WEBCON

BPS 2019.1

Changes to SDK & API

Topic:	BPS 2019.1 – Changes to SDK & API
Author:	Tomasz Batko
Creation date:	2018-12-01
Modification date:	2020-12-10 15:22:51

1. Introduction

The following document contains a list of changes in WEBCON BPS SDK and API when compared to version 2017.1. These changes break code compatibility or change how existing code works.

2. Changes to SDK

2.1. New SDK libraries

New SDK libraries were created:

WebCon.WorkFlow.SDK.SP – contains base classes of controls displayed in the Classic interface (SharePoint);

WebCon.WorkFlow.SDK – contains base classes with the logic part of plugins (executed independently from interface), and SDK tool classes (previously found under WebCon.WorkFlow.SDK.Logic)

WebCon.WorkFlow.SDK.Logic has been removed, whereas WebCon.WorkFlow.SDK.Objects should no longer be use in WEBCON BPS 2019.

2.2. Plugin base classes

In WEBCON BPS 2019, the following base classes were changed:

2.2.1. Configuration base class

Configuration properties of plugins must be placed in separate classes, which inherit from a new base class: `PluginConfiguration`.

2.2.2. Generic base classes

The base classes of individual plugin types have become generic

```
CustomAction<TConfig>: where TConfig:PluginConfiguration
```

In order to implement a plugin in WEBCON BPS 2019, it is necessary to create at least two classes:

- Class containing configuration properties of plugins
- Classes containing plugin logic – inheriting from a base class of the given type, and being a generic class to the configuration properties class.

You can refer to a plugin's configuration from its logic with the property: `Configuration`.

Previously:

```
public class MyAction : WFCustomAction
{
    [ConfigEditableText(DisplayName = "My property")]
    public string Prop1 { get; set; }

    public override void Run(RunCustomActionParams args)
    {
        var p = Prop1;
    }
}
```

Currently:

```
public class MyAction : CustomAction<MyConfig>
{
    public override void Run(RunCustomActionParams args)
    {
        var p = Configuration.Prop1;
    }
}

public class MyConfig: PluginConfiguration
{
    [ConfigEditableText(DisplayName = "My property")]
    public string Prop1 { get; set; }
}
```

2.2.3. Separating the logic and interface of controls

Components that have their own interface: custom control and field customizations, as well as those that can modify the interface: Item list customizations, have now been split into two parts:

- Interface – responsible for displaying the control in the given environment. Currently, the only interface section available is that for SharePoint environments.
- Logic – responsible for the operating logic of the control: e.g. sending data to external systems. The logic can be connected to the interface, but they are activated independently, e.g. if a workflow instance is launched by a timeout action, the interface part will not be activated, but the logic will.

Both parts must use the same configuration class.

Previously:

```
   public class MyItemsListCustomization : WFSubelementsCustomization
    {
        [ConfigEditableText(DisplayName = "My property")]
        public string Prop1 { get; set; }

        public override void OnCellDisplay(CellDisplayParams args)
        {
            args.IsReadOnly = IsRowReadonly(args.SubElementsValues, args.ConfigRowIndex,
Prop1);
        }

        public override void OnBeforeElementSave(BeforeSaveParams args)
        {
            SendToCrm(args.SubElementsValues);
        }

    }
```

Currently:

```
    public class MyConfig : PluginConfiguration
    {
        [ConfigEditableText(DisplayName = "My property")]
        public string Prop1 { get; set; }
    }

    public class MyItemsListCustomization :
WebCon.WorkFlow.SDK.ItemListPlugins.ItemsListExtension<MyConfig>
    {
        public override void OnBeforeElementSave(BeforeSaveParams<ItemListExtensionContext>
args)
        {
            SendToCrm(args.Context.CurrentItemsList);
        }

    }

    public class MyItemsListCustomizationSP :
WebCon.WorkFlow.SDK.SP.ItemListPlugins.ItemListExtensionSP<MyConfig>
    {

        public override void OnCellDisplay(CellDisplayParams args)
        {
            args.IsEditable = !IsRowReadonly(args.Context.CurentItemRow,
Configuration.Prop1);
        }

    }
```

2.3. Plugin manifest

The plugin name and description have been removed from the base class. Starting with version 2019, the names and descriptions are to be provided in the plugin manifest file. The plugin manifest is a JSON file that lists the plugins available in a project, as well as defining the connection between interface and logic parts of each control.

