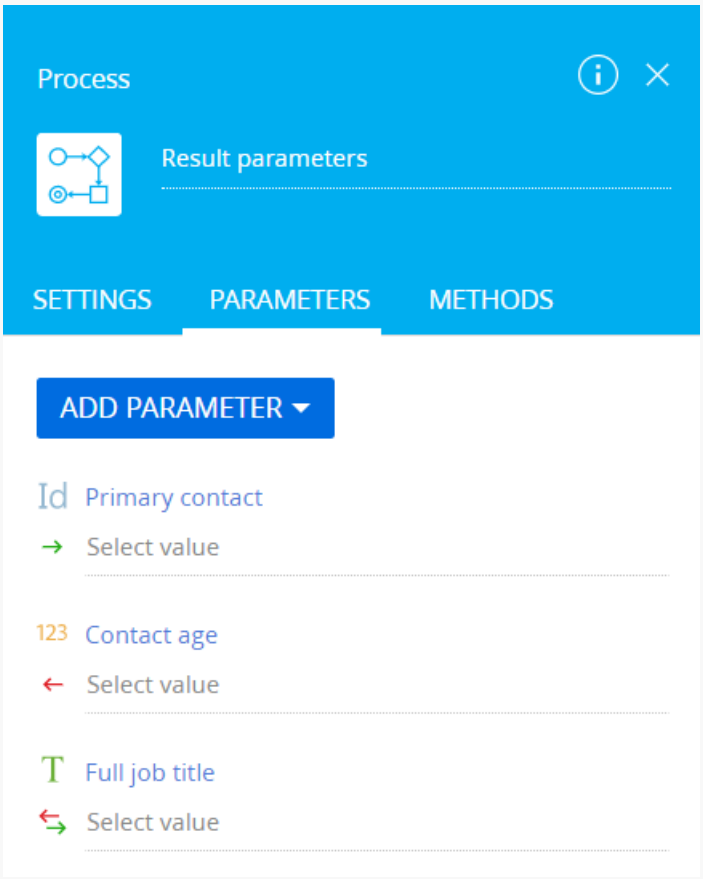


4. Add **business process parameters**.

   View the process parameters values to fil out in the table below.

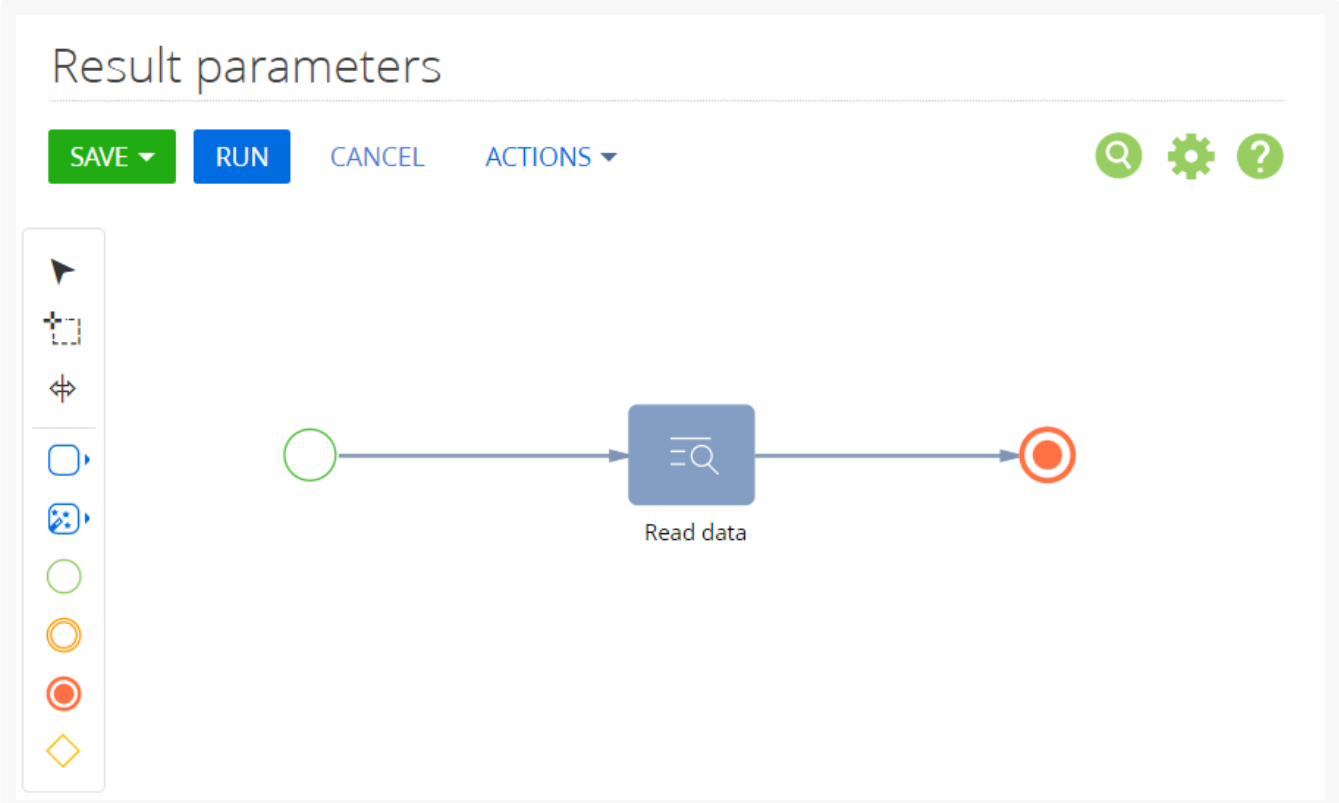
Process parameters values

[*Title*]	[*Code*]	[*Data type*]	[*Direction*]
"Primary contact"	"ProcessSchemaContact Parameter"	"Unique identifier"	"Input"
"Contact age"	"ProcessContactAge"	"Integer"	"Output"
"Full job title"	"ProcessFullJob"	"Text (500 characters)"	"Output"



