WEBCON

BPS 2020.1

Changes to SDK & API

1. Introduction

The following document contains a list of changes in WEBCON BPS SDK and API when compared to version 2019.1. These changes break code compatibility or change how existing code works.

2. Changes to plugin packages

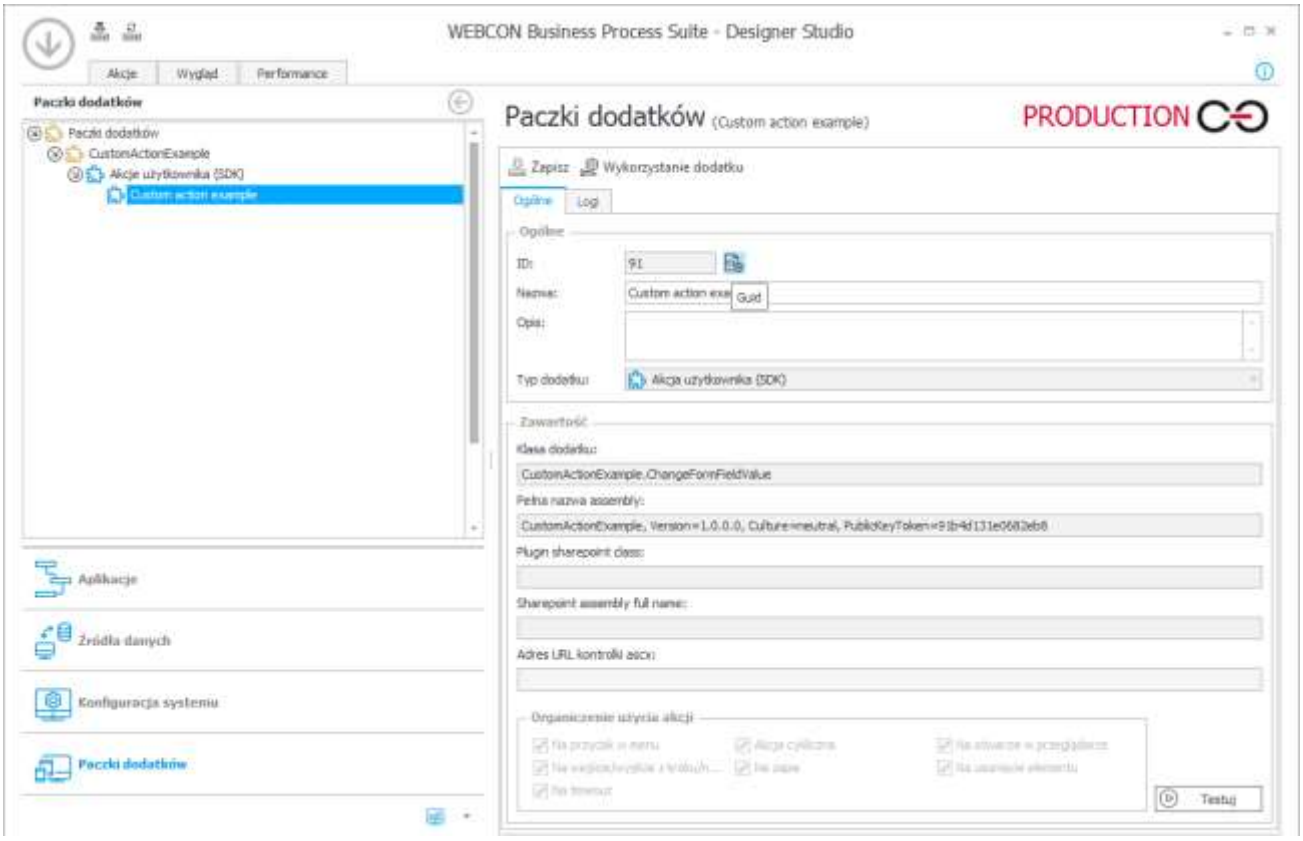
Changes to plugin packages apply only to version older than 2019.1.4, due to the introduction of a new field in the plugin package manifest: GUID.

