WEBCON

Creating SDK plugins

1. Introduction 3
2. Architecture 4
3. SDK plugins.. 5
4. Creating plugins from a template in Visual Studio 6
5. Creating custom action 9
6. Example of custom action..10
7. SDK plugin registration 12
8. Types of SDK plugins..16
9. Example of plugin with interface 17

WEBCON Sp. z o.o.
ul. Skotnicka 252a, 30-394 Kraków
tel: +48 12 443 13 90, fax: +48 12 443 13 75

Sąd Rejonowy dla Krakowa Śródmieścia
XI Wydział Gospodarczy Krajowego Rejestru Sądowego
Wysokość kapitału zakładowego: 50 000 zł

NIP: 676-241-40-97
REGON: 121153997
KRS: 0000349649

www.webcon.pl

1. Introduction

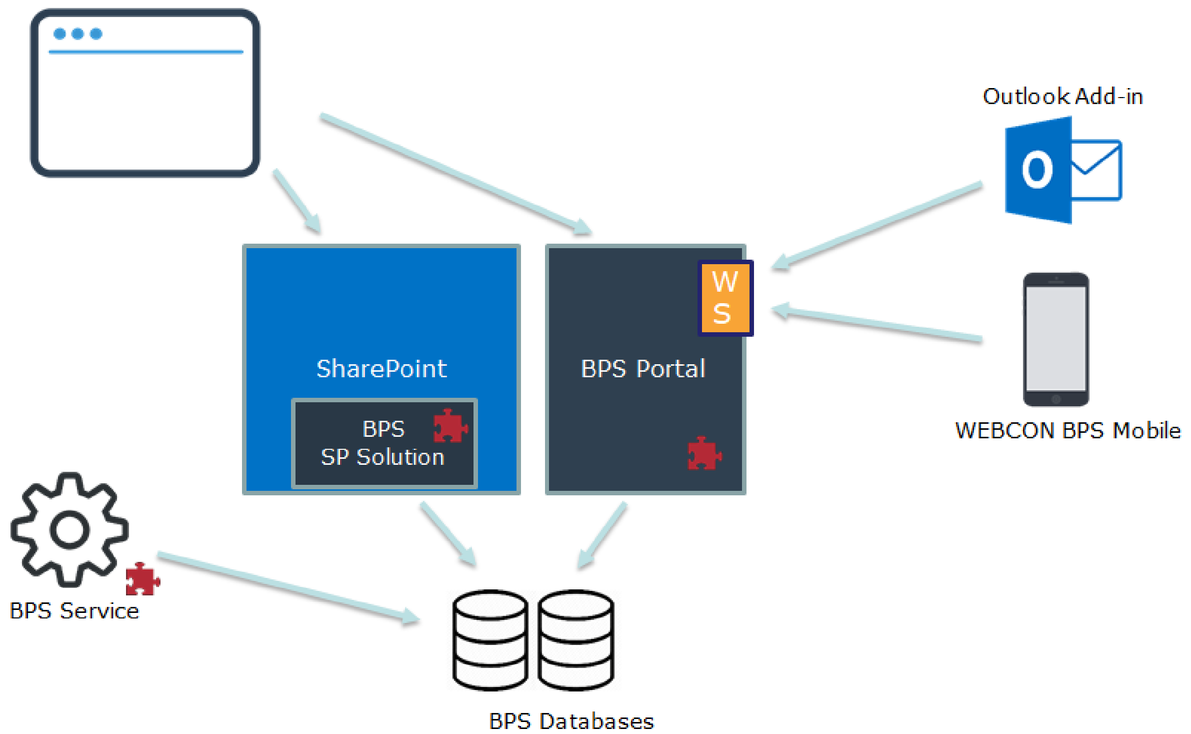
This document will show you the basics of SDK plugins and also how to create them.

