Installation and configuration

Last updated by | Hieu Le | May 5, 2025 at 10:31 AM GMT+2

Installing and configuring the add-on

The Epinova AI Assistant is easy to get started with. Most functionality is automatically set up after registering the module, but there are also custom attributes that can be used to further configure the module to work with each sites content modules.

For Optimizely CMS 12

1. Install the `Epinova.CMS.AiAssistant` package from the Optimizely nuget feed .
2. Add the following to Startup.cs

```
using Epinova.CMS.AiAssistant;
...
services.AddAiAssistant();
```

3. Configure the api key for the Ai Assistant add-on in app settings, for instance in appSettings.json

```
"Epinova": {
    "AiAssistantOptions": {
      "ApiKey": "your-api-key"
    }
}
```

4. Ensure that the file `Epinova.CMS.AiAssistant.zip` are created under `modules/_protected/Epinova.CMS.AiAssistant` after building the project.

```
modules
  - _protected
      - Epinova.CMS.AiAssistant
          Epinova.CMS.AiAssistant.zip
```

For Optimizely CMS 11

1. Install the `Epinova.CMS.AiAssistant` package from the Optimizely nuget feed .
2. Configure the AI Assistant API routes in Global.asax.cs

```
using Epinova.CMS.AiAssistant;
....
public class EPiServerApplication : EPiServer.Global
{
    protected override void RegisterRoutes(RouteCollection routes)
    {
        ...
        base.RegisterRoutes(routes);
        routes.RegisterAiAssistantRoutes();
        ...
    }
}
```

3. Configure the api key for the AiAssistant add-on in web.config

```
<appSettings>
    ...
    <add key="Epinova-AiAssistantOptions-ApiKey" value="your-api-key" />
</appSettings>
```

4. Ensure that the file `Epinova.CMS.AiAssistant.zip` and the `Views` folder are created under `modules/_protected/Epinova.CMS.AiAssistant` after building the project. They should also be included in the project.

```
modules
  - _protected
      - Epinova.CMS.AiAssistant
          - Views
              - Default
                  Index.cshtml
              - Shared
                  _ShellLayout.cshtml
                _ViewStart.cshtml
                Web.config
            Epinova.CMS.AiAssistant.zip
```

TLS 1.3 Validation

If you encounter the error **"The SSL connection could not be established"** during development, it may indicate that your operating system does not support **TLS 1.3**.

To bypass TLS validation for development purposes (not recommended for production), you can temporarily add the following code:

Add in Startup.cs (Optimizely CMS 12)

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
#if DEBUG
    ServicePointManager.ServerCertificateValidationCallback +=
        (sender, certificate, chain, sslPolicyErrors) => true;
#endif
    ...
}
```

Add in Global.asax.cs (Optimizely CMS 11)

```
protected void Application_Start(object sender, EventArgs e)
{
#if DEBUG
    ServicePointManager.ServerCertificateValidationCallback +=
        (s, cert, chain, sslPolicyErrors) => true;
#endif
    ...
}
```

⚠️ **Warning**: Disabling certificate validation is insecure and should only be used in a controlled development environment.

For more information about supported TLS protocol versions, refer to Microsoft's official documentation:
Protocols in TLS/SSL (Schannel SSP)

[Optional] Configuration for meta title, meta description, introduction, image alt text

Configure the property for generating the meta title and meta description

- You can configure fields that are used for meta title and meta description to be able to automatically suggest content based on the content types main content. This is done by adding the **AiMetaTitleGenerator** and **AiMetaDescriptionGenerator** attributes to the respective properties with a reference to the main content. Main content can be any type , like an `XHTMLString` or `ContentArea` property.
- If a `ContentArea` is used, the AI Assistant will extract string properties from its contained content items. To exclude specific properties within a content item or entire content item types from this extraction process for use in prompts, apply the **AiExcludeFromPrompt** attribute.

```
public virtual XhtmlString MainBody { get; set; }

public virtual ContentArea ContentArea { get; set; }

[AiMetaTitleGenerator(nameof(MainBody), nameof(ContentArea))]
public virtual string MetaTitle { get; set }

[AiMetaDescriptionGenerator(nameof(MainBody), nameof(ContentArea))]
public virtual string MetaDescription { get; set }
```

- The default property combination type is `PropertyCombinationType.And` when using multiple sources. This means that all content from the sources is combined into the prompt. When `PropertyCombinationType.Or` is used, the sources are processed in order, skipping any empty sources.