Sample manifest:

```
[
  {
    "Type": "CustomAction",
    "Name": "New Action",
    "Description": "",
    "Assembly": "Webcon.BPSExt.NewSDK",
    "Class": "Webcon.BPSExt.NewSDK.NewAction"
  },
  {
    "Type": "CustomDataSource",
    "Name": "My data source",
    "Description": "",
    "Assembly": "Webcon.BPSExt.NewSDK",
    "Class": "Webcon.BPSExt.NewSDK.TestDataSource"
  },
  {
    "Type": "FormFieldExtension",
    "Name": "Test FormFieldExt",
    "Description": "",
    "Assembly": "Webcon.BPSExt.NewSDK",
    "Class": "Webcon.BPSExt.NewSDK.Layouts.AttExtLogic",
    "SPUrl": "/_layouts/15/WebCon/attributeExtensions/AttributeExtension1.ascx"
  },
  {
    "Type": "CustomFormField",
    "Name": "Test custom ctrl",
    "Description": "",
    "Assembly": "Webcon.BPSExt.NewSDK",
    "Class": "Webcon.BPSExt.NewSDK.Layouts.CustomControlLogic",
    "SPUrl": "/_layouts/15/WebCon/CustomControls/CustomControCustomActionl1.ascx"
  },
  {
    "Type": "ItemListExtension",
    "Name": "Items list Logic",
    "Description": "",
    "Assembly": "Webcon.BPSExt.NewSDK",
    "Class": "Webcon.BPSExt.NewSDK.ItemsListCustomizations.SampleLogic",
    "SPAssembly": "Webcon.BPSExt.NewSDK.SP",
    "SPClass": "Webcon.BPSExt.NewSDK.SP.ItemsListCustomizations.Sample"
  }
]
```

Available parameters:

Type	Plugin type. Possible values: CustomAction, CustomBusinessRule, CustomDataSource, CustomFormField, FormFieldExtension, ItemListExtension, CustomLabelPrint
Name	Plugin name
Description	Plugin description
Assembly	Name of library containing plugin (or its logic part)
Class	Full name of plugin class (plugin logic part)
SPAssembly	Name of library with the interface (SharePoint) part of the plugin. Used only for ItemListExtension
SPClass	Full name of the class with the interface (SharePoint) part of the plugin. Used only for ItemListExtension
SPUrl	SharePoint control address, in the following format: `"/_layouts/15…"`. Used for CustomFormField and FormFieldExtension

2.4. Project structure in Visual Studio

When migrating to version 2019, it is necessary to distinguish two projects:

- Project with logic parts – class library type

Should contain plugins without an interface (e.g. actions, data sources), and the logic parts of controls

- Project w interface parts (‘SharePoint – BPS Extensions’ or ‘SharePoint – Project’ type)

Should contain all controls (ASCX and their base classes), and resources used by controls (JS and CSS files)

One of the projects should also contain a JSON file with the plugin manifest.

2.5. Plugin packages

Version 2019 introduces the plugin package mechanism. The package is a ZIP file containing all non-system DLL that are mandatory for the plugin logic to work, and the manifest file describing the plugins (one JSON file, its name is irrelevant).

Before registering new plugin in Designer Studio, first create and save a plugin package.

Plugin packages are registered in the 'Plugins' section of Designer Studio, the same place where regular plugins are registered.

Once a plugin package is registered and saved, you can register and select a plugin that belongs to this package.