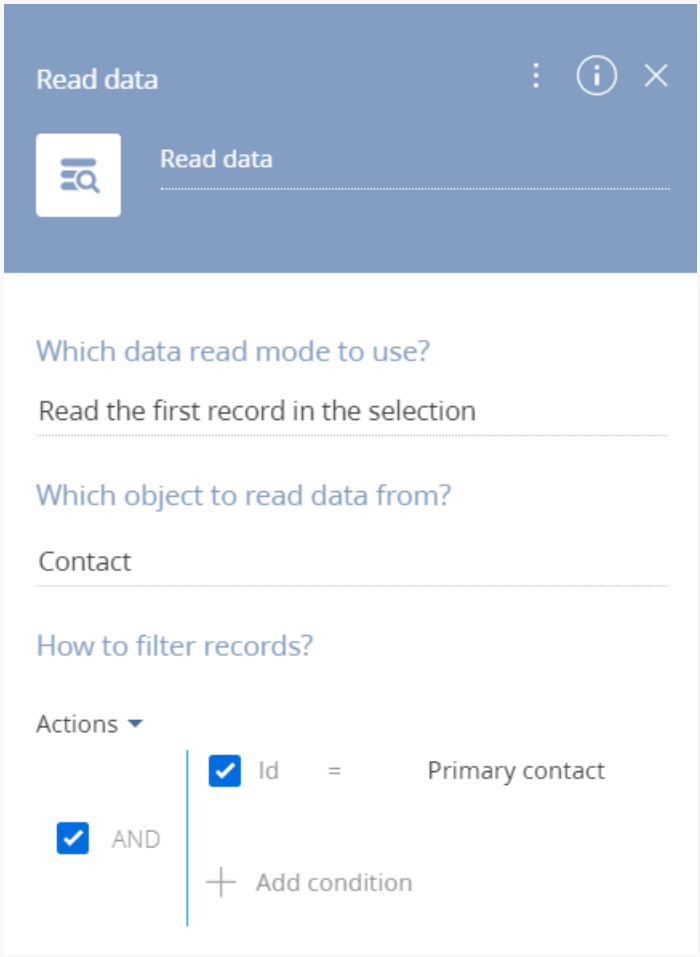
5. Implement data reading.

   a. Add the [*Read data*] element to the Process Designer workspace.



b. Fill out the **element properties**.

- Set [*Which data read mode to use?*] to "`Read the first record in the selection`."
- Set [*Which object to read data from?*] to "`Contact`."
- Set the filter by the [*Id*] column in the [*How to filter records?*] property. Set the value of the "`Primary contact`" incoming parameter as the column value.

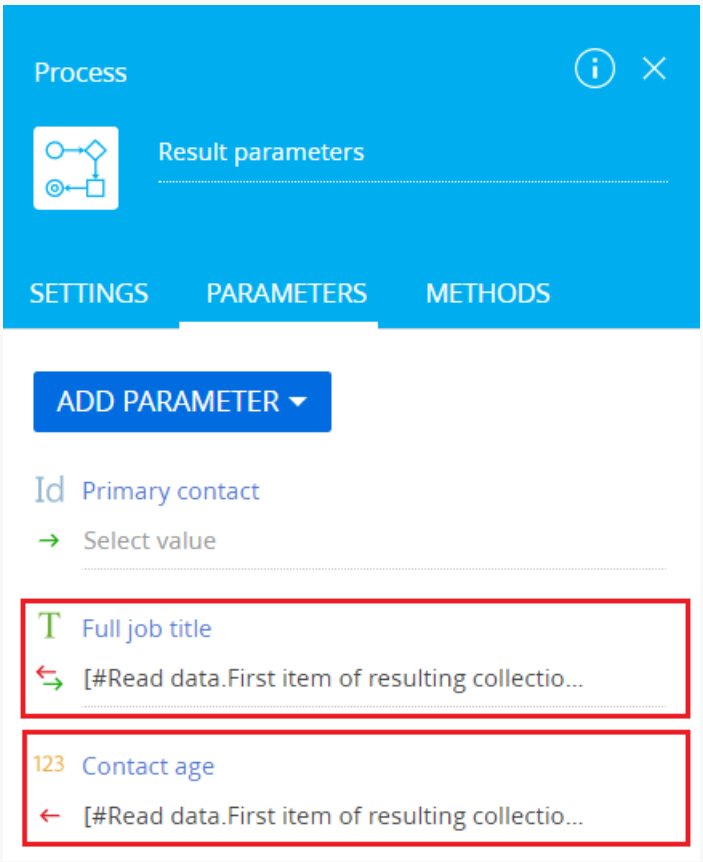


6. Add the **values of the outgoing parameters**.

a. Go to the [*Parameters*] tab.

b. Enter "`[#Read data.First item of resulting collection.Full job title#]`" in the [*Value*] field for the `Full job title` parameter.

