This how-to guide explores the core features that are available in Windows SDK, development environment and also details how to develop a sample SAP application using this development kit with step by step instructions that details every stage of the development process.
Abstract

Windows Software Development Kit (SDK) for Windows 8.1 is the latest version of Windows SDK that lets you develop Windows apps. The Windows Software Development Kit (SDK) for Windows 8.1 contains headers, libraries, and tools you can use when you create apps that run on Windows operating systems. You can use the Windows SDK, along with your chosen development environment, to write Windows Store apps and desktop apps for Windows 8.1 as well as Windows 8, Windows Server 2012, Windows Server 2008 R2, and Windows Server 2008.

SAP Mobile Platform 3.0 is the latest in the series of Mobile Application Development Platforms (MADP) from SAP that provides an open development environment, enabling users to develop mobile applications with familiar environments and languages, open source tools and third party toolkits, libraries and frameworks. Using SMP 3.0 it is possible to swiftly design cost-effective and intuitive apps with the most open and standards-based mobile application development platform.

SAP recently announced the release of SAP Mobile Platform SDK 3.0 SP07 which includes many new Windows features like providing offline support for Windows applications, Windows push notifications and certificate authentication directly with SAP Gateway. This service pack release furthers strengthens the capabilities of SAP Mobile Platform SDK 3.0 with the ability to build truly world-class Windows 8.1 applications for SAP with support for all the native features and functionality offered by the best Windows devices in the market.

This how-to guide explores the new features that are available with SAP Mobile Platform SDK 3.0 SP07 and explains how to develop a sample application that integrates with an SAP backend using this development kit with step by step instructions captured at every stage of the development process. Also included is an overview of the Windows SDK including its core features, the development environment, pre-requisites, various data operations and the procedures for app development using the SDK. We at Innovapptive have made every attempt to make this how-to guide as simple as possible. Hence, wherever possible, code snippets and screen shots have been provided to render the explanation more meaningful.
Overview of Windows SDK for 8.1

The SDK kit for Windows 8.1 consists of headers, libraries and tools you can use when you create apps that run on windows operating systems. Using Windows SDK along with your chosen development environment, you can develop windows store apps and desktop apps for a variety of windows operating systems such as Windows 8.1, Windows 8, Windows 7, Windows Vista, Windows Server 2012, Windows Server 2008 R2 and Windows Server 2008.

As a mobile application development platform, it solves mobility challenges and supports mobile apps that syncs with your business to enterprise (B2E) or business-to-consumer (B2C) use cases. Apart from that, it allows flexibility in balancing device user requirements with enterprise requirements.

Key features

- Seamlessly connect to SAP mobile platform endpoints to access your enterprise data.
- Create, update, and delete SAP objects.
- Query SAP objects using OData.
- Leverages your own strongly-typed C# objects.
- Streamline the authentication and onboarding process.
- Use the same C# API for iOS and Android.
- Leverage async/await to ensure that your UI is responsive and you code simple
- Provides a simple, yet compact API.

The windows SDK also includes the Windows App Certification Kit 3.3 (Windows ACK) to test your app for a variety of operating systems such as Windows 8, Windows 8.1, and Windows 8.1 phone certification program. However, one thing to be noted here is windows SDK no longer consists of a complete-line build environment; you can proceed installing Visual Studio 2013 that includes the appropriate components of the Windows SDK.
Enterprise App Development using SAP Mobile Platform

As a developer, your main objective is to write applications that solves business paradigms and meets the business goals and requirements. Apart from taking care of the business requirements, there are a variety of enterprise requirements that are common to all the applications including authentication, data security, administration, offline access, push notifications and synchronization. This is where SAP Mobile Platform (SMP) SDK helps you to write applications to meet your business requirements in a much simpler manner and simultaneously helps you to delegate the responsibilities of meeting enterprise requirements to a higher authority.

SAP Mobile Platform SDK 3.0 SP07 - Support for Windows

SAP Mobile Platform SDK 3.0 SP07 now comes with the capability to build truly world-class Windows 8.1 applications for SAP with support for all the native features and functionality. There are several key features that have been added in SMP SDK to support Windows. One important aspect of SMP SDK is that it not only supports the Windows Store applications but it also supports any Windows machine running .NET 4.5 or higher. So in essence, you could build an application using the SDK for both the new Windows 8.1 operating system and also for older legacy Windows machines running .NET 4.5 or higher. Also, using the Universal Windows app project templates, it is now possible to build applications using a single code base that runs on Windows Phone and Windows. These are exciting times in the mobility arena and I am quite optimistic about the future of SMP Windows SDK.