2. Architecture

System architecture can assume two different modes, based on the installation type:

1. classic based on SharePoint platform and BPS Portal
2. standalone based on BPS Portal

The communication between different layers of the system has been presented on the diagram below:



Plugins can be run in three different environments: SharePoint, BPS Portal and BPS Service. Depending on your business scenario, you can write one logic SDK plugin and run it across all the environments mentioned above. When it comes to UI SDK plugins, at this moment, there are only base classes for SharePoint solutions.

3. SDK plugins

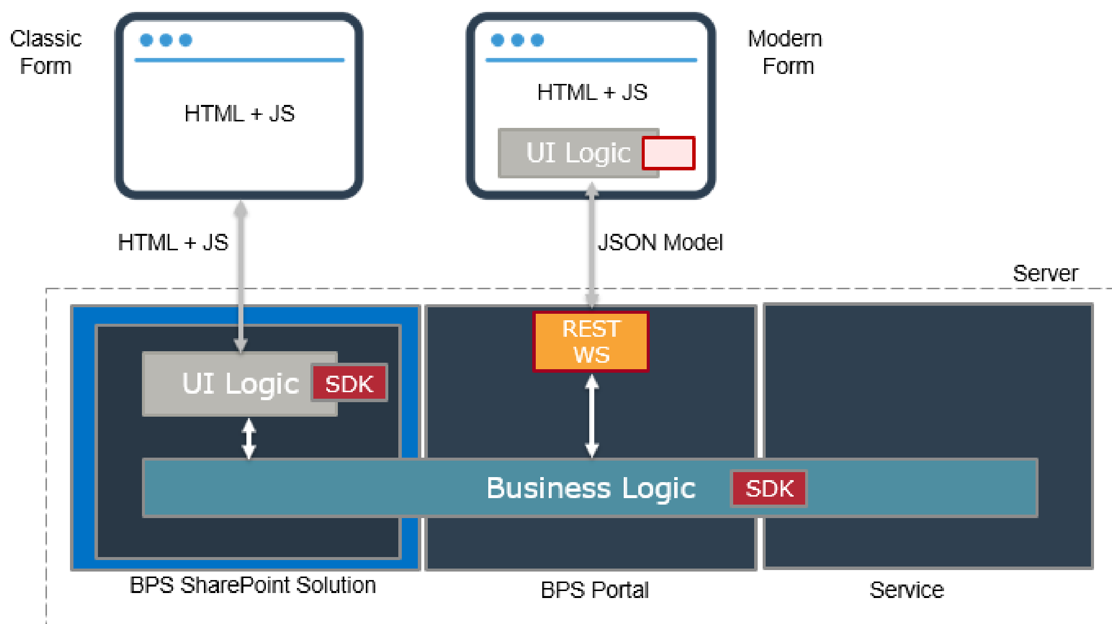
SDK plugins can be split into two parts – one associated with logic part of the control e.g. transferring data to an external system, and another one associated with displaying the control in a specific environment.

There is no requirement to create an interface for your plugins. Running the logic plugin is independent from the UI e.g. when you start an action on timeout, the interface part will not be run, but the logic part will be.

We have two SDK libraries:

1. **WebCon.WorkFlow.SDK.SP** which contains base classes for controls displayed in the classic SharePoint interface
2. **WebCon.Workflow.SDK** which contains base classes for logic parts of plugins, containers for data, configuration attributes, exceptions, and SDK tool classes

Base classes for controls displayed in the standalone interface will be available in the next version of WEBCON BPS.