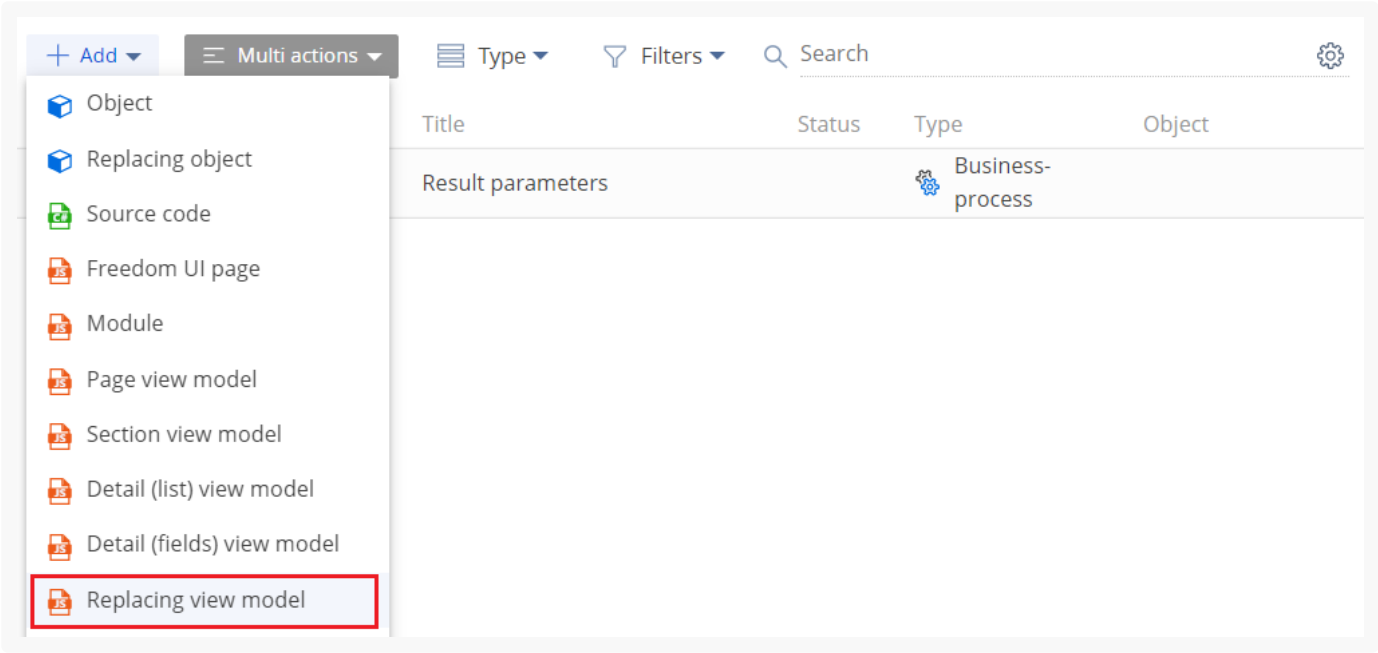
c. Enter "`[#Read data.First item of resulting collection.Age#]`" in the [*Value*] field for the `Contact age` parameter.



7. Click [*Save*] on the Process Designer's toolbar.

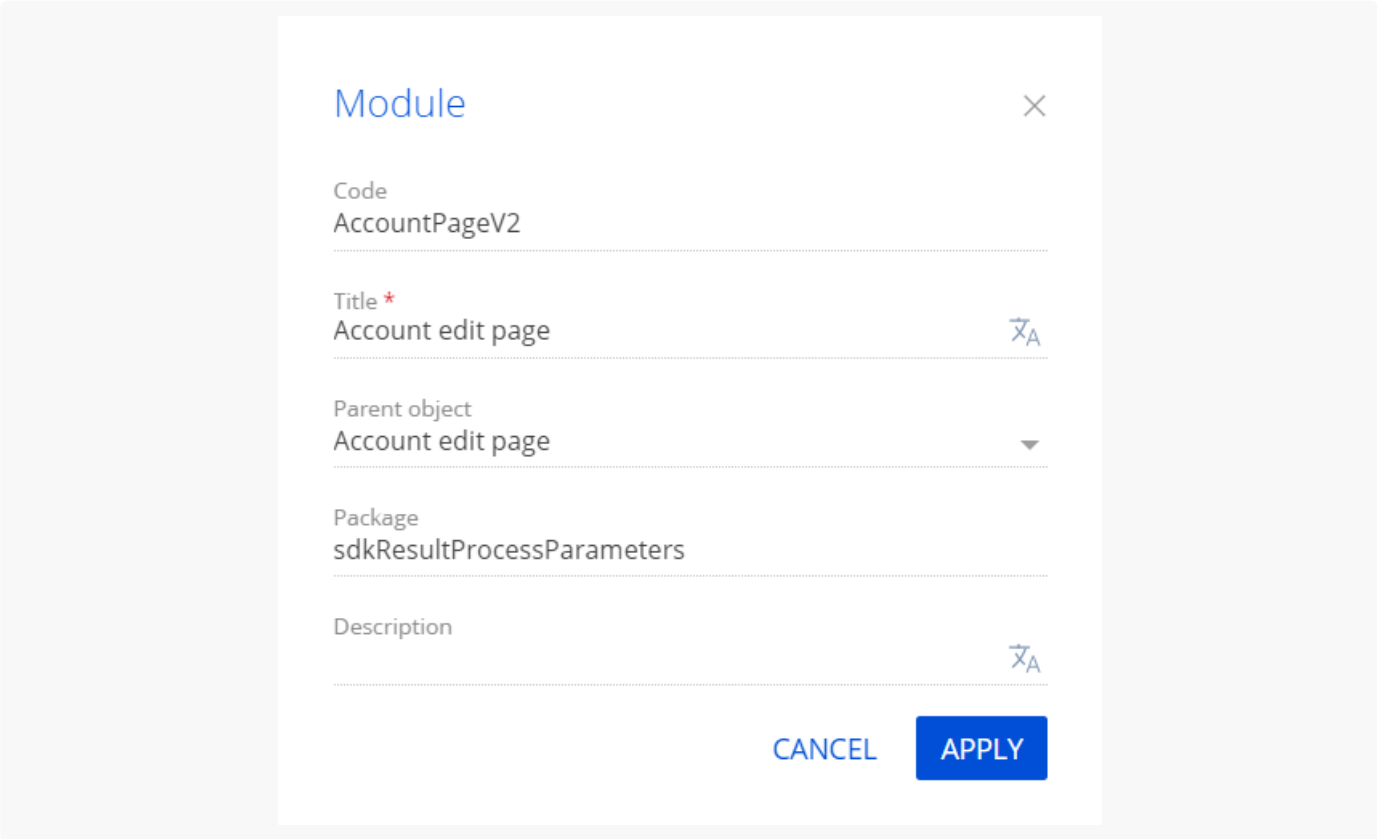
2. Create a replacing view model schema of the account page

1. Open the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Replacing view model*] on the section list toolbar.



3. Fill out the **schema properties**.

   - Set [*Code*] to "`AccountPageV2`."
   - Set [*Title*] to "`Account edit page`."
   - Set [*Parent object*] to "`AccountPageV2`."