The plugin package mechanism requires that library files be versioned – it is not possible to register two packages with DLLs of the exact same full assembly name, but different contents. Different contents means that the library version must also be different.

Plugin packages are now also included in the Import-Export mechanism in Designer Studio.

Plugins that are ASCX controls (CustomFormField and FormFieldExtension) must be made up of two parts:

- WSP containing the interface part of the plugin
- Package with the plugin DLL and its manifest (if a DLL of the logic part of the plugin exists)

WSP is not included in the Import-Export mechanism, it must be deployed on SharePoint separately.

After migrating your database to version 2019, plugin packages will be created automatically (without content). The plugins themselves must be migrated manually:

- Migrate the plugins to the new SDK version
- Prepare a manifest describing your plugins
- Create a ZIP file containing libraries and a JSON manifest
- Upload the new content to the empty packages created during the migration

2.6. Migrating WebCon.WorkFlow.SDK.Objects

Below is a list of plugin types available in WebCon.WorkFlow.SDK.Objects and their corresponding updates in version 2019.

2.6.1. WFCustomAction

New base class for custom actions is CustomAction<TConfig>. In version 2019, the cyclical action invocation (without workflow instance context) has been separated from actions that are invoked in the context of a workflow instance (timeout, on path transition, menu button).

To create an action that is activated without the context of a workflow instance, overwrite method:

```
RunWithoutDocumentContext(RunCustomActionWithoutContextParams args)
```

If an action is intended to be available as a cyclical action (without instance context) and also as a Menu button, invoke method RunWithoutDocumentContext from the Run method.

```
public override void Run(RunCustomActionParams args)
 {
     RunWithoutDocumentContext(new
RunCustomActionWithoutContextParams(args));
 }

 public override void
RunWithoutDocumentContext(RunCustomActionWithoutContextParams args)
 {
     HandleActionWithoutDocumentContext(args);
 }
```

Below is a list of custom action properties in version 2017, and their corresponding properties in version 2019:

Version 2017	Version 2019
WFCustomAction	CustomAction<TConfig>
WFCustomAction.WorkFlowObject.SelectedPath WFCustomAction.WorkFlowObject.StartPath WFCustomAction.WorkFlowObject.StartStep WFCustomAction.WorkFlowObject.PreviousStep WFCustomAction.WorkFlowObject.CurrentStep WFCustomAction.WorkFlowObject.NextStep	Available in the Run method as args.TransitionInfo

WFCustomAction.WorkFlowObject.ElementCurrent	Available in the Run method as args.Context.CurrentDocument
WFCustomAction.WorkFlowObject.WorkFlowDefinition WFCustomAction.WorkFlowObject.WorkFlow WFCustomAction.WorkFlowObject.IsElementNew WFCustomAction.WorkFlowObject.ModifiedBy WFCustomAction.WorkFlowObject. IsInAdminMode	Available in the Run method as args.Context or args.Context.CurrentDocument
WFCustomAction.WorkFlowObject.AssignedPersons WFCustomAction.WorkFlowObject.AssignedPersonsDW	Available in the Run method as args.TransitionInfo.TasksToAdd
WFCustomAction.SelectedPath	Available in the Run method as args.TransitionInfo.CurrentPathID
SelectedAction	Available in the Run method as args.Context.CurrentActionID
NextStepId	Available in the Run method as args.TransitionInfo. NextStepID
ElementCurrent	Available in the Run method as args.Context.CurrentDocument
PluginName	Moved to the manifest
PluginDescription	Moved to the manifest
GetSubelementsControl(…)	Available in the Run method as args.Context.CurrentDocument.ItemsLists

2.6.2. WFCustomBusinessRule

New base class is: CustomBusinessRule<TConfig>

Version 2017	Version 2019
WFCustomBusinessRule	CustomBusinessRule<TConfig>
PluginName	Moved to the manifest
PluginDescription	Moved to the manifest