New features in SMP SDK 3.0 SP07 for Windows support -

- **Offline Support for Windows** - An offline OData store is initialized and populated with a list of OData collection end-points to be stored offline on the device. Device users can fully manipulate this data (query, update, create, delete, and so on), then merge the updates to the back-end with offline changes. An Offline application is well suited when connectivity may become unreliable, or requires infrequent updates of back-end data.

- **Technical Cache** - A new feature called Technical cache for Windows (8.1 and Phone 8.1) has been added, that provides enhanced support for online applications that are occasionally offline. The technical cache extends the OData API to enhance online applications in cases where the application is occasionally offline by optimizing application-to-server communication for online stores, performing conditional HTTP or OData requests to optimize network bandwidth utilization for cached responses from previous OData requests, and using cached content to improve application performance.

- **Certificate authentication directly with SAP Gateway in the MAF Logon Core** - certificate based authentication for SAP Gateway allows for certificate based authentication when connecting to a Gateway server. The certificate is provided by Afaria or a plugin via the certificate provider interface. The functionality matches that of SAP Mobile Platform Server, and applies to conversation manager as well. Scenarios supported include:
  - Standard HTTPS communication, certificate requested by the TLS layer.
  - HTTPS communication, the server responds with a certificate challenge.
  - HTTP communication, the server responds with a certificate challenge.

- **SAML support** - The Hybrid SDK (Kapsel) plugins now support Security Assertion Markup Language (SAML) for authentication in Windows applications.
Windows Push Notifications.

An important point to be noted is the APIs that are provided on the 3 platforms such as Windows desktop, Windows tablets and Windows Phone 8.1 are identical. The platform variances are handled inside the libraries and hence you can call LogonCore or ODataOnline (or any other library) APIs in the same fashion, irrespective of the targeted Windows platform. Apart from that, you can use the universal windows app project templates to build applications with the help of a single code base that runs on Windows phone and Windows.

Other important value added feature in SDK is the dynamic nature of consuming the OData services. Instead of having to add a service reference to the OData Service at the time of design to build the proxy classes with respect to WCF applications, you can build applications, which can dynamically consume any OData service at runtime using the SMP SDK. Apart from that, even complex properties can be handled in the sample application during runtime.

**Supported environments and devices**

**Windows Store Applications**

- Windows 8.1 (WinRT APIs) (Ex: Surface Pro 3, any PC with Windows 8.1 etc.)
- Windows Phone 8.1 (WinRT APIs) (Ex: Lumia 1520, HTC One M8 etc.)

**Unsupported environments and devices**

- Silverlight Runtime on Windows Phone
- Windows Phone (from version 7.0 to 8.0)

**Windows Desktop Applications**

- Any system running .NET 4.5 or higher – 4.5.1 and 4.5.2
- Windows 8.1

The API set that is available for Windows store applications was always windows runtime. The API set available for Windows Phone from version 7.0 to 8.0 was a modified Silverlight Runtime. With the unification of tablet and phone platforms by Microsoft, the Windows Phone 8.1 now supports the Windows runtime and the modified Silverlight Runtime's (to ensure backward compatibility) API set.

For building universal applications that can target both the Windows and Windows phone devices with a single code base (business logic that can be shared between the devices), Microsoft Visual Studio 2013 + Update 2 is required.

Note: You can also try Microsoft Visual Studio + Update 3 to build applications.

**Development Environment**

**Windows Store Applications**

- Windows 8.1 x 64 – Pro or above version, since Hyper-V is required
- Microsoft Visual Studio 2013 + Update 2 and above

**Windows Desktop Applications (any system running .NET 4.5 or higher)**

- Microsoft Visual Studio 2013 + Update 2 and above
Windows application development using the OData API

We will use the Windows mobile application - mCustomer to demonstrate how SMP SDK 3.0 SP07 can be leveraged to integrate an SAP backend to a Windows mobile application. The mCustomer app is a customer contact information lookup and management application, which can be used to search the list of customer accounts under an organization, based on parameters like the sales organization, distribution channel and division. The app also allows users to further drill down a specific customer account to get detailed information of the account under various heads such as the Address of the Account, Communication Details (Telephone, Fax, etc), Contact Persons and also the Quick Sales information. The application syncs with SAP to get customer contacts, letting the user to edit contacts.

**Prerequisite - Windows mobile app UI development using Windows Phone SDK has been completed.**

The subsequent sections of this document explores the procedures and the logic that we have incorporated to integrate the Windows mobile application to an SAP backend using SMP SDK 3.0. We earnestly believe this will help you get started to build your own application and integrate to an SAP backend using the steps below -

1. Register devices and connect users
2. Initialize an online store
3. Read operations
4. Bind responses to UI controls
5. Perform CUD (create, update and delete) operations
6. Perform logging operations.
Step 1 - Register devices and connect users

Overview

Registration is also referred to as onboarding a device, which is the first stage in windows application development. Register a device and login as a user using LogonCore and SecureStore library calls.

