*DRAFT*

Connected Services SDK Overview

Service providers use the Connected Services SDK to create Visual Studio extensions that let app developers integrate the services into their app without leaving the Visual Studio IDE.

The Connected Services SDK gives the app developer a well-known entry point for adding your service to their projects, but gives you the service provider complete control over the configuration process, from simply adding an authentication key to the app config file, to adding NuGets and references, to creating or modifying code and other project files. The provider extension can be targeted to any language or project type supported by Visual Studio.

Visual Studio comes with several Connected Services pre-installed, or the app developer can add to the installed providers using the Visual Studio Gallery. After the extension is installed by the app developer, they select it from the Add Connected Services dialog, providing them with a well-known path to integrate the service.

# Why Connected Services?

Visual Studio has long supported adding Services to a project through the **Add Service Reference** dialog. But app developers have to know where to find these services and had to learn how integrate them into their projects using procedures specific to the service. Providers had to write or wrap their service in Windows Communication Foundation (WCF) classes and were restricted to the programming language and platforms that they could target and the authentication implementation.

Connected Services address these limitations by letting you:

* Provide discovery of your service directly in Visual Studio.
* Configure the service using the **Add Connected Service** window that provides a common UI that is fully customizable.
* Collect and use the configuration data of your choice.
* Target any programming language and platform supported by Visual Studio.
* Use any type of communication protocol, such as JSon.
* Authenticate users in the way that best suits your service.
* Modify the app project by using Connected Services helper classes or by using the full power of the Visual Studio SDK extensibility apis.

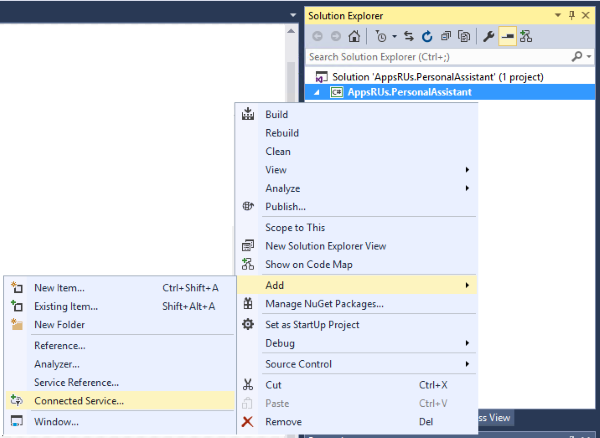
The main advantage of the Connected Services over service references might be what is missing. Connected Services extensions are not limited to any one set of protocols, implementations, or targets. Anything that can be implement by hand in Visual Studio can be automated in a Connected Service extension. You can host your service anywhere, use the authentication schemes that you like, and collect as much or as little configuration information as you need. You then have complete control over the modifications you make to the target app, from adding a single item to a config file to creating new projects in the app solution.

# How an app developer consumes a Connected Service

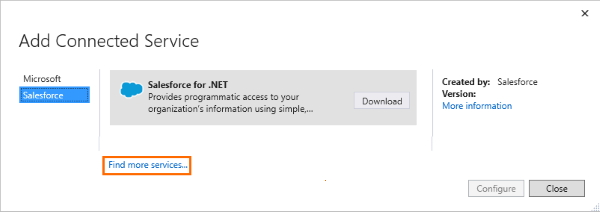
Here’s how an app developer experiences the discovery, configuration, and implementation of your connected service:

## Discovering the service

Mary, the developer of a personal assistant app hears that Contoso, Inc. has created a cloud-based service that provides a simple back-end for a to-do list. This functionality is exactly what she needs to compose one part of her app. She opens the **Add Connected Services** dialog through the context menu of their Visual Studio project.



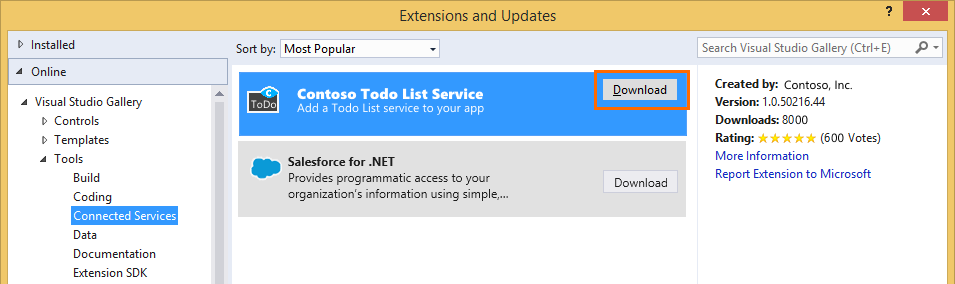
The Add Connected Service window shows her the connected service extensions that are already installed on the computer.



If the extension is not already installed, she can browse the Connected Services category of the Visual Studio Gallery to find the package.

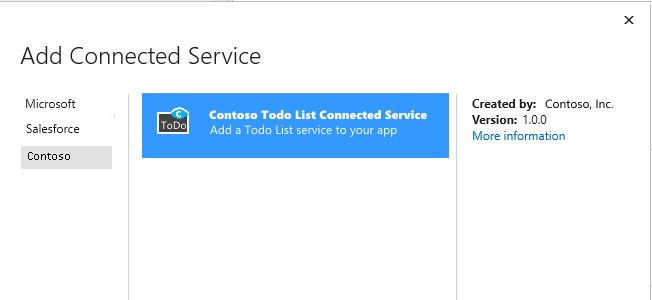
## Installing the Connected Services extension

She selects your service from the Connected Services category of the Extensions and Updates dialog, and then downloads and installs it.



## Adding the service to an app project

After the Todo List Service extension is downloaded to her computer, she can add it to her project from the Add Connected Service dialog.



