



Semantic Kernel for C#

2024 .NET version 1.0 map

.NET

Get packages

Core kernel

Add AI services

Connectors

Add plugins

Plugins

Add filters and telemetry

Add telemetry

Build the kernel

Core kernel

Add memory

Memory

Create prompts

Prompts and templates

Chat completion

Chat completion

1 Get packages

Core kernel

<https://aka.ms/sk/kernel>

```
//Import the NuGet package the first time into your project
#r "nuget: Microsoft.SemanticKernel"

//Use the package
using Microsoft.SemanticKernel;
using Kernel = Microsoft.SemanticKernel.Kernel;

//Create a new Kernel builder
IKernelBuilder builder = Kernel.CreateBuilder();
```

2 Add AI services

Connectors

<https://aka.ms/sk/aiservices>

```
// Semantic Kernel allows you to add and swap out different AI services depending on your needs. In addition to Azure OpenAI and OpenAI, Semantic Kernel supports Google Gemini, MistralAI, Ollama, local models, and more

// Azure OpenAI example
builder.AddAzureOpenAIChatCompletion(AOAI_DEP_NAME, AOAI_ENDPOINT, AOAI_KEY);

// OpenAI example
builder.AddOpenAIChatCompletion(OAI_MODEL_NAME, OAI_KEY);

// other modalities
builder.AddAzureOpenAIAudioToText(AOAI_DEP_NAME, AOAI_ENDPOINT, AOAI_KEY);
builder.AddAzureOpenAITextToAudio(AOAI_DEP_NAME, AOAI_ENDPOINT, AOAI_KEY);
builder.AddAzureOpenAITextToImage(AOAI_DEP_NAME, AOAI_ENDPOINT, AOAI_KEY);

// other services
builder.AddGoogleAIGeminiChatCompletion(GG_DEP_NAME, GG_DEP_KEY);
builder.AddMistralChatCompletion(MAI_MODEL_ID, MAI_KEY);
builder.AddOllamaChatCompletion(OLLAMA_MODEL_ID, OLLAMA_URI);
builder.AddOnnxRuntimeGenAIChatCompletion(ONNX_MODEL_ID, ONNX_MODEL_PATH);
```

3 Add plugins

Plugins

<https://aka.ms/sk/plugins>

```
// Plugins are a way to extend the functionality of the kernel. Plugins are added differently depending on where they are stored.

// Import of OpenAPI (most common)
var plugin = await OpenApiKernelPluginFactory.CreateFromOpenApiAsync(
    pluginName: "WeatherForecast",
    uri: new Uri($"http://localhost:5134/swagger/v1/swagger.json")
);
builder.Plugins.Add(plugin);

// Import from Type
builder.Plugins.AddFromType<TimeInformation>();

class TimeInformation
{
    [KernelFunction]
    [Description("Get the current time in UTC")]
    public string GetCurrentUtcTime() => DateTime.UtcNow.ToString("R");
}

// Add from Object
builder.Plugins.AddFromObject(new MyPlugin());

// Import from File Directory
builder.Plugins.AddFromPromptDirectory("path/to/plugins");
```

4 Add filters and telemetry

Add telemetry

<https://aka.ms/sk/filters>

```
// Set the level of tracing for each service
ILogger logger = LoggerFactory.Create(builder =>
builder.AddConsole()).CreateLogger("SemanticKernel");

// Setup logger using the telemetry package and attach to service
builder.Services.AddSingleton(logger);
builder.Services.AddSingleton<IPromptRenderFilter, PromptRenderLoggingFilter>();

class PromptRenderLoggingFilter : IPromptRenderFilter
{
    private readonly ILogger _logger;
    public PromptRenderLoggingFilter(ILogger logger) => _logger = logger;

    public async Task OnPromptRenderAsync(PromptRenderContext context,
Func<PromptRenderContext, Task> next)
    {
        await next(context);
        _logger.LogInformation($"Prompt: {context.RenderedPrompt}");
        context.RenderedPrompt = ValidateWithPromptShields(context.RenderedPrompt);
    }
}
```