4. Creating plugins from a template in Visual Studio

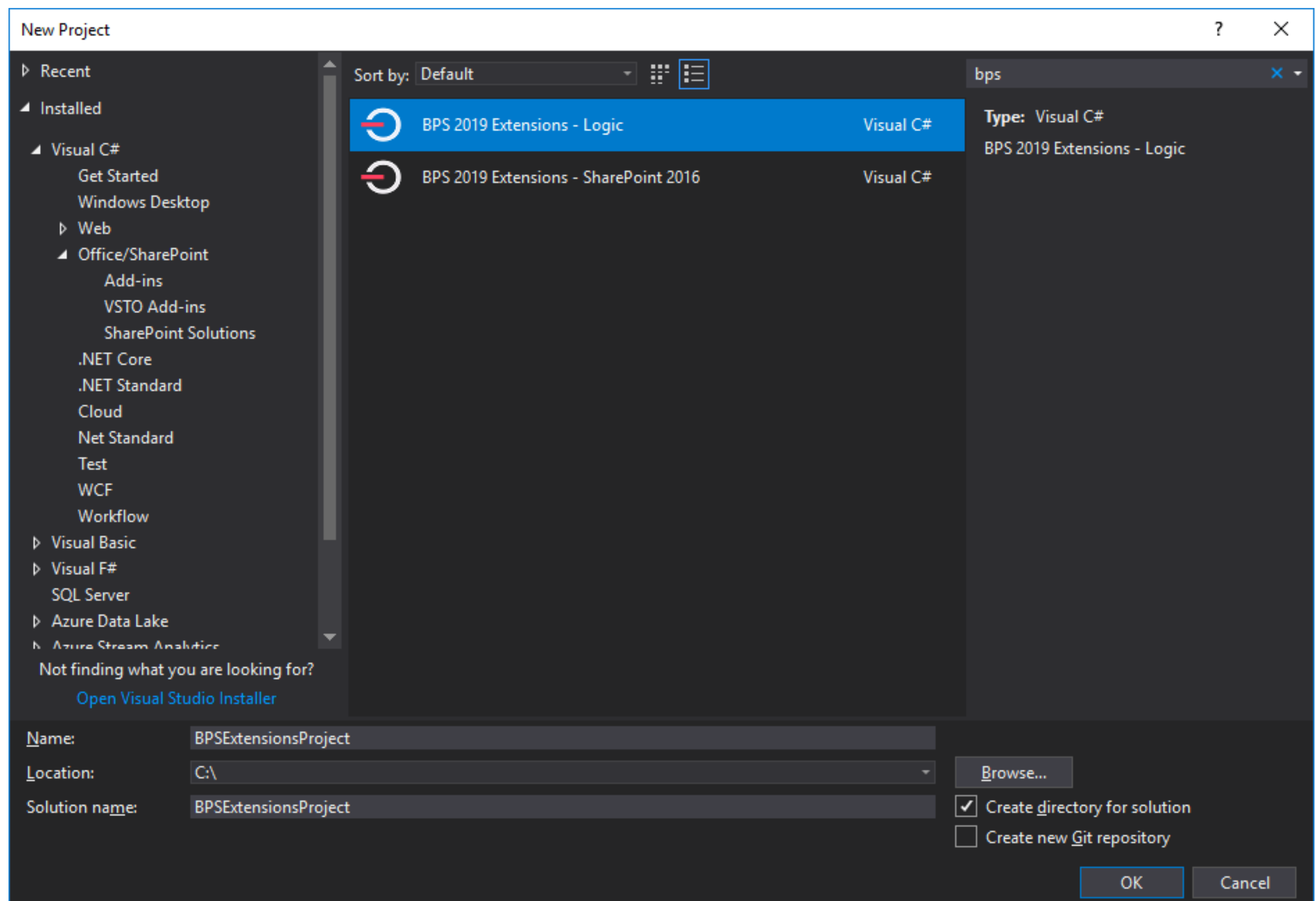
The process of creating the SDK plugin is very simple. You can create your projects manually using Class Library for the logic part, and SharePoint Project for the UI part, but you can also make use of provided templates that make this operation even more effortless.

At the time of writing this document, we provide templates for creating logic part of the plugin and for creating interface part of the plugin for SharePoint 2013, SharePoint 2016 and SharePoint 2019.