She then configures the service by supplying the necessary data to the connected service. The kind and complexity of the configuration data collected by the extension is determined by the service provider, from a simple user name to possibly far more complex data that is required to modify her

## Implement the service in the app

After the extension runs, a Getting Started With Contoso Todo Lists page appears in the browser. The page describes the changes that are made to her project, such as adding configuration items and references, and modifying and creating files. The page then describes how Mary accesses and uses the service.

# Create a Connected Service provider

This section provides an overview of the processes you use to create a Visual Studio Connected Service extension that lets app developers add your service to their apps.

To create a Connected Service, you’ll need

* Visual Studio 2015 RC
* Visual Studio 2015 SDK RC
* Microsoft Connected Services SDK (installed through NuGet in your product)

# Connected Services Architecture

To create a Connected Service extension you use the [Visual Studio SDK](https://github.com/Microsoft/ConnectedServices-ProviderAuthorSamples/blob/master/Externals/Microsoft.VisualStudio.ConnectedServices.vsix) and the .NET [Managed Extensibility Framework (MEF)](https://msdn.microsoft.com/en-us/library/dd460648(v=vs.110).aspx) .to create a Visual Studio extension that an app developer can select from the Add Connected Service dialog and add your service to their project. The Connected Services SDK abstracts much of the complexity of the VS SDK and MEF into an API that quickly gets you up and running, lets you target multiple platforms and languages in a single project, and provides helper classes and methods to provide authentication and modify the app’s project files.

## The Connected Service Extension project

The Connected Services extension is created as a VSIX project in Visual Studio. The VSIX project template is available in the C#/Extensions folder of the New Project dialog.

In the .vsixmanifest file of the project, you’ll nedd to add properties, including a unique Provider ID, MEF support, and a dependency on the Connected Services SDK that you will install next.

You install the Connected Services SDK from the NuGet Package Manager for your project and installing the Microsoft.VisualStudio.ConnectedServices package. Installing the package downloads the Microsoft.VisualStudio.ConnectedServices.dll add a reference to it in the project. References are also added to the necessary .NET Framework assemblies.

## The Connected Services UI Model

The Connected Service UI presents three windows to the user.

The provider window specifies the information to display when an app developer selects your service in the Add Connected Services window. The UI for the provider shows the service name, the Add Connected Service category where the service is listed, and other information to help the developer understand the purposes of the service.

The configurator window collects information from the app developer. There are three types of configurator windows:

* The SinglePage configurator gathers the configuration data in a single page.
* The Wizard configurator displays a sequence of pages to collect the config data.
* The Grid configurator does not collect user configuration data, but lets the user chooser choose configurations from a list of options.

You can also add an authorization control to the SinglePage and Grid windows

The **handler** window displays progress information as your connected service is applied to the app.

The Connected Services windows that an app developer sees are created by created by the SDK using a variation of the WPF Model-ViewModel-View (MVVM) design pattern. MVVM separates the XAML UI (the View) from the logic and data it displays by using a class (the ViewModel) that is connected to the View by WPF data binding.

The ConnectedServices SDK infrastructure provides the complete XAML UI of the provider, Grid configurator, and the handler windows. You specify the information displayed in the window by deriving ViewModel classes from abstract classes in the Connected Services API.

For the SinglePage and Wizard configurators, you first create a WPF user control, then connect the control to the parent window by deriving the control’s ViewModel from a ConnectedServices abstract class. The base class creates the frame of the window, displaying the title, description, and controls to signal the completion of the configuration and navigation between the Wizard pages. The user control fills the remaining space of the window. The view of the control can contain any WPF controls that you need to present and collect the user choices. Your View specifies the title and description of the frame, and properties to contain that contain the configuration data. The properties are connected to your class through WPF data binding.

## Providers, Configurators, and Handlers

The provider class, derived from the ConnectedServiceProvider class, supplies data about your service to the Add Connected Service, and it also orchestrates the logic of the extension. When the user chooses to configure the service, the provider calls a configurator object that collects the data needed to connect the service to the app. When the configurator finishes, the provider retrieves the ConnectedServiceInstance that contains the configuration data. Methods in the base class then call a handler that performs the modifications to the app project that connects you service to the app.

You create a configurators by deriving a ViewModel class from the ConnectedServiceSinglePage, ConnectedServiceWizard, or the ConnectedServiceGrid. The configurators are WPF windows that enable you to collect as much information as you need.

* The Single Page configurator collects all the user data in one page. You implement a WPF user control for the central frame of the page whose model view is derived from the abstract Connected Services ConnectedServiceSinglePage abstract class.
* The Wizard configurator enables you to collect user data in a sequence of pages. You implement the wizard as a ConnectedServicesWizard that manages the pages by creating user controls for each page whose view models are derived from the ConnectedServicesWizardPage class.
* The Grid configurator is the simplest but most restrictive of the configurators. Derived from ConnectedServiceGrid class, it displays a non-configurable list of options to the developer.

You can also easily add authentication to the Single Page and Grid windows by including a ConnectedServiceAuthenticator user control in the window.

The user control of a single page or wizard configurator are entirely under your control. You specify the data required to configure your service in the properties of the ViewModel class, and connect the properties to the XAML UI of the control through XAML data binding.

Once the configuration process completes successfully, the provider is notified and the configuration information is transferred from the configurator to the provider in the metadata of a handler. The provider then finds, and initializes a handler that is derived from the ConnectedServicesHandler-class. It is the handler actually that adds the service to the app project.

In the handler you can add to the references of a project, modify its configuration and other files, and add new files, Connected Services SDK classes make it easy to add references, read and write to XML files, and create new files. You can also use the full functionality of the Visual Studio SDK to perform more complex tasks.