5 Build the kernel

Core kernel

```
// Building the kernel attaches everything from the previous steps
Kernel kernel = builder.Build();
```

6 Add memory

Memory

<https://aka.ms/sk/memory>

```
// Semantic Kernel offers several memory store connectors to vector databases that you can use to store and retrieve information. Including Azure AI Search, Azure SQL Database, Azure CosmosDB, Chroma, DuckDB, Milvus, MongoDB Atlas, Pinecone, Postgres, Qdrant, Redis, Sqlite, Weaviate and more.

class Glossary
{
    [VectorStoreRecordKey]
    public ulong Key { get; set; }

    [VectorStoreRecordData]
    public required string Term { get; set; }

    [VectorStoreRecordData]
    public required string Definition { get; set; }

    [VectorStoreRecordVector(1536)]
    public ReadOnlyMemory<float> DefinitionEmbedding { get; set; }
}

// Construct and in-memory vector store, substitute any data connector here
var vectorStore = new InMemoryVectorStore();

// Get or create a collection
var collection = vectorStore.GetCollection<ulong, Glossary>("skglossary");
await collection.CreateCollectionIfNotExistsAsync();

// Create an embeddings generation service
var textEmbeddingService = new
AzureOpenAITextEmbeddingGenerationService(AOAI_EMBEDDING_DEP_NAME, AOAI_ENDPOINT, AOAI_KEY);

// Generate embeddings
var entries = Glossary.ContentEntriesLists();
var tasks = entries.Select(async entry =>
{
    entry.DefinitionEmbedding = await textEmbeddingService.GenerateEmbeddingAsync(entry.Term);
});
await Task.WhenAll(tasks);

// Upsert the entries into the collection and return their keys
var upsertKeys = await Task.WhenAll(entries.Select(async entry =>
{
    return await collection.UpsertAsync(entry);
})));

// Search the collection using a vector search
var searchString = "What is Semantic Kernel?";
var searchVector = await textEmbeddingService.GenerateEmbeddingAsync(searchString);
var searchResults = await collection.VectorizedSearchAsync(searchVector);
var resultRecord = await searchResults.Results.FirstAsync();

Console.WriteLine($"Search result (score: {resultRecord.Score}):
'{resultRecord.Record.Definition}'");
```

7 Create prompts and invoke

Prompts and templates

<https://aka.ms/sk/prompts>

```
// Prompts are a way to interact with the kernel using natural language. Prompts can be used to ask questions, get information, or execute functions.

// Invoke a basic prompt (this prompt will call a function)
var result = await kernel.InvokePromptAsync("Tell me about GenAI");
Console.WriteLine(result.ToString());

// Templated prompt using handlebars
result = await kernel.InvokePromptAsync("The current time is
{{TimeInformation.GetCurrentUtcTime}}.");

// Templated prompt with Kernel arguments
KernelArguments arguments = new() {{"topic", "Dogs"}};
result = await kernel.InvokePromptAsync("Tell me about {{topic}}", arguments);

// Define settings for the prompt, this setting will allow the prompt to automatically execute functions
OpenAIPromptExecutionSettings settings = new () {
    FunctionChoiceBehavior = FunctionChoiceBehavior.Auto()
};

result = await kernel.InvokePromptAsync("How long until Christmas? Explain you thinking.",
new(settings));
```

8 Chat completion

Chat completion

<https://aka.ms/sk/chat>

```
// ChatCompletionService is a common way to interact with the models
var chatService = kernel.GetRequiredService<IChatCompletionService>();
var chatHistory = new ChatHistory("You are a librarian, expert about books");

// Add a user message
chatHistory.AddUserMessage("Hi, I'm looking for book suggestions");

// Get a response from the service
var reply = await chatService.GetChatMessageContentAsync(chatHistory);
chatHistory.Add(reply);

// Or stream a response
await foreach (StreamingChatMessageContent chatUpdate in
chatService.GetStreamingChatMessageContentsAsync(chatHistory)) {
    Console.Write(chatUpdate.Content);
}
```