A new device needs to be registered with the SMP, before it can accomplish any of the CRUD operations against the backend, as part of the administration capabilities. This process is typically referred to as onboarding and is quite helpful for administration purposes. You now have a device history log that tends to be quite helpful for troubleshooting purposes. Apart from that, you can target the specific device for any notification, when it comes to server initiated push notifications.

With the new SMP SDK, the process of onboarding a device has become pretty simple by just making asynchronous calls using the await/async keywords. The compiler on its part takes care of all the heavy stuff and the end result is code that closely resembles synchronous code. Once you successfully onboard, an application connection ID is created and sent to the device along with the server response. This application ID uniquely identifies the device and this application ID is passed to the header during further communication with the SMP server.

Prerequisites

1. Initially create project in Visual Studio
2. Add certain libraries to the project
3. Define the application and client password policy using the cockpit in the SMP server.

![Figure 1: Create New project](image)
To create a project in Visual Studio

1. In the FILE menu, click New and then click Project.

2. In the New Project screen:
   a. Click Blank App (Windows Phone)
   b. Type the name of the project and browse to define the location where you want to save the project files.
   c. Click OK to create the new project.

Creating libraries

To get the app running, you need to add certain libraries to the project. These libraries allow you to use the OData APIs as well as the SMP APIs. ODATA libraries are available in the NuGet store, while SMP libraries come with the SMP 3.0 mobile SDK install.

To add libraries -

1. From the PROJECT menu of the mCustomer - Microsoft Visual Studio screen, click Manage NuGet Packages.

2. From the Solution Explorer of the mCustomer - Microsoft Visual Studio screen, click Manage NuGet Packages.
3. In the mCustomer – Manage NuGet Packages screen:

a. Search for the libraries that you want to install in the Search box to display a list of packages.

b. Click Install corresponding to the package that you want to install. A confirmation sign is displayed (in green) notifying that the package has been successfully installed.

c. Click Settings to proceed adding NuGet packages.

*Note: All pre-installed packages are also displayed with a confirmation sign.*
Figure 5: mCustomer - Manage NuGet Packages

Figure 6: mCustomer - Manage NuGet Packages
4. In the Options screen:
   
   a. Under NuGet Package Manager, click Package Sources to display the list of package sources that are available.
   
   b. In the Available Packages section, click the package source (SMP) to add the path to SMP NuGet packages as one of the package sources (on your local system).
   
   c. Type the name of the package sources and define the folder (source), where you want to install these packages.
   
   d. Click OK to save the changes.

![Options](image)

*Figure 7: mCustomer – Options (Add Package Sources)*

5. In the mCustomer - Manage NuGet Packages screen:
   
   a. Click the + button to add a new NuGet Source. At the bottom of this window, Fill in the name, in our case SMP, and the point the Source to the location of SMP OData Framework (Normally `<SMP SDK install path>\NativeSDK\OData Framework\Windows`).
   
   b. Click OK to close & Save the changes.
   
   c. Repeat Step 1, but change the NuGet Source as shown in below screenshot. And follow the procedure to add libraries to the project.
Procedure

1. Initialize the `logonCore` variable using the following code snippet.

   ```csharp
   var logonCore = await SAP.Logon.Core.LogonCore.InitWithApplicationIdAsync("<Application id>反倒);
   ```

2. Call `RegisterWithContextAsync` to register the device that sends an HTTP POST request to the SMP server with a payload to identify the device with the SMP server. Subsequently, the SMP registers the device and reverts the registration information as part of the response.

   ```csharp
   var logonContext = new LogonContext
   {
       RegistrationContext = new RegistrationContext
       {
           ApplicationId = "<Application id反倒>",
           ServerHost = "<Server HostAddress反倒>",
           IsHttps = <True/False>
           ServerPort = <SMP Server Port>
           BackendUserName = "<Backend User name反倒>",
           BackendPassword = "<Backend Password反倒>",
           SecurityConfig = "<Security Config反倒>",
       },
   };

   Once the LogonContext is available, we can now perform Registration
   // registers the device
   await logonCore.RegisterWithContextAsync(logonContext);

3. Persist locally all registration information received from SMP server using the following code snippet.

   ```csharp
   await logonCore.PersistRegistrationAsync(passcode, logonContext);
   ```

   The passcode is a randomly chosen password that is eligible for the passcode policy reverted by the server.
4. Store the application connection ID after logon:

```csharp
string connectionId = (string)logonContext.RegistrationContext.ConnectionData["ApplicationConnectionId"];```

**Registration Process Flow**

Depending on device status and the password policy, the registration and log on process varies, which is outlined as follows:

**Logon Process Flow**

![Diagram of Logon Process Flow]

If the device is registered:

- **OnNavigatedTo** - Windows Store applications
- **Event handler for the Windows Loaded event** - Windows desktop applications

**OnNavigatedTo** is called when a page is about to appear in a frame, which tends be an ideal time to check whether a device is registered or not. This is executed by calling `InitWithApplicationIdAsync`.