Depending on the version of WEBCON BPS that we will be upgrading from, there are two possible scenarios:

2.1. Upgrading from version lower than 2019.1

The first scenario involves migrating plugins from a version of WEBCON BPS where plugin packages didn't yet exist e.g. BPS 2017. As soon as WEBCON BPS is updated, empty plugin packages will be created automatically. For every package, you will need to migrate every individual plugin. This process is described in the BPS 2019.1 SDK migration document.

For every registered plugin, you will need to update the plugin package manifest with their information, including the new GUID field found in the database. The best way to find the GUID is to use WEBCON BPS Designer Studio as shown here:

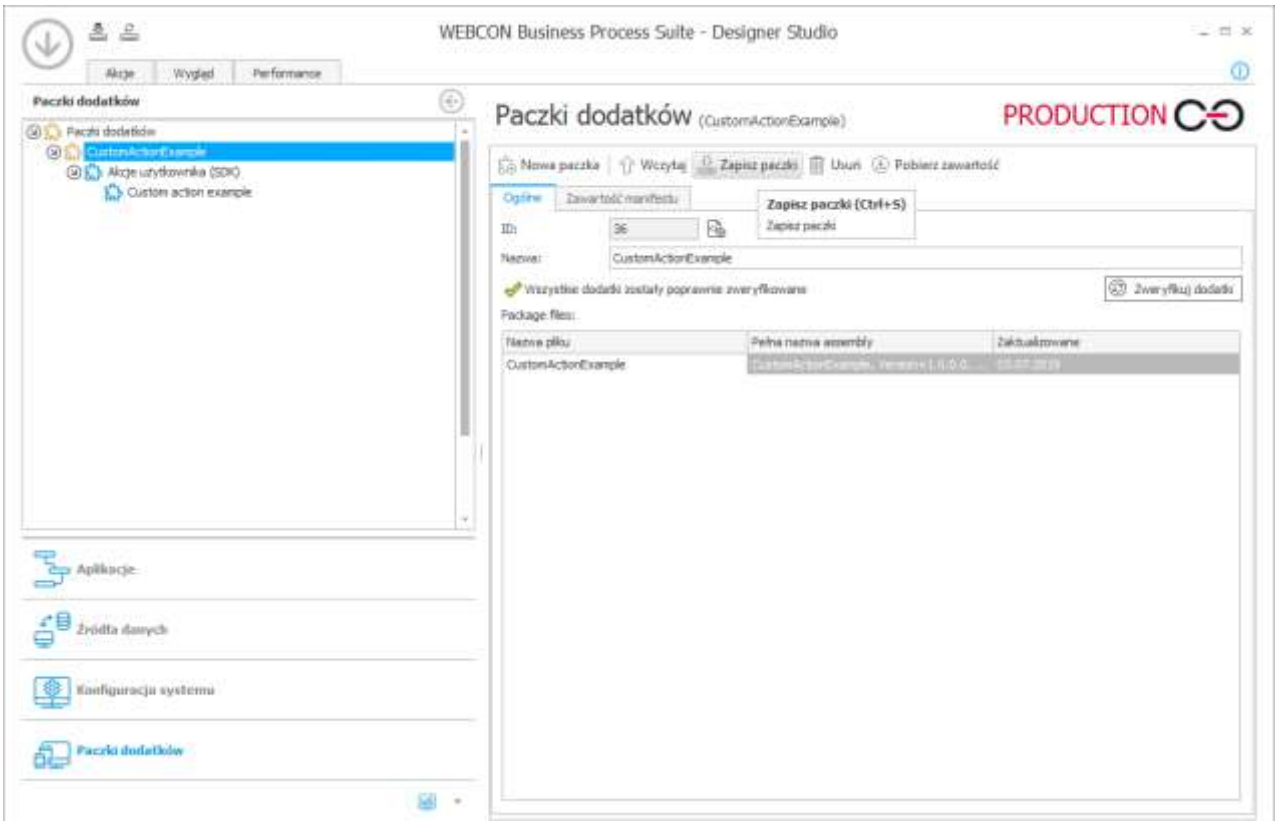


If we create new unregistered plugins during the migration, each record in the plugin manifest will need to have a GUID generated.