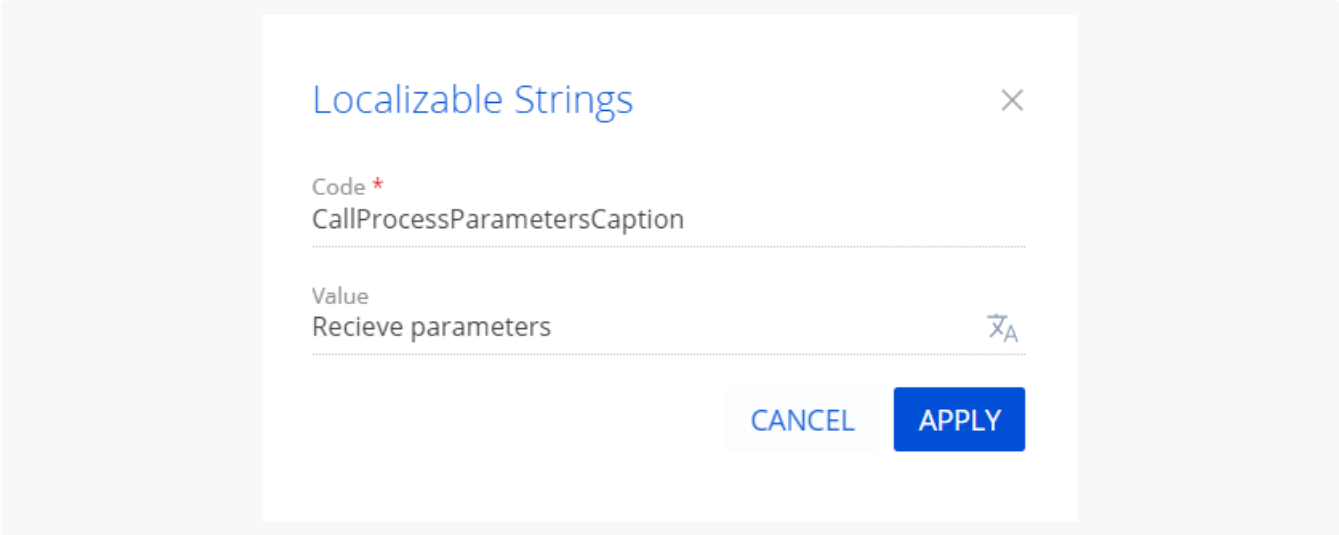


4. Add a **localizable string** that contains the menu item caption.

   a. Click the + button in the context menu of the [*Localizable strings*]) node.

   b. Fill out the **localizable string properties**.

      - Set [*Code*] to "`CallProcessParametersCaption`."
      - Set [*Value*] to "`Recieve parameters`."



e. Click [*Add*] to add a localizable string.

5. Implement the **menu item logic**.

   Implement the following **methods** in the `methods` property:

   - `isAccountPrimaryContactSet()`. Checks whether the primary contact of the account is filled out and defines the visibility of the menu item.
   - `callCustomProcess()`. Action handler method. Runs the business process and displays the values of the resulting parameters in the dialog.
   - `getActions()`. An overloaded base method. Returns the action collection of the replacing page.

   View the source code of the replacing view model schema of the account page below.

AccountPageV2

```
define("AccountPageV2", [], function() {
    return {
        /* The name of the record page object's schema. */
        entitySchemaName: "Account",
        /* The methods of the record page's view model. */
        methods: {
            /* Check the [Primary contact] field of the account is filled out to define the v
            isAccountPrimaryContactSet: function() {
                /* Return true if the primary contact is filled out, return false otherwise.
                return this.get("PrimaryContact") ? true : false;
            },
            /* Overload the base virtual method that returns the collection of record page ac
            getActions: function() {
                /* Call the parent method implementation to retrieve the collection of base p
                var actionMenuItems = this.callParent(arguments);
                /* Add a separator line. */
                actionMenuItems.addItem(this.getActionsMenuItem({
                    Type: "Terrasoft.MenuSeparator",
```

```
                    Caption: ""
                }));
                /* Add a menu item to the action list of the record page. */
                actionMenuItems.addItem(this.getActionsMenuItem({
                    /* Bind the menu item caption to the localizable schema string. */
                    "Caption": {
                        bindTo: "Resources.Strings.CallProcessParametersCaption"
                    },
                    /* Bind the action handler method. */
                    "Tag": "callCustomProcess",
                    /* Bind the property of the menu item visibility to the value returned by
                    "Visible": {
                        bindTo: "isAccountPrimaryContactSet"
                    }
                }));
                return actionMenuItems;
            },
            /* The action handler method. Run the business process and display the values of
            callCustomProcess: function() {
                /* Receive the ID of the account’s primary contact. */
                var contactParameter = this.get("PrimaryContact");
                /* Create an object of the Terrasoft.RunProcessRequest class. */
                const runProcessRequest = Ext.create("Terrasoft.RunProcessRequest", {
                    /* The process schema code. */
                    "schemaName": "UsrParametersProcess",
                    /* The ID of the process diagram. */
                    "schemaUId": "9ff60fa7-314e-4b9e-b9fc-57f91b8b2e21",
                    /* Incoming process parameters. */
                    "parameterValues": {
                        "ProcessSchemaContactParameter": contactParameter.value
                    },
                    /* Outgoing process parameters. */
                    "resultParameterNames": [
                        "ProcessContactAge",
                        "ProcessFullJob"
                    ]
                });
                /* Run the custom business process. */
                runProcessRequest.execute(function(response) {
                    /* If the process is executed successfully, display the values of the res
                    if (response.isSuccess()) {
                        var job = response.resultParameterValues["ProcessFullJob"];
                        var age = response.resultParameterValues["ProcessContactAge"];
                        Terrasoft.showInformation("Full job title: " + job + " Age: " + age);
                    }
                }, this);
            }
        }
    };
```