2.6.3. WFCustomDataSource

New base class is: CustomDataSource<TConfig>

Version 2017	Version 2019
WFCustomDataSource	CustomDataSource<TConfig>
PluginName	Moved to the manifest
PluginDescription	Moved to the manifest
TestQuery(…)	Virtual method with a default implementation invoking GetData
GetColumns()	The class name of returned elements is now: `DataSourceColumn`

2.6.4. WFExtensionLabelFormat

New base class is: CustomLabelPrint<TConfig>

Version 2017	Version 2019
WFExtensionLabelFormat	CustomLabelPrint<TConfig>
PluginName	Moved to the manifest
PluginDescription	Moved to the manifest
TemplateCustom	Available in the OnGetLabelCustom method as parms.TemplateCustom
TextQuantity	Available in the OnGetLabelCustom method as parms.TextQuantity
PrinterType	Available in the OnGetLabelCustom method as parms.PrinterType
PrintTemplateEncoding	Available in the OnGetLabelCustom method as parms.PrintTemplateEncoding

2.6.5. WFAttributeExtension

The form field extension plugin in version 2019 has been divided into logic and interface parts. The base classes for these parts are accordingly:

- **Logic:** `FormFieldExtension<TConfig>` (from WebCon.WorkFlow.SDK)
- **Interface:** `FormFieldExtensionSPControl<TConfig>` (from WebCon.WorkFlow.SDK.SP)

If the value passed between the logic and interface parts is a complex value (e.g. WEBCON BPS only contains the ID of the record, and additional data is loaded from an external system), then it is possible to use base classes and provide a generic value class. These are:

`FormFieldExtension<TConfig,TValue>` (from WebCon.WorkFlow.SDK)
`FormFieldExtensionSPControl<TConfig,TValue>` (from WebCon.WorkFlow.SDK.SP).

`TValue` is any type used to pass values.

If no type was provided, you can only use one part (logic or interface).

Classes that are components of controls are represented in the plugin package manifest.

Both plugin parts need to have a matching `TConfig` (if `TValue also exists`).

If the interface part is being used, it must be also uploaded to SharePoint (e.g. via deploy WSP).

In version 2019, in the ASCX file of a form field control, it is needed to register the WebCon.WorkFlow.SDK and WebCon.WorkFlow.SDK assemblies.

Previously:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind=" SimpleFormField.ascx.cs"
Inherits="WebCon.BpsExt.Example.Layouts.AttributeExtensions.SimpleFormField,
$SharePoint.Project.AssemblyFullName$"%>

<asp:TextBox runat="server" ID="testTxt1" />
```

Currently:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind=" SimpleFormField.ascx.cs"
Inherits="WebCon.BpsExt.Example.Layouts.AttributeExtensions.SimpleFormField,
$SharePoint.Project.AssemblyFullName$"%>

<%@ 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" %>

<asp:TextBox runat="server" ID="testTxt1" />
```

Below is a list of form field extension properties in version 2017, and their corresponding properties in version 2019:

Version 2017	Version 2019
WFAttributeExtension	FormFieldExtension<TConfig> i FormFieldExtensionSPControl<TConfig>
PluginName	Moved to the manifest
PluginDescription	Moved to the manifest
InitializeControlOutsideWebForm()	Method removed, behavior must be recreated in the logic part.
CheckSaveRestrictions(…)	VerifySaveRestrictions(…)
HasEmptyValue	IsFieldEmpty(…)
FieldName	In various methods as: args.Context.CurrentFormField.DbColumnName
CompanyId	In various methods as: args.Context.CurentDocument.CompanyID
IsReadOnly	In various methods as: args.Context.CurrentFormField.IsEditable
IsRequired	In various methods as: args.Context.CurrentFormField.IsRequired
ElementConfiguration.Element	In various methods as: args.Context.CurentDocument
ElementConfiguration. IsElementNew	In various methods as: args.Context.IsNewDocument
ElementConfiguration.IsAdminMode	In various methods as: args.Context.IsAdminMode
ElementConfiguration.CurrentStep	In various methods as: args.Context.CurentDocument.StepID
ElementConfiguration.DocumentType	In various methods as: args.Context.CurentDocument.DocumentTypeID
SetValue(…)	Available in the interface part
GetValue()	Available in the interface part
SetFocus()	Available in the interface part. Virtual method returns false by default.
GetSetValueJavascript()	Available in the interface part