One of the ways to do this is to use Windows PowerShell and execute a static method: [guid]::NewGuid(). The generated value is then copied to the node in the manifest that represents the migrated plugin. Done.

2.2. Upgrading from version 2019.1 to 2019.4

The second type of migration takes place if we upgrade plugins from version 2019.1 or higher. Existing plugin packages will undergo the migration script which will upgrade them to be compatible with the newest version. Once you install the newest version of WEBCON BPS, save the plugin packages in the Plugins section of Designer Studio, extract the manifest files and place them in the associated project in Visual Studio. Once this is done, the plugins are ready for use.



2.3. Uniform plugin identifiers across multiple WEBCON BPS databases

In the case of plugins registered across multiple connected WEBCON BPS databases (DEV, TEST, PROD), multiple plugin manifests will exist with different identifiers for the same plugin, e.g.:

TEST database:


```
[
 {
  "Guid": "10bd08ad-a9a3-4004-91a9-94167fe3ee9e",
  "Name": "Custom action example",
  "Assembly": "CustomActionSample",
  "Class": " CustomActionSample.ChangeFormFieldValue",
  "Type": "CustomAction"
 }
]
```


PROD database:


```
[
 {
  "Guid": "760b455f-8a99-4469-95b9-dba0ded9e667",
  "Name": "Custom action example",
  "Assembly": "CustomActionSample",
  "Class": " CustomActionSample.ChangeFormFieldValue",
  "Type": "CustomAction"
 }
]
```


In order to substitute the TEST database GUIDs with those found in the PROD database, we must:

- First, modify the PROD database manifest and add an AlternativeGuid field containing the GUID from the TEST database.

For example:


```
[
 {
  "Guid": "760b455f-8a99-4469-95b9-dba0ded9e667",
```

```
  "AlternativeGuid ": "10bd08ad-a9a3-4004-91a9-94167fe3ee9e",
  "Name": "Custom action example",
  "Assembly": "CustomActionSample",
  "Class": " CustomActionSample.ChangeFormFieldValue",
  "Type": "CustomAction"
 }
]
```


- Once prepared, such a manifest along with the plugin package should be updated on the TEST database. This is done simply by uploading the new package containing the new manifest into WEBCON BPS Designer Studio.

Registered plugins will be associated with the appropriate entries in the manifest based on the AlternativeGuid field, and this value will then be copied to the main GUID field. From this point, a plugin will have one uniform identifiers across different databases, while the AlternativeGuid field will be ignored.


```
[
 {
  "Guid": "760b455f-8a99-4469-95b9-dba0ded9e667",
  "Name": "Custom action example",
  "Assembly": "CustomActionSample",
  "Class": " CustomActionSample.ChangeFormFieldValue",
  "Type": "CustomAction"
 }
]
```


If more databases exist (e.g. DEV), the process must be repeated for each additional database.

3. Changes to SDK

3.1. DataSourcesHelper – loading data from form fields and item list columns

From the class defining parameters loaded from form fields and item list columns (`GetDataTableFromFieldDataSourceParams`, `GetDataTableFromItemsListColumnDataSourceParams`), the following property has been removed: `GetDataTableRunningContextParams`.

It has been replaced by the property `Context`.

You can pass the context in which the plugin is to be run, or create a new context of a process, an instance, or an item list column.