Inside the projects you can also make use of item template such as:

- BPS 2019 Custom Action
- BPS 2019 Custom Business Rule
- BPS 2019 Custom Control
- BPS 2019 Custom Control Logic
- BPS 2019 Custom Data Source
- BPS 2019 Custom Label Template
- BPS 2019 Form Field Extension
- BPS 2019 Form Field Extension Logic
- BPS 2019 Items List Extension
- BPS 2019 Items List Extension Logic

Project templates



Item templates

Add New Item - WebCon.BPS.HelloWorld.SDK

Installed
Visual C# Items
WPF
Code
Data
General
SQL Server
Storm Items
Web
Windows Forms
Workflow
Online

Sort by: Default

Template	Type
BPS 2019 Custom Action	Visual C# Items
BPS 2019 Custom Business Rule	Visual C# Items
BPS 2019 Custom Control	Visual C# Items
BPS 2019 Custom Control Logic	Visual C# Items
BPS 2019 Custom Data Source	Visual C# Items
BPS 2019 Custom Label Template	Visual C# Items
BPS 2019 Form Field Extension	Visual C# Items
BPS 2019 Form Field Extension Logic	Visual C# Items
BPS 2019 Items List Extension	Visual C# Items
BPS 2019 Items List Extension Logic	Visual C# Items

Search (Ctrl+E)

Type: Visual C# Items
BPS 2019 Custom Action

Name: CustomAction1.cs

Add Cancel

5. Creating custom action

You can create the logic part of the action with provided templates or by creating the Class Library project.

Then, in your project structure you have to create configuration class which will be used by yours custom action. Configuration class is just C# class which derives from **PluginConfiguration** and has public properties marked with specific attributes dependent on the result you want to achieve. This configuration will be available to set in action configuration in the WEBCON BPS Studio. To access configuration from your plugin you can call **base.Configuration**. One important thing to take into consideration is the fact that the logic and the UI part have to use the same configuration class.

The next step is to create custom action class. Again, it is just C# class which derives from generic class **CustomAction<T>**. Instead of **T** you write your configuration class.

In the logic part you can override few properties and methods. The most important methods in order to run the action are:

- **void Run(RunCustomActionParams args)**
- **void RunWithoutDocumentContext(RunCustomActionWithoutContextParams args)**

The main difference between these two lies where the action is triggered in the context of WEBCON BPS workflows. The second one is invoked by actions on timeout and cyclical actions. These actions are run outside of SharePoint and outside of the context of workflow instances. The first one is invoked in all other action triggers (action activation types).

In order to register your action in WEBCON BPS Studio you have to create manifest which is a JSON file which defines information about each of the plugins:

- Name
- Description
- Assembly and class plugins logic part
- Assembly and class plugins UI part
- Type
- Controls URL (ASCX)

When you use one of the templates, the sample JSON is generated automatically.

The last step is to create a package. The package is a zipped directory which consists of:

- Generated DLL files of your project
- JSON manifest
- Non-systemic DLL files used by plugins

The DLL files are versioned.

6. Example of custom action

WebServiceCallConfiguration.cs

```
using WebCon.WorkFlow.SDK.Common;
using WebCon.WorkFlow.SDK.ConfigAttributes;

namespace WebCon.BPS.HelloWorld.SDK
{
    public class WebServiceCallConfiguration : PluginConfiguration
    {
        // Configurable properties - properties that can be
        // configured in Designer Studio in the action configuration form
        [ConfigEditableText(
            DisplayName = "WebserviceURL",
            Description = "URL of vacation webservice",
            IsRequired = true)]
        public string WebServiceUrl { get; set; }      // URL to Vacation webservice
    }
}
```