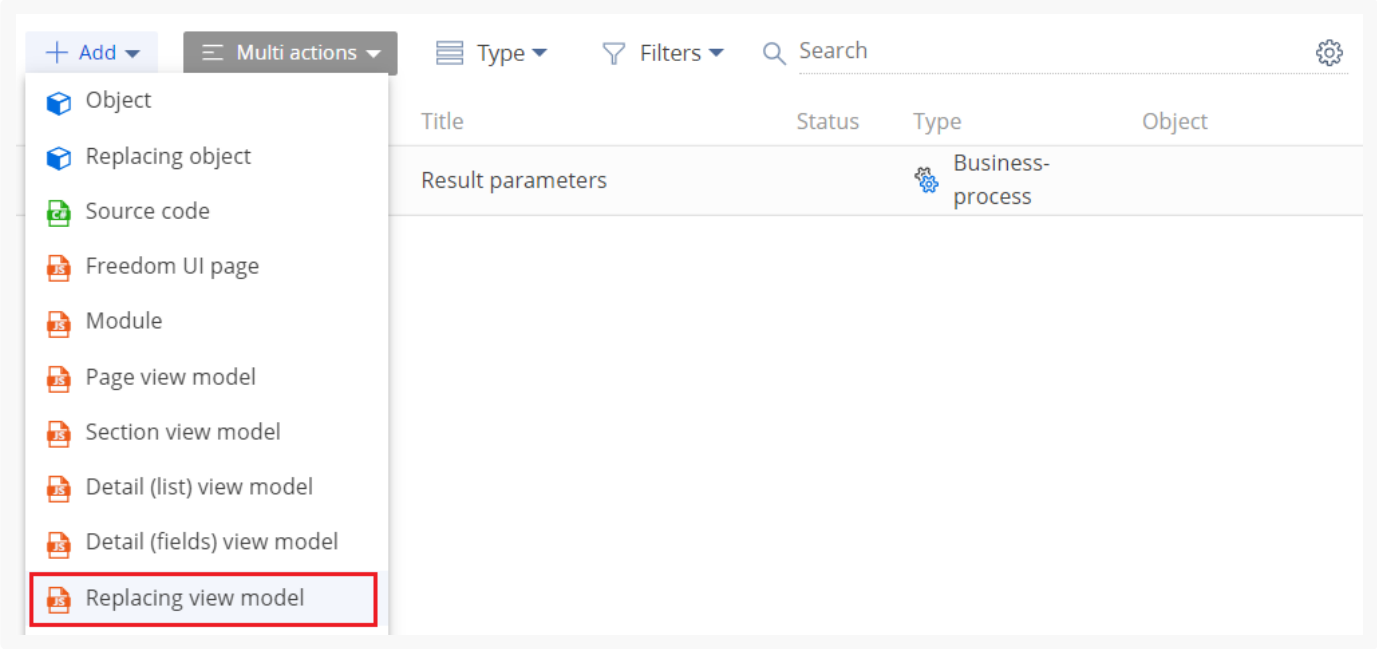
```
});
```

6. Click [*Save*] on the Module Designer's toolbar.

Creatio does not display the custom account page action in combined mode.

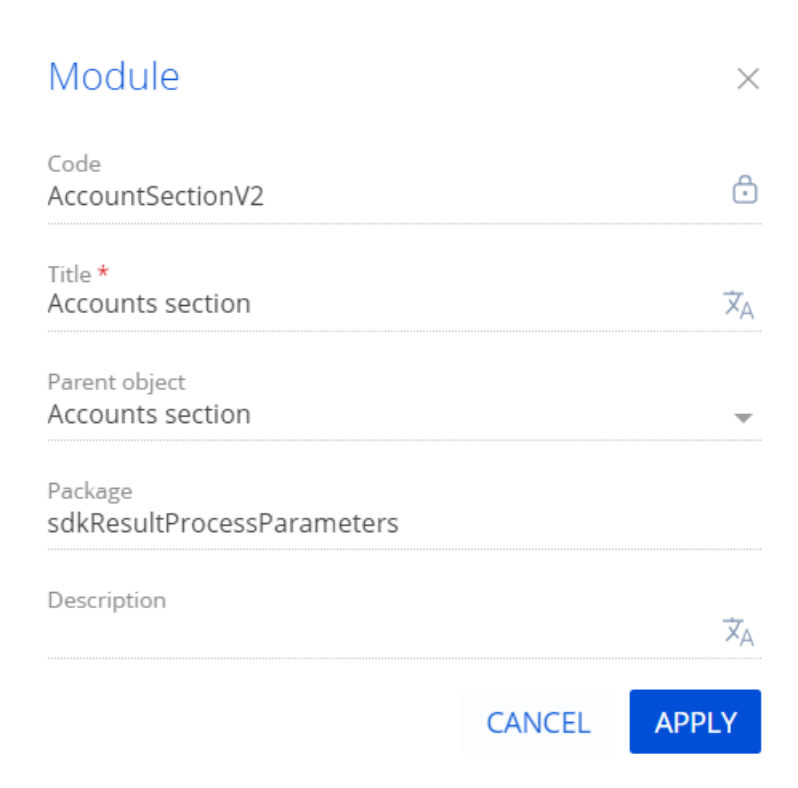
3. Create a replacing view model schema of the section page

1. Open the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Replacing view model*] on the section list toolbar.



3. Fill out the **schema properties**.

- Set [*Code*] to "`AccountSectionV2`."
- Set [*Title*] to "`Accounts section`."
- Set [*Parent object*] to "`AccountSectionV2`."



4. Add a **localizable string** that contains the menu item caption. To do this, take step 4 of the procedure to create a replacing view model schema of the account page.
5. Implement the **menu item behavior**. To do this, implement the `isAccountPrimaryContactSet()` method in the `methods` property. The method checks whether the primary contact of the account is filled out and defines the visibility of the menu item.