GetGetValueJavascript()	Available in the interface part
GetEnableFieldJavascript()	Available in the interface part
GetDisableFieldJavascript()	Available in the interface part

2.6.6. WFExtensionControl

The Custom form field (aka. Custom control) plugin in plugin in version 2019 has been divided into logic and interface parts. The base classes for these parts are accordingly:

- **Logic:** `CustomFormField<TConfig>` (from WebCon.WorkFlow.SDK)
- **Interface:** `CustomFormFieldSPControl<TConfig>` (from WebCon.WorkFlow.SDK.SP)

If the control must pass a value between the logic and interface parts, then it is possible to use base classes and provide a generic value class. These are:

`CustomFormField <TConfig,TValue>` (from WebCon.WorkFlow.SDK)

`CustomFormField <TConfig,TValue>` (from WebCon.WorkFlow.SDK.SP)

`TValue` is any type used to pass values.

If no type was provided, you can skip the logic part and use only the interface part.

Classes that are components of controls are represented in the plugin package manifest.

Both plugin parts need to have a matching `TConfig` (if `TValue also exists`).

If the interface part is being used, it must be also uploaded to SharePoint (e.g. via deploy WSP).

In version 2019, in the ASCX file of a form field control, it is needed to register the WebCon.WorkFlow.SDK and WebCon.WorkFlow.SDK assemblies.

Previously:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="SimpleCustomFormField.ascx.cs"
Inherits="WebCon.BpsExt.Example.SP.Layouts.CustomControls.SimpleCustomFormField,
$SharePoint.Project.AssemblyFullName$" %>

<asp:TextBox runat="server" ID="testTxt1" />
```

Currently:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="SimpleCustomFormField.ascx.cs"
Inherits="WebCon.BpsExt.Example.SP.Layouts.CustomControls.SimpleCustomFormField,
$SharePoint.Project.AssemblyFullName$" %>

<%@ 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" %>

<asp:TextBox runat="server" ID="testTxt1" />
```

Below is a list of custom control properties in version 2017, and their corresponding properties in version 2019:

Version 2017	Version 2019
WFExtensionControl	CustomFormField <TConfig> and CustomFormFieldSPControl <TConfig>
PluginName	Moved to the manifest
PluginDescription	Moved to the manifest
InitializeControlOutsideWebForm()	Method removed, behavior must be recreated in the logic part.
CheckSaveRestrictions(…)	VerifySaveRestrictions(…)
AttributeDefinition	In various methods as: args.Context.CurrentField
FieldName	In various methods as: args.Context.CurrentField.DisplayName
FormFieldDisplayNamePlace	Available in the interface part
IsReadOnly	In various methods as: args.Context.CurrentField.IsEditable
IsRequired	In various methods as: args.Context.CurrentField. IsRequired
IsAdminMode	In various methods as: args.Context.IsAdminMode
ElementCurrent	In various methods as: args.Context.CurentDocument
IsElementNew	In various methods as: args.Context.IsNewDocument
OnDelete()	OnBeforeElementDelete(…)

2.6.7. WFSubelementsCustomization

The Item list extensions plugin (aka. Item list customization) plugin in version 2019 has been divided into logic and interface parts. The base classes for these parts are accordingly:

- **Logic:** `ItemsListExtension<TConfig>` (from WebCon.WorkFlow.SDK)
- **Interface:** `ItemListExtensionSP<TConfig>` (from WebCon.WorkFlow.SDK.SP)

It is possible to use two connected parts, or one part individually.

Classes that are components of controls are represented in the plugin package manifest.

Both plugin parts need to have a matching TConfig.