WebServiceCallAction.cs

```
using System;
using System.Linq;
using WebCon.BPS.HelloWorld.SDK.WebServiceProxyStub;
using WebCon.WorkFlow.SDK.ActionPlugins;
using WebCon.WorkFlow.SDK.ActionPlugins.Model;

namespace WebCon.BPS.HelloWorld.SDK.CustomActions
{
    public class WebServiceCallAction : CustomAction<WebServiceCallConfiguration>
    {
        // Form fields identifiers in the process configuration
        private const int StartDateFormFieldID = 14;           // Form field that contains
start date of a vacation request
        private const int EndDateFormFieldID = 15;           // Form field that contains
end date of a vacation request
        private const int RequestorFormFieldID = 16;           // Form field that contains
requestor login of a vacation request
        private const int VacationIDFormFieldID = 17;           // Form field to store the
identifier that registered vacation obtained in HR external system

        public override void Run(RunCustomActionParams args)
        {
            try
            {
                // create web service client
                var client = new VacationWebServiceClient(Configuration.WebServiceUrl);
                // prepare input parameters - read them from the Form Fields
                var registerVacationParams = new RegisterVacationParams
                {
                    StartDate =
args.Context.CurrentDocument.DateTimeFields.GetByID(StartDateFormFieldID).Value.Value,
                    EndDate =
args.Context.CurrentDocument.DateTimeFields.GetByID(EndDateFormFieldID).Value.Value,
```

```
                    Requestor = 
args.Context.CurrentDocument.PeopleFields.GetByID(RequestorFormFieldID).Values.First().BpsI
D
                };
                // call web service method
                var result = client.RegisterVacation(registerVacationParams);
                // store the result in the Form Field

args.Context.CurrentDocument.TextFields.GetByID(VacationIDFormFieldID).Value = 
result.VacationID;
            }
            catch (Exception ex)
            {
                // when something goes wrong you can stop the workflow
                args.HasErrors = true;                  // Mark that there was error in the 
action
                args.Message = "Error: " + ex.Message;  // Prepare the message for end 
users
                args.LogMessage = ex.ToString();        // Prepare log entry
            }
        }
    }
}
```

Manifest JSON file

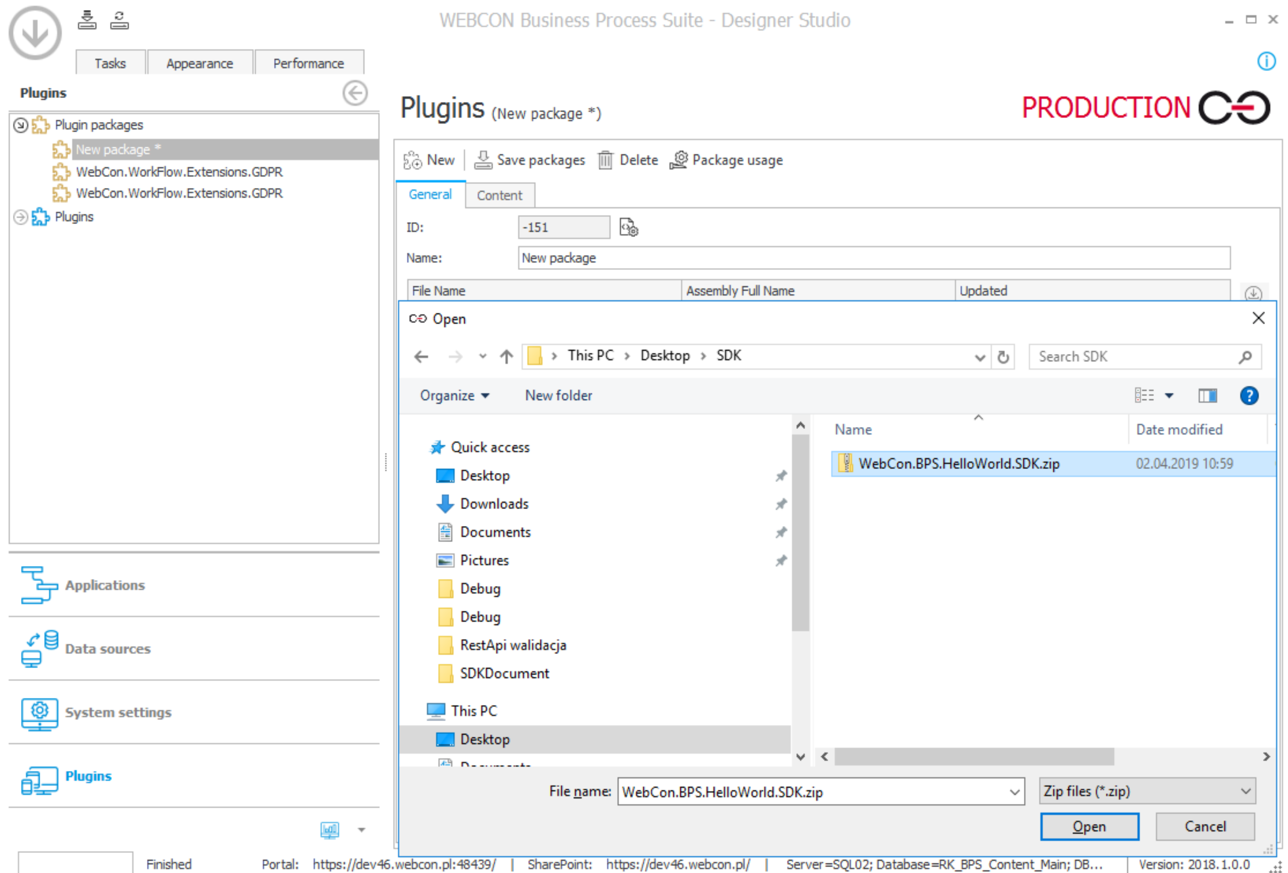
```
[
  {
    "Name": "Vacation webservice call",
    "Description": "Plugin for calling external webservice for registering vacation 
applications and returning vacation's ID.",
    "Assembly": "WebCon.BPS.HelloWorld.SDK",
    "Class": "WebCon.BPS.HelloWorld.SDK.CustomActions.WebServiceCallAction",
    "Type": "CustomAction"
  }
]
```