   View the source code of the replacing view model schema of the section page below.

AccountSectionV2

```
define("AccountSectionV2", [], function() {
    return {
        /* The name of the section page schema. */
        entitySchemaName: "Account",
        /* The methods of the section page's view model. */
        methods: {
            /* Check whether the [Primary contact] field of the account is filled out. */
            isAccountPrimaryContactSet: function() {
                /* Define the active record. */
                var activeRowId = this.get("ActiveRow");
                if (!activeRowId) {
                    return false;
                }
                /* Retrieve the data collection of the section list's list view. Retrieve the
                var selectedAccount = this.get("GridData").get(activeRowId);
                if (selectedAccount) {
```

```
                    /* Receive the [Primary contact] model property. */
                    var selectedPrimaryContact = selectedAccount.get("PrimaryContact");
                    /* Return true if the primary contact is filled out, return false otherwi
                    return selectedPrimaryContact ? true : false;
                }
                return false;
            }
        }
    };
});
```

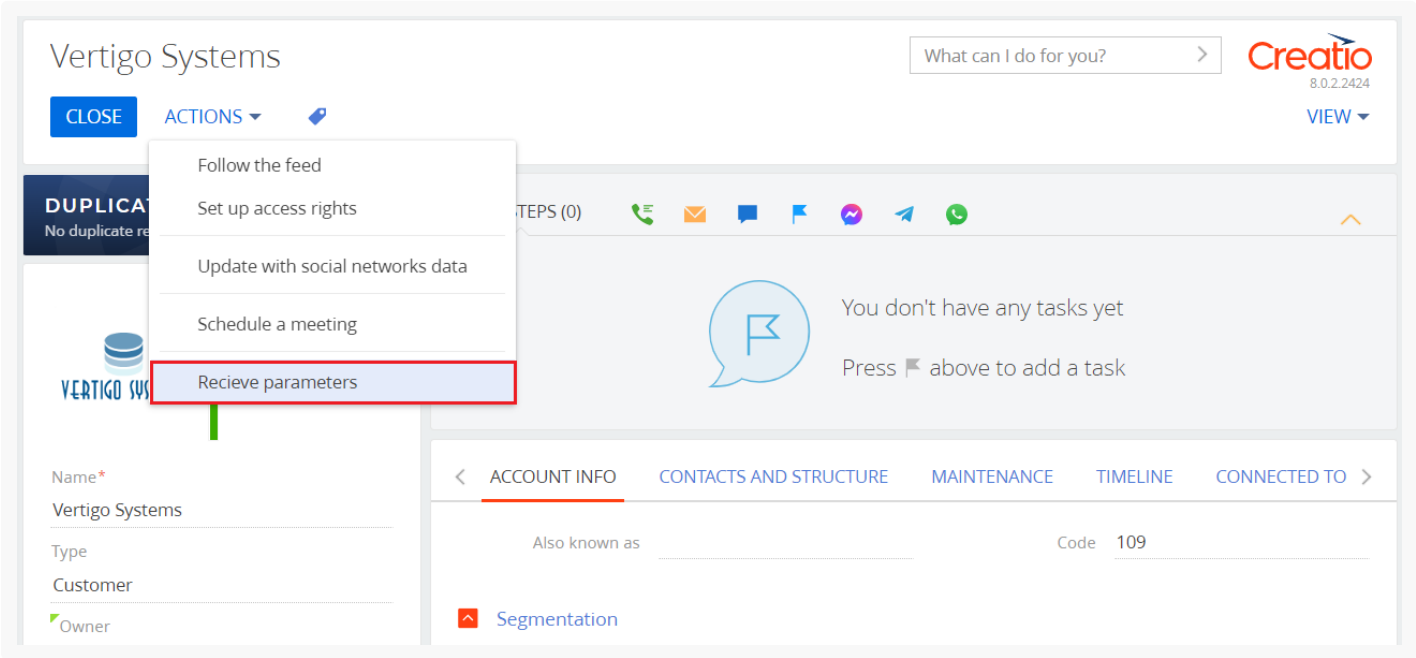
6. Click [*Save*] on the Module Designer's toolbar.