```
public virtual XhtmlString MainBody { get; set; }

public virtual ContentArea ContentArea { get; set; }

[AiMetaTitleGenerator(PropertyCombinationType.Or, nameof(MainBody), nameof(ContentArea))]
public virtual string MetaTitle { get; set }

[AiMetaDescriptionGenerator(PropertyCombinationType.Or, nameof(MainBody), nameof(ContentAre
public virtual string MetaDescription { get; set }
```

Configure the property for generating the introduction

The `AiIntroductionGeneratorAttribute` is used to automatically generate an introduction for a content type by extracting relevant information from specified properties. This attribute utilizes AI to create meaningful introductions based on the main content of a page.

Basic usage

If no parameters are provided, the introduction will be generated using all specified properties (`PropertyCombinationType.And`), with a default maximum length of **500** characters.

```
public virtual XhtmlString MainBody { get; set; }

public virtual ContentArea ContentArea { get; set; }

[AiIntroductionGenerator(nameof(MainBody), nameof(ContentArea))]
public virtual string Introduction { get; set; }
```

- The AI will extract relevant information from **both** `MainBody` and `ContentArea` to generate the introduction.

Setting a maximum length

If you want to **limit** the introduction to a specific number of characters, you can specify a `maxLength` parameter:

```
[AiIntroductionGenerator(300, nameof(MainBody), nameof(ContentArea))]
public virtual string Introduction { get; set; }`
```

- This ensures that the introduction does not exceed **300 characters**.

Using Property Combination Type

By default, `PropertyCombinationType.And` is used, meaning content from all source properties is combined.
If `PropertyCombinationType.Or` is used, sources are processed **in order**, stopping when a non-empty value is found.

```
[AiIntroductionGenerator(300, PropertyCombinationType.Or, nameof(MainBody), nameof(ContentArea)
public virtual string Introduction { get; set; }
```

- If `MainBody` has content, it will be used for the introduction.
- If `MainBody` is empty, `ContentArea` will be used instead.

AiExcludeFromPrompt attribute usage

To prevent certain properties or entire content items from being included in the AI prompt, use the `AiExcludeFromPrompt` attribute.

- For example, to exclude entire content of an `HeroBlock` content type from the extracted string, apply the **AiExcludeFromPrompt** attribute on the content type.

```
[AiExcludeFromPrompt]
public class HeroBlock : BlockData
{
    public virtual ImageBlock MainImage { get; set; }

    public virtual string Heading { get; set; }

    public virtual string Subheading { get; set; }
}
```

- For example, to exclude `MainImage` property content of an `HeroBlock` content type from the extracted string, apply the **AiExcludeFromPrompt** attribute on the property.

```
    [AiExcludeFromPrompt]
    public virtual ImageBlock MainImage { get; set; }

    public virtual string Heading { get; set; }

    public virtual string Subheading { get; set; }
```

Configure the property for generating image alt text

You can configure the AI Assistant to be able to automatically generate alt texts for an image. Since Optimizely does not contain a built in way to handle alt texts, you need to connect the alt text and image properties. This is done by connecting your properties using the **AiAltTextGenerator** attribute:

```
    [UIHint(UIHint.Image)]
    public virtual ContentReference Source { get; set; }

    [AiAltTextGenerator(nameof(Source))]
    public virtual string AltText { get; set; }
```

Configure the properties in `ImageData` for generating image alt text.