Both the logic and interface parts should be included in the plugin package. It is recommended for them to be in separate libraries.

Version 2017	Version 2019
WFSubelementsCustomization	ItemsListExtension<TConfig and ItemListExtensionSP<TConfig>
PluginName	Moved to the manifest
PluginDescription	Moved to the manifest
OnRunBeforeExitActions(…)	Method removed, please replace it with an action or business rule
OnCanFinishTask(…)	Method removed, please replace it with an action or business rule
OnCanGoToNextStep(…)	Method removed, please replace it with an action or business rule
OnAddTasks(…)	Method removed, please replace it with an action or business rule
OnHeaderCellDisplay(…)	Available in the interface part
OnCellDisplay(…)	Available in the interface part
OnFooterCellDisplay(…)	Available in the interface part
OnButtonsDisplay(…)	Available in the interface part: OnRowButtonsDisplay()
OnDataLoaded(…)	Available in the interface part
OnVerifySaveRestrictions(…)	VerifySaveRestrictions(…)
OnControlValidation(…)	Validate(…)

OnColumnsCustomization(…)	Available in the interface part: OnItemsListCustomization
OnAddCustomButtons(…)	Available in the interface part: OnInitCustomButtons
OnHandleCustomCommand(…)	Available in the interface part
OnValidateImportedRow(…)	Available in the interface part
OnAfterImportData(…)	Available in the interface part

Properties of individual methods were also changed:

Version 2017	**Version 2019**
CellDisplayParams.CurrentCell	CellDisplayParams.CurrentHtmlCell
CellDisplayParams.DataRowIndex	CellDisplayParams.Context. CurentItemRow – a data row is returned instead of the index
CellDisplayParams .ConfigRowIndex	CellDisplayParams.Context. CurentColumn – a column is returned instead of the index
CellDisplayParams.IsReadOnly	CellDisplayParams.IsEditable
DisplayCustomizationParams.ElementCurrent	In various methods as: args.Context.CurrentDocument
BaseCustomizationParams.SubElementsValues	In various methods as: args.Context.CurrentItemsList.Rows
BaseCustomizationParams.SubElementsConfiguration	In various methods as: args.Context.CurrentItemsList.Columns
IsAdminMode	In various methods as: args.Context.IsAdminMode
AttributeDefinition	In various methods as: args.Context.CurrentItemsList

2.6.8. WFExtensionSubElemsControl

Plugin removed, please replace with a Custom form field plugin.

2.6.9. WFFlowCustomization

Plugin removed, please replace with a business rule.

2.7. Migration of WebCon.WorkFlow.SDK.Logic

The library: WebCon.WorkFlow.SDK.Logic was removed, its contents were moved to the library: WebCon.WorkFlow.SDK, below is a list of moved classes:

Version 2017	Version 2019
DocumentManager	WebCon.WorkFlow.SDK.Documents. DocumentsManager
AccentsHelper	WebCon.WorkFlow.SDK.Tools.Other.TextHelper
AttachmentsHelper	WebCon.WorkFlow.SDK.Documents. DocumentAttachmentsManager
CompanyStructureHelper	WebCon.WorkFlow.SDK.Tools.Users.CompanyStructureHelper
ConnectionsHelper	WebCon.WorkFlow.SDK.Tools.Data.ConnectionsHelper
DetailConfigsHelper	Helper removed
ElementsHelper	WebCon.WorkFlow.SDK.Documents. DocumentsManager i WebCon.WorkFlow.SDK.Tools.Other.TextHelper
FileGenerationHelper	WebCon.WorkFlow.SDK.Tools.Files. FileGenerationHelper
SqlExecutionHelper	WebCon.WorkFlow.SDK.Tools.Data.SqlExecutionHelper
SubelementsConfigHelper	Helper removed
TimeZoneHelper	WebCon.WorkFlow.SDK.Tools.Other.TimeZoneHelper
Logger	WebCon.WorkFlow.SDK.Tools.Log. Logger
XmlLogBuilder	WebCon.WorkFlow.SDK.Tools.Log. XmlLogBuilder
UserDataProvider	WebCon.WorkFlow.SDK.Tools.Users.UserDataProvider
SecurityManager	WebCon.WorkFlow.SDK.Tools.Security.SecurityManager
WorkFlowManager	Manager removed, please use: DocumentsManager
DataSourcesHelper	WebCon.WorkFlow.SDK.Tools.Data. DataSourcesHelper