Outcome of the example

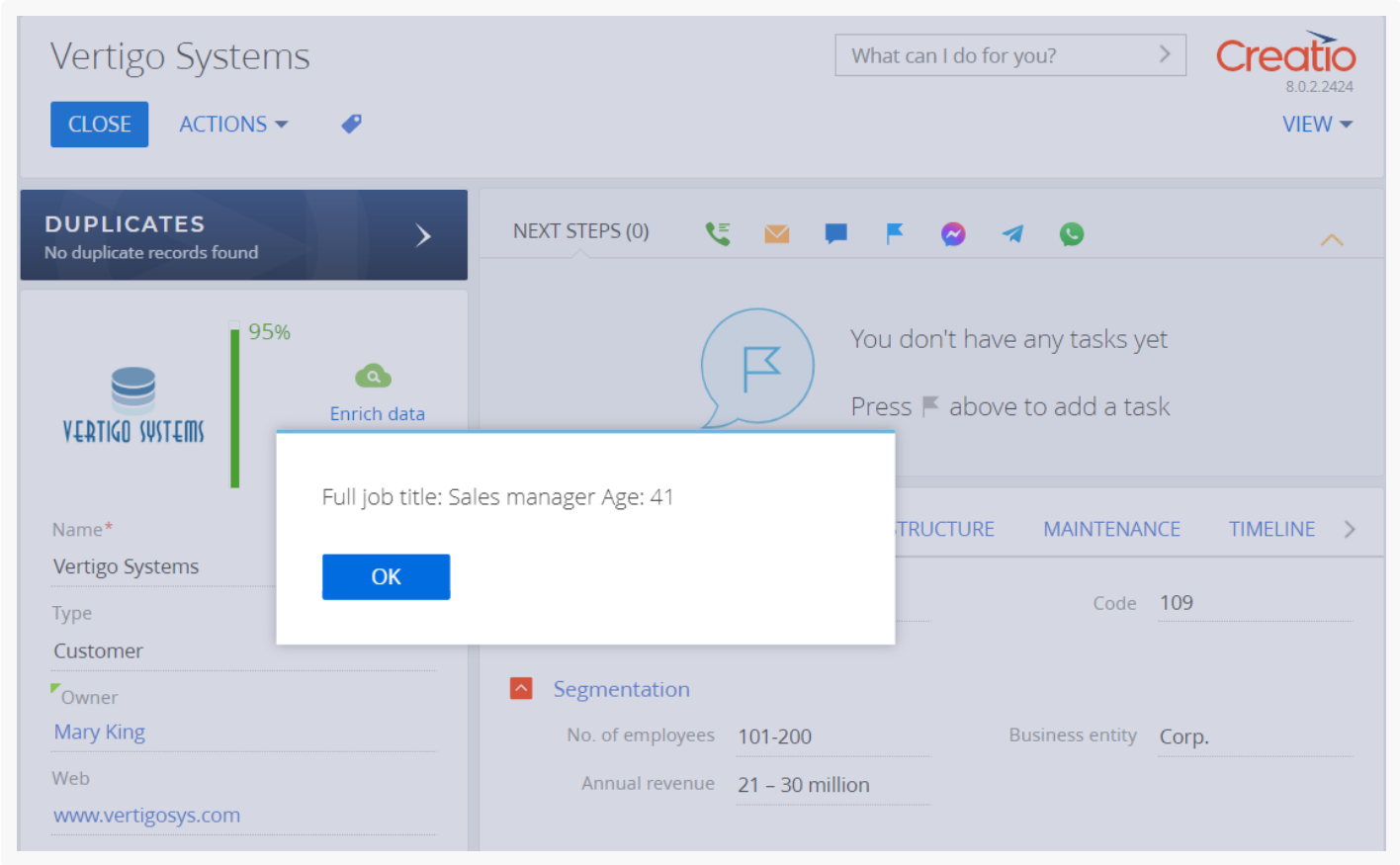
To **view the outcome of the example**:

1. Refresh the [*Accounts*] section page.
2. Open an account page, for example, `Vertigo Systems`.

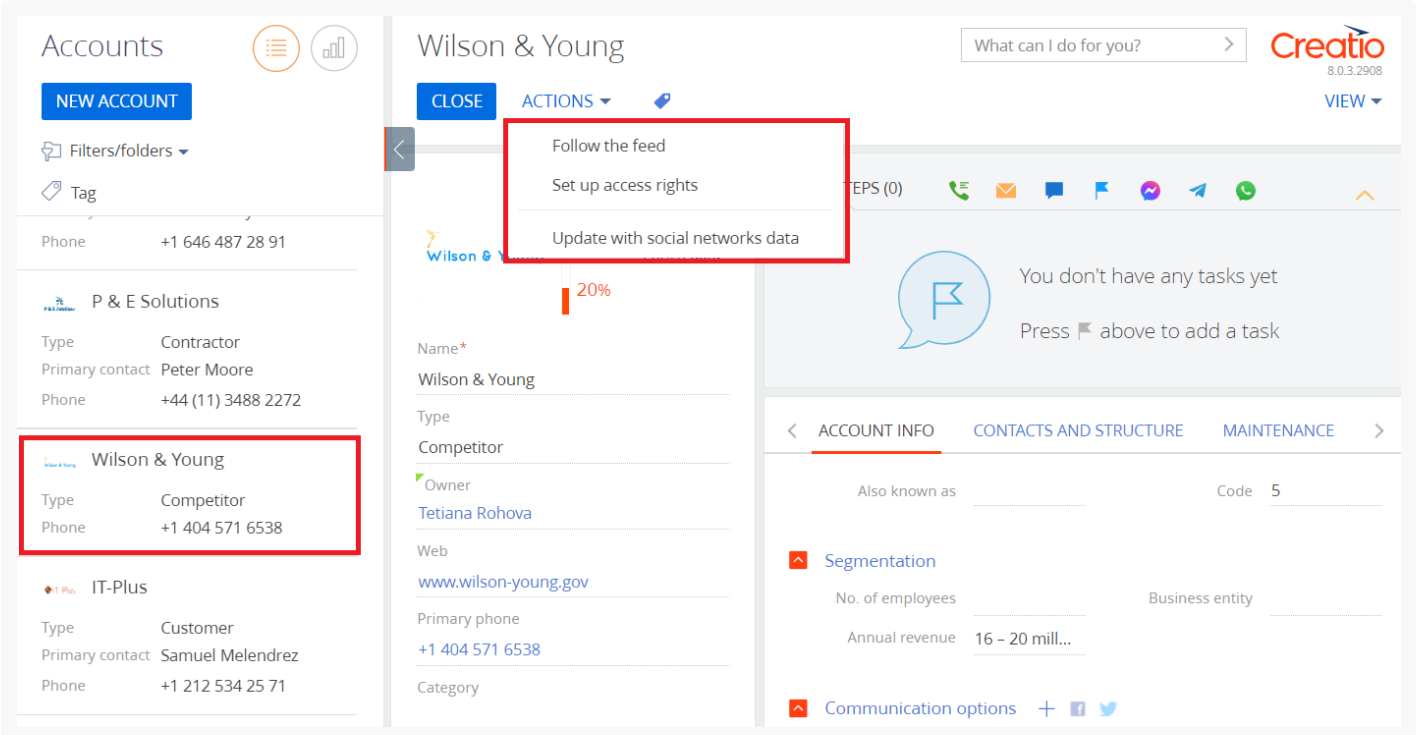
As a result, Creatio will add the [*Receive parameters*] action to the account page.



Click the [*Receive parameters*] action to run the [*Result parameters*] business process. The primary contact of the account will be passed to the business process as a parameter. The job title and age of the account's primary contact will be displayed as the resulting parameters of the business process.