```
public class ImageMediaData : ImageData
{
    [Display(Name = "Alternate text in English")]
    AiAltTextGenerator(Language = "en")]
    public virtual string EnglshAltText { get; set; }

    Display(Name = "Alternate text in Swedish")]
    AiAltTextGenerator(Language = "se")]
    public virtual string SwedishAltText { get; set; }
}
```

Configure for `SeoInformation` in Optimizely Commerce

`SeoInformation` is a built-in block in Optimizely Commerce that includes `Title` and `Description` properties, representing SEO metadata for catalog entries and nodes. Since these properties cannot be decorated with custom attributes directly, we introduce an alternative approach to attach `MetaTitleGenerator` and `MetaDescriptionGenerator` logic using an initialization module.

Example: Registering Generators in Initialization Module

```
using Epinova.CMS.AiAssistant;
using Epinova.CMS.AiAssistant.Builders;

[InitializableModule]
public class AiInitialization : IInitializableModule
{
    public void Initialize(InitializationEngine context)
    {
            // Register AI generators for all GenericVariant content
        var variantBuilder = AiBuilder.ForContentType<GenericVariant>();
        variantBuilder
            .With(x => x.SeoInformation.Title)
            .Use<AiMetaTitleGeneratorAttribute>()
            .Configure(x => x.SourcePropertyNames, [nameof(GenericVariant.Description)]);

        variantBuilder
            .With(x => x.SeoInformation.Description)
            .Use<AiMetaDescriptionGeneratorAttribute>()
            .Configure(x => x.SourcePropertyNames, [nameof(GenericVariant.Description)]);
    }

    public void Uninitialize(InitializationEngine context)
    {
    }
}
```

Registering `SeoInformation` with AI Assistant

To integrate `SeoInformation` into the AI Assistant gadget panel in the Editorial UI, override the `SeoInformation` property from `EntryContentBase` in your `GenericVariant` class and decorate it with the `[AiAssistant]` attribute:

```
[CatalogContentType(DisplayName = "Generic Variant", GUID = "1aaa2c58-c424-4c37-81b0-77e76d254e
public class GenericVariant : VariationContent
{
     ...
     [AiAssistant]
     public override SeoInformation SeoInformation { get; set; }
     ...
}
```

Adding custom AI options

Developers have the capability to create customized AI options and display them on the UI.

Create custom tone

1. Create a class that implements `IAiTone` interface.

```
public class ProfessionalTone : IAiTone
{
    public string Name => "Professional";

    public int Order => 10;
}
```

2. Register the new tone in Startup.cs

```
services
    .AddAiAssistant()
    .AddTone<ProfessionalTone>();
```

Create custom format

1. Create a class that implements `IAiFormat` interface.

```
public class SalePitchFormat : IAiFormat
{
    public string Name => "Sale Pitch";

    public int Order => 10;
}
```

2. Register the new format in Startup.cs

```
services
   .AddAiAssistant()
   .AddFormat<SalePitchFormat>();
```

Create custom length

1. Create a class that implements `IAiLength` interface.

```
public class MediumLength : IAiLength
{
    public string Name => "Medium";

    public int Order => 5;
}
```

2. Register the new length in Startup.cs

```
services
    .AddAiAssistant()
    .AddLength<MediumLength>();
```

Create custom prompt

To create a custom AI prompt, follow these steps:

1. Implement the `IAiPrompt` interface

Create a class that implements `IAiPrompt`. This class defines the AI service type, prompt behavior, and UI options.

Example: IntroductionPrompt implementation

```
using EPiServer.ServiceLocation;
...
[ServiceConfiguration(typeof(IAiPrompt), Lifecycle = ServiceInstanceScope.Scoped)]
public class IntroductionPrompt : IAiPrompt
{
    public AiServiceType AiServiceType => AiServiceType.TextGeneration;

    public string InstructionType => AiInstructionTypes.SuggestionFromSources;

    public string Instruction => "[Unique instruction name]";

    /// <summary>
    /// Show as Summarize on the UI.
    /// </summary>
    public string InstructionDisplayName => AiInstructions.Summarize;

    public InstructionAvailability InstructionAvailability => InstructionAvailability.ShowOnFoc

    public bool ShowTextArea => false;

    public bool ShowTone => true;

    public bool ShowFormat => false;

    public bool ShowLength => false;

    public virtual string BuildPromptText(PromptModel model)
    {
        var sb = new StringBuilder("Write a concise ");

        if (!string.IsNullOrEmpty(model.Tone))
        {
            sb.Append($"and {model.Tone} ");
        }

        sb.Append($"introduction in {model.Language} ");

        if (model.MaxLength is not null)
        {
            sb.Append($"under {model.MaxLength} characters ");
        }