2.8. Working outside of SharePoint context

In previous versions, all plugins were always activated in the context of SharePoint. With the introduction of version 2019, it is possible to invoke certain plugins outside of the context of SharePoint, these include:

- Cyclical and timeout actions
- Plugins activated in WEBCON BPS Portal
- Plugins activated by Rest API (e.g. action invocation on path transition)

Please keep these changes in mind when migrating your plugins.

Below is a list of most commonly used SharePoint elements and how to migrate them:

- Lists, document libraries – Access to SharePoint objects should be handled with ClientObjectModel
- `SPSecurity.RunWithElevatedPrivileges(…) – please replace with: WebCon.WorkFlow.SDK.Tools.AppPoolContext.RunWithElevatedPrivileges(…).`

  For invocations on SharePoint, both methods are handled in the context of the application pool, and for invocations outside of SharePoint `AppPoolContext.RunWithElevatedPrivileges(…)` will not attempt to reference a non-existent SharePoint context.
- SPContext.Current.Web.Url – create a variable in the plugin configuration. A value can be defined as a global constant in Designer Studio and pass it to the plugin using the variable.

2.9. Switching to Modern form display mode

A full and complete migration of plugins to version 2019 is possible for **Classic on SharePoint** or **Classic on BPS Portal** form display modes.

Forms that use **Modern on BPS Portal** display mode have the following limitations:

- Custom form field plugin is not supported
- Form field extensions and Item list extensions will only use their logic parts.
- Custom actions do not register JavaScripts on the form (property: CustomJavascript)

2.10. Changes to component behavior

2.10.1. Change of behavior in methods: `SetFieldValue` and `SetValue` for int type fields.

In version 2019, these methods will return an error when attempting to assign a value that is not an int type. In previous versions, a null value would be assigned to such fields.

To replicate the previous behavior, we can use:

```
var field = args.CurrentDocument.Fields.GetByID(FieldID);
if (field is IntegerField)
	if (int.TryParse(value, out int result))
		field.SetValue(result);
	else
		field.SetValue(null);
else
	field.SetValue(value);
```

2.10.2. Change of behavior in attribute: `ConfigEditableText` for fields that aren't nullable.

In version 2019, these fields will return an error when attempting to assign an empty string to them. To replicate the previous behavior, we can mark these fields as nullable.

Previously:

```
[ConfigEditableText(DisplayName = "Can set null")]
public int Prop1 { get; set; }
```

Currently:

```
[ConfigEditableText(DisplayName = "Can set null")]
public int? Prop1 { get; set; }
```

3. Changes to Web API

3.1. Changes to how default vales work on existing workflow instances

In WEBCON BPS 2019, the way in which default values on form fields work has been altered – they are set only on form in edit mode (i.e. checked out for editing). When loading instance WEB API via method GetElementById(…), the form is loaded in read-only mode, which means that default values for form fields will not be set on it.

To force default values to be set on instances that are loaded, we can:

- If using API for a version lower than 2017_1, then update it to this version.
- In the GetElementById(…) method invocation, set the EditMode parameter to: true

Invocation example:

```
var client = CreateClient();
var requestForm = client.GetElementById(new BPSApi.GetElementByIdParams()
{
       ElementId = documentID,
       EditMode = true
});


client.MoveElementToNextStep(new BPSApi.MoveElementToNextStepParams()
{
       Element = requestForm,
       PathId = acceptancePathID
});
```

WebAPI 8_3 does not support setting default values when operating on existing workflow instances.