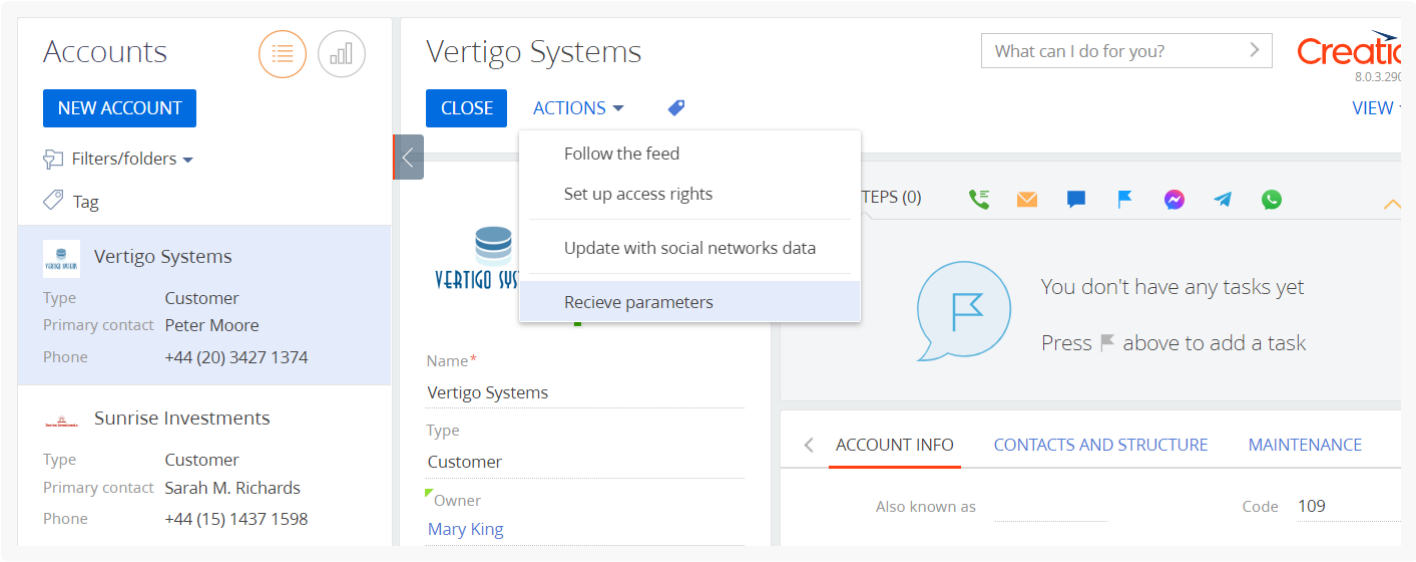


The action will be active if the account has a primary contact, i. e., the [*Primary contact*] field of the account page is filled out. For example, the action is inactive for the `Wilson & Young` account.



The [*Receive parameters*] custom action will be displayed on the [*Accounts*] section page and active for the

selected account that has a primary contact.



ProcessEngineService.svc web service API

Advanced

Request structure

```
/* Request string. */
GET Creatio_application_address/0/ServiceModel/ProcessEngineService.svc/[processSchemaName/]meth

/* Request headers. */
ForceUseSession: true
BPMCSRF: authentication_cookie_value
```

Response structure for the Execute() method

```
/* Status code. */
Status: code

/* Response body. */
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">
    "[
        {\"Id\":\"object1_id\",\"object1 field1\":\"object1 field_value1\",\"object1 field2\":\"
        {\"Id\":\"object2_id\",\"object2 field1\":\"object2 field_value1\",\"object2 field2\":\"
        ... {},
    ]"
</string>
```

Request string

`GET` **required**

Method of the request to run business processes.

`Creatio_application_address` **required**

Creatio instance URL.

`ServiceModel` **required**

The path to the service that runs business processes. Unmodifiable part of the request.

`ProcessEngineService.svc` **required**

The address of the web service that runs business processes. Unmodifiable part of the request.

`methodName` **required**

The method of the web service that runs business processes.

Primary **methods**:

- `Execute()`. Runs a business process. Lets you pass a set of incoming parameters and return the outcome of the web service execution.
- `ExecProcElByUId()`. Executes a specific business process element. You can only execute an element of the running process. If Creatio has already executed the process element by the time the method is called, the element is not executed again.

`ProcessSchemaName` **optional**

Applicable for the `Execute()` method.

The name of the business process schema. You can find the name of the business process schema in the [*Configuration*] section.

`ResultParameterName` **optional**

Applicable for the `Execute()` method.

The variable of the process parameter code. Unmodifiable part of the request.

`resultParameterName` **optional**

Applicable for the `Execute()` method.

The code of the process parameter that stores the outcome of the process execution. If you omit this parameter, the web service runs the business process without waiting for the outcome of the process execution. If the process lacks the parameter code, the web service returns `null`.

`inputParameter` **optional**

Applicable for the `Execute()` method.

The variable of the incoming process parameter code. If you pass multiple incoming parameters, combine them using the `&` character.

`inputParameterValue` **optional**

Applicable for the `Execute()` method.

The values of the incoming process parameters.

`ProcessElementUID` **optional**

Applicable for the `ExecProcElByUId()` method.

The ID of the process element to run.

Request headers

`ForceUseSession` `true` **required**

Forced use of an existing session.

`BPMCSRF` `authentication_cookie_value` **required**

The authentication cookie.

Response body

`[]`

The object collection.

`{}`

The collection object instances.

`Id`

The name of the [*Id*] field.

`object1_id, object2_id, ...`

The IDs of the collection object instances.

`object1 field1, object1 field2, ..., object2 field1, object2 field2, ...`

The names of the `field1, field2, ...` fields in the `object1, object2, ...` collection object instances.

`object1 field_value1, object1 field_value2, ..., object2 field_value1, object2 field_value2, ...`

The values of the `field1, field2, ...` fields in the `object1, object2, ...` collection object instances. Available only for `GET` and `POST` requests.