        sb.Append("based on the following text: ");

        return sb.ToString();
    }
}
```

- Use `[ServiceConfiguration]` attribute to register the prompt generator in the service container with a lifecycle of `Transient`, `Scoped` or `Singleton`.
- **AiServiceType (enum)**: Defines the types of AI services available for processing user requests.
  - **`TextGeneration`** : Used for generating, summarizing, or transforming text-based content.
  - **`Vision`** : Used for analyzing and interpreting images.
- **InstructionType (string)**: Defines different AI instruction types for generating content.
  - **`AiInstructionTypes.Suggestion`** : Generates text with configurable tone, format, and length.
  - **`AiInstructionTypes.SuggestionFromSources`** : Generates text based on specified source properties from the model.
  - **`AiInstructionTypes.SuggestionFromImages`** : Generates text based on analyzed content from images.

    - **`AiInstructionTypes.FixSpellingAndGrammar`** : Corrects spelling and grammar errors in the provided text.
    - **`AiInstructionTypes.Translation`** : Translates text into a selected language.
- **Instruction (string)**: The unique name of the instruction, ensuring distinct identification.
- **InstructionAvailability (enum)**: Defines UI behavior for displaying instructions.
    - **`InstructionAvailability.Show`** : The instruction is always visible in the UI.
    - **`InstructionAvailability.ShowOnFocus`** : The instruction is visible only when the user focuses on the associated field.
    - **`InstructionAvailability.Hidden`** : The instruction is hidden and not displayed in the UI.
- **ShowTextArea (bool)**: Determines whether the input field is displayed.
- **ShowTone (bool)**: Determines whether the tone selection is displayed.
- **ShowFormat (bool)**: Determines whether the format selection is displayed.
- **ShowLength (bool)**: Determines whether the length selection is displayed.

Implements the **BuildPromptText** method to construct AI prompt dynamically. The **PromptModel** contains the data submitted by the user, triggered when the user clicks the **Generate Draft** button.

- **Text (string)**: The input text used as the basis for AI-generated content.
- **Language (string)**: The language in which the content should be generated.
- **Tone (string?)**: The tone of the generated content (e.g., Professional, Informational).
- **Format (string?)**: The format of the generated content (e.g., Paragraph, Summary, Report).
- **Length (string?)**: The length of the generated content (e.g., < 150 chars, 500-1000 chars, or < 4000 chars).
- **MaxLength (int?)**: The maximum number of characters allowed for the generated content.
- **IsCustomPrompt (bool)**: Indicates whether the prompt is custom, identified by a `#` at the beginning of the prompt text.

2. Create a Custom Attribute that inherits `AiAssistantAttribute`

Extend `AiAssistantAttribute` to define the **source properties** (AI input) and **target properties** (UI activation upon user focus).

Example: AiIntroductionGeneratorAttribute implementation

```
public class AiIntroductionGeneratorAttribute : AiAssistantAttribute
{
    public AiIntroductionGeneratorAttribute(
        int maxLength,
        PropertyCombinationType sourcePropertyCombinationType,
        params string[] sourcePropertyNames)
    {
        MaxLength = maxLength;
        SourcePropertyCombinationType = sourcePropertyCombinationType;
        SourcePropertyNames = sourcePropertyNames;
        Prompt = typeof(IntroductionPrompt);
    }

    public AiIntroductionGeneratorAttribute(
        int maxLength,
        params string[] sourcePropertyNames) : this(maxLength, PropertyCombinationType.And, sou
    {
    }

    public AiIntroductionGeneratorAttribute(
        params string[] sourcePropertyNames) : this(500, PropertyCombinationType.And, sourcePro
    {
    }
}
```

- **MaxLength (int)**: Allows setting a maximum character length for the generated text.
- **SourcePropertyCombinationType (PropertyCombinationType)**: Defines how multiple source properties are combined when generating content. Supports `PropertyCombinationType` values such as **AND** and **OR** logic. For more details, refer to the **Configure the property for generating the meta title and meta description** section.
- **SourcePropertyNames (string[])**: Specifies the properties used as input for AI-generated introductions.
- **Prompt (Type)**: Associates with a specific AI prompt for text generation.