- The code for loading data from form fields in previous versions:

```
var fieldDataSourceParams = new GetDataTableFromFieldDataSourceParams();
fieldDataSourceParams.FieldId = formFieldID;

fieldDataSourceParams.GetDataTableRunningContextParams = new
GetDataTableRunningContextParams(args.Context); //current context
//fieldDataSourceParams.GetDataTableRunningContextParams = new
GetDataTableRunningContextParams(documentInstance); //context of a specific document

var dt = DataSourcesHelper.GetDataTableFromFieldDataSource(fieldDataSourceParams);
```

Currently it is:

```
var fieldDataSourceParams = new GetDataTableFromFieldDataSourceParams();
fieldDataSourceParams.FieldId = formFieldID;

fieldDataSourceParams.Context = args.Context; //current context
//fieldDataSourceParams.Context = new ProcessContext(processID); //context of the specific
process
//fieldDataSourceParams.Context = new ProcessContext(documentInstance); //context of the
specific document

var dt = DataSourcesHelper.GetDataTableFromFieldDataSource(fieldDataSourceParams);
```

- The code for loading data from item list column in previous versions:

```
var itemsListDataSourceParams = new GetDataTableFromItemsListColumnDataSourceParams();
itemsListDataSourceParams.ItemsListColumnID = itemsListColumnID;

itemsListDataSourceParams.GetDataTableRunningContextParams = new
GetDataTableRunningContextParams(args.Context); //current context
//itemsListDataSourceParams.GetDataTableRunningContextParams = new
GetDataTableRunningContextParams(documentInstance); //context of a specific document

itemsListDataSourceParams.ItemsListRow = itemsListRow;
var dt = DataSourcesHelper.GetDataTableFromSubelementDataSource(itemsListDataSourceParams);
```

Currently it is:

```
var itemsListDataSourceParams = new GetDataTableFromItemsListColumnDataSourceParams();
itemsListDataSourceParams.ItemsListColumnID = itemsListColumnID;

itemsListDataSourceParams.Context = args.Context; //current context; if it contains a items list row you don't need to provide it separately
/itemsListDataSourceParams.Context = new ProcessContext(documentInstance, itemsListRow);
//context of the specific document and items list row

var dt = DataSourcesHelper.GetDataTableFromSubelementDataSource(itemsListDataSourceParams);
```

3.2. User control – changes to the method name

In the user control, the SharePoint interface component without its own data model (class `CustomFormFieldSPControl<TConfig>`), the method name has been changed from

```
void SetValue(SetControlParams<CustomFormFieldContext> args)
```

to

```
void SetControlValue(SetControlParams<CustomFormFieldContext> args)
```

3.3. Logic part – changes to the class name

The class name has been changed from `SPConnection` to `SharePointConnection` (namespace `WebCon.WorkFlow.SDK.Tools.Data.Model`)

3.4. Logic part – changes to the namespace

The namespace has been changed:

Previous name	Current name
WebCon.WorkFlow.SDK.Documents.ItemsLists	WebCon.WorkFlow.SDK.Documents.**Model**.ItemsLists
WebCon.WorkFlow.SDK.Documents.Fields	WebCon.WorkFlow.SDK.Documents.**Model**.Fields

3.5. Custom data sources – changes loading process IDs

For plugins like ‘custom data source’ written before version 2019.1.3., if you refer to the ProcessID, you must migrate the code according to the example below.

Previously:

```
public override DataTable GetData(SearchConditions searchConditions)
{
            var processId = Context.CurrentProcessID;

            …
 }
```

Currently:

```
public override DataTable GetData(SearchConditions searchConditions)
{
            var processId = Context.CurrentProcessID.HasValue ?
            Context.CurrentProcessID.Value : 0;

            …
}
```

4. Changes to REST API

4.1. New version of REST API 2.0

In the 2020.1 version, a new version of REST API 2.0 has been added.

4.2. Changes to transferring a transition path

In the Beta version, when compared to the 2019.1 version, the way of passing the transition path parameters has been changed, previously was:

POST, PUT */api/data/v2.0/db/{dbId}/elements? pathId={pathID}*

POST, PUT */api/data/v2.0/db/{dbId}/elements? pathGuid={ pathGuid}*

POST, PUT */api/data/v2.0/db/{dbId}/elements? defaultPath =1*

Currently it is:

POST, PUT */api/data/v2.0/db/{dbId}/elements? path={path}*

As a path value we give the ID, GUID or ‘default’ value.

This way of passing the transition path is also adopted in REST API 2.0.