7. SDK plugin registration

You can register previously created package and SDK plugin in Plugins tab inside WEBCON BPS Studio.

After doing this you can attach your plugin to a specific action.

Package registration



Plugin registration

WEBCON Business Process Suite - Designer Studio

Tasks | Appearance | Performance

Plugins

- Plugin packages
- Plugins
  - Custom action (SDK)
  - Custom control (SDK)
  - Field customizations (SDK)
  - Item list customizations (SDK)
  - Label printout template (SDK)
  - Custom data source (SDK)
  - Business rule (SDK)

Applications

Data sources

System settings

Plugins

Plugins (*)

PRODUCTION

Choose plugin

Choose pacakge containing plugin

WebCon.BPS.HelloWorld.SDK

Choose plugin class:

Vacation webservice call

Plugin name:

Vacation webservice call

Plugin description

Plugin for calling external webservice for registering vacation applications and returning vacation's ID.

Assembly full name:

WebCon.BPS.HelloWorld.SDK, Version=1.0.0.0, Culture=neutral, PublicKeyToken=91b4d131e0682eb8

Plugin class:

WebCon.BPS.HelloWorld.SDK.WebServiceCallAction

OK | Cancel

On menu button | Recurrent action | On browser opening

On transition params | On save | On delete

On timeout

Refresh | Choose | Test

Finished | Portal: https://dev46.webcon.pl:48439/ | SharePoint: https://dev46.webcon.pl/ | Server=SQL02; Database=RK_BPS_Content_Main; DB... | Version: 2018.1.0.0

Action registration

HelloWorldWorkflow - Step edit: Start *

General | Forms | Paths | Actions | Analysis | Flow control | Subworkflows settings

Actions list

On entry
On exit *
WebServiceCallAction *
On timeout
On browser opening
Menu button
On path
Go to next step
Upon instance deleting
Upon instance saving
Attachments menu
On attachment add

Action settings

ID: -13
Active
Name: WebServiceCallAction
Documentation:
Action type: Run an SDK action
Template: <None> Configure template
Plugin (SDK): Vacation webservice call
Execution condition:

Configuration

Log execution
Configure

Previous step
Next step
Save process
Close

WEBCON

Action configuration

Configuration - Run an SDK action

Plugin for calling external webservice for registering vacation applications and returning vacation's ID.

Plugin properties

WebserviceURL *

URL of vacation webservice

Functions | Values | Objects

Enter text to search...

Name
- System fields
- Form fields
- Context variables
- Global constants
- Process constants

Switch all editors into advanced edit mode

Save | Cancel

8. Types of SDK plugins

As mentioned in chapter 3, there are following types of SDK plugins:

- CustomAction
- CustomBusinessRule
- CustomDataSource
- CustomFormField
- FormFieldExtension
- ItemListExtension
- CustomFormFieldSPControl
- FormFieldExtensionSPControl
- ItemListExtensionSP

9. Example of plugin with interface

To be able to use the created plugin with an interface you have to take two steps:

- create package with generated files of your project
- deploy WSP file containing ASCX control with associated JS, CSS files and DLL file with interface part to SharePoint

Webcon.BPS.SampleEN.DateRangeControl Class Library project

DateRangeConfig.cs

```
using WebCon.WorkFlow.SDK.Common;
using WebCon.WorkFlow.SDK.ConfigAttributes;

namespace WebCon.BPS.SampleEN.DateRangeControl.CustomControls
{
    public class DateRangeConfig : PluginConfiguration
    {
        [ConfigEditableText(DisplayName = "Date from", Description = "Database field where
\"date from\" will be stored")]
        public int DateFromDbID { get; set; }

        [ConfigEditableText(DisplayName = "Date to", Description = "Database field where
\"date to\" will be stored")]
        public int DateToDbID { get; set; }
    }
}
```

DateRangeValue.cs

```
using System;

namespace WebCon.BPS.SampleEN.DateRangeControl.CustomControls
{
    public class DateRangeValue
    {
        public DateTime? FromDate { get; set; }
        public DateTime? ToDate { get; set; }
    }
}
```

DateRangeLogic.cs

```
using WebCon.WorkFlow.SDK.Common.Model;
using WebCon.WorkFlow.SDK.FormFieldPlugins;
using WebCon.WorkFlow.SDK.FormFieldPlugins.Model;

namespace WebCon.BPS.SampleEN.DateRangeControl.CustomControls
{
    public class DateRangeLogic : CustomFormField<DateRangeConfig, DateRangeValue>
    {
        public override DateRangeValue LoadValue(LoadValueParams<CustomFormFieldContext>
args)
        {
            return new DateRangeValue()
            {
                FromDate =
args.Context.CurentDocument.DateTimeFields.GetByID(Configuration.DateFromDbID).Value,
                ToDate =
args.Context.CurentDocument.DateTimeFields.GetByID(Configuration.DateToDbID).Value
            };
        }

        public override void
OnBeforeElementSave(BeforeSaveParams<CustomFormFieldValueContextInfo<DateRangeValue>> args)
        {

args.Context.CurentDocument.DateTimeFields.GetByID(Configuration.DateFromDbID).SetValue(arg
s.Context.Value?.FromDate);

args.Context.CurentDocument.DateTimeFields.GetByID(Configuration.DateToDbID).SetValue(args.
Context.Value?.ToDate);
        }

        public override void
Validate(ControlValidationParams<CustomFormFieldValueContextInfo<DateRangeValue>> args)
        {
            if (!args.Context.CurrentField.IsRequired)
                return;
            if (args.Context.Value == null || args.Context.Value.FromDate == null ||
args.Context.Value.ToDate == null)
            {
                args.IsValid = false;
                args.ErrorMessage = "Date from and date to must contain values!";
                return;
            }
            if(args.Context.Value.FromDate > args.Context.Value.ToDate)
            {
                args.IsValid = false;
                args.ErrorMessage = "Date to cannot be earlier than date from!";
            }
        }
    }
}
```

WEBCON

Webcon.BPS.SampleEN.DateRangeControl SharePoint project

DateRangeControl.ascx

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="DateRangeControl.ascx.cs"
Inherits="WebCon.BPS.SampleEN.DateRangeControl.SP.Layouts.DateRangeControl,
$SharePoint.Project.AssemblyFullName$"%>
<%@ Assembly Name="WebCon.WorkFlow.SDK, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=c30f1f18c194ceba" %>
<%@ Assembly Name="WebCon.WorkFlow.SDK.SP, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=c30f1f18c194ceba" %>
<%@ Register assembly="Microsoft.SharePoint, Version=15.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" namespace="Microsoft.SharePoint.WebControls"
tagprefix="cc1" %>
<tr>
    <td>
        <table border ="1">
            <tr>
                <td>
                    FROM
                </td>

                <td>
                    <cc1:DateTimeControl ID="DateTimeControl1" runat="server" />
                </td>
            </tr>
            <tr>
                <td>
                    TO
                </td>
                <td>
                    <cc1:DateTimeControl ID="DateTimeControl2" runat="server" />
                </td>
            </tr>
        </table>
    </td>
</tr>
```

DateRangeControl.ascx.cs

```
using WebCon.BPS.SampleEN.DateRangeControl.CustomControls;
using WebCon.WorkFlow.SDK.Documents.Model.Base;
using WebCon.WorkFlow.SDK.SP.FormFieldPlugins;
using WebCon.WorkFlow.SDK.SP.FormFieldPlugins.Model;

namespace WebCon.BPS.SampleEN.DateRangeControl.SP.Layouts
{
    public partial class DateRangeControl : CustomFormFieldSPControl<DateRangeConfig,
DateRangeValue>
    {

        public override DisplayNamePlace FormFieldDisplayNamePlace =>
DisplayNamePlace.Beside;

        public override DateRangeValue
GetControlValue(GetControlValueParams<CustomFormFieldContext> args)
        {
            if (!DateTimeControl1.IsDateEmpty && !DateTimeControl2.IsDateEmpty)
                return new DateRangeValue()
                {
                    FromDate = DateTimeControl1.SelectedDate,
                    ToDate = DateTimeControl2.SelectedDate
                };
            else
                return null;
        }

        public override void SetControlValue(SetControlValueParams<CustomFormFieldContext,
DateRangeValue> args)
        {
            ValidateReadOnly(args.Context.CurrentField);
            if (args.ValueToSet != null)
            {
                if (args.ValueToSet.FromDate != null)
                    DateTimeControl1.SelectedDate = args.ValueToSet.FromDate.Value;
                if (args.ValueToSet.ToDate != null)
                    DateTimeControl2.SelectedDate = args.ValueToSet.ToDate.Value;
            }
        }

        private void ValidateReadOnly(FormElement currentField)
        {
            DateTimeControl1.Enabled = currentField.IsEditable;
            DateTimeControl2.Enabled = currentField.IsEditable;
        }
    }
}
```

WEBCON

Manifest JSON file

```
[
  {
    "Name": "DateRangeControl",
    "Description": "Control for selecting date range",
    "SPAssembly": "WebCon.BPS.SampleEN.DateRangeControl.SP",
    "Assembly": "WebCon.BPS.SampleEN.DateRangeControl",
    "SPClass": "WebCon.BPS.SampleEN.DateRangeControl.SP.DateRangeControl",
    "Class": "WebCon.BPS.SampleEN.DateRangeControl.DateRangeLogic",
    "SPUrl": "/_layouts/15/WebCon/DateRangeControl.ascx",
    "Type": "FormFieldExtension"
  }
]
```

Project structure