When a fully initialized logonCore is returned, the application tries to create a DataVault on the registered device. If a DataVault already exists, the application attempts to unlock the DataVault without any password, if successful, `RegistrationContext` is loaded. If the DataVault does not exist, a new LogonContext is generated with default values -

- If the device is already registered, the application unlocks the DataVault by sending in NULL as the password. If this process fails, the user is prompted to type a password for the DataVault.
- If the device is not registered, call `RegisterWithContextAsync` to register the device with SMP server.

To check whether the device is registered or not:
var logonCore = await SAP.Logon.Core.LogonCore.InitWithApplicationIdAsync("application id");
if (logonCore.State.IsRegistered)
{
    await logonCore.UnlockSecureStoreAsync(null or passcode);
}
else
{
    // device is not registered
}

• Call RegisterWithContextAsync only if the device is not registered. This method ensures that your device gets registered on SMP server.

await logonCore.RegisterWithContextAsync(logonContext);
var logonContext = new SAP.Logon.Core.LogonContext
{
    RegistrationContext = new SAP.Logon.Core.RegistrationContext
    {
        ApplicationId = "App Id", ServerHost = "IP Address Or Hostname", IsHttps =
false, ServerPort = 8080
    }
};

• The PasswordPolicy.IsEnabled property of the logonContext object is queried to check whether a password is enabled or not:

if (logonContext.PasswordPolicy.IsEnabled)
{
    // Prompt the user for passcode. Call the PersistRegistrationAsync method with the passcode that the user entered
    await logonCore.PersistRegistrationAsync(passcode, logonContext);
}
else
{
    // No need to prompt the user for passcode. Simply call PersistRegistrationAsync method by passing null for passcode
    await logonCore.PersistRegistrationAsync(null, logonContext);
}

Step 2 - Initialize an online store

Overview
The ODataStore library interacts extensively with the OData service. Note that one ODataStore instance interacts with one OData source only. If you define a SMP server-hosted application with two or more back-end URLs (also called as whitelisted connections), a separate ODataStore instance is required for each connection. The ODataStore hides several of the complexities of interacting with the OData source and internally makes use of the Microsoft OData parser.

You need to initialize an online OData store for online OData access.

Prerequisites
In SMP server, define the back-end OData connections required by the application using the cockpit.
1. Configure the HTTPClient to open the online store. The below listed configuration can handle basic and SAML authentications. Apart from that, X-CSRF-Token handling is enabled and the connection id is sent automatically in each request -

```csharp
var client = new SAP.Net.Http.HttpClient(
        Credentials = new System.Net.NetworkCredential(  
            (string)registrationContext.BackendUserName,
            (string)registrationContext.BackendPassword)
    }, true;

client.DefaultRequestHeaders.TryAddWithoutValidation("X-SMP-APPCID", connectionId);
client.DefaultRequestHeaders.TryAddWithoutValidation("X-SUP-APPCID", connectionId);
client.ShouldHandleXcsrfToken = true;
client.ShouldHandleSamlRequests = true;
    registrationContext.ServerHost,    
    registrationContext.ServerPort, "/SAMLAuthLauncher").Uri;
client.SamlFinishEndpointParameter = "finishEndpointParam";
await SharedContext.Context.Store.OpenAsync(client);
```

2. Create an online ODataStore using one of the two overloaded constructors as outlined below:

**Constructor 1**

```csharp
public ODataStore(string serviceUri, ODataStore.EntityFormat entityFormat = ODataStore.EntityFormat.XML);
```

//The first parameter is the URI for the OData Service passed as a string  
//The second parameter is the entity format. The default option is XML.  
//Using JSON greatly reduces network traffic and is recommended.

```csharp
public enum EntityFormat
{
    JSON = 0,
    XML = 1,
}
```

//Example:
var store = new ODataStore("uri");

**Constructor 2:**

```csharp
public ODataStore(Uri serviceUri, ODataStore.EntityFormat entityFormat = ODataStore.EntityFormat.XML);
```

//The first parameter is the URI for the OData Service passed as a System.Uri  
//The second parameter is the entity format. The default option is XML.  
//Using JSON greatly reduces network traffic and is recommended.

```csharp
public enum EntityFormat
{
    JSON = 0,
    XML = 1,
}
```

//Example:
var store = new ODataStore(new Uri("uri"), EntityFormat.JSON);
Step 3 - Perform Read Operations

Overview

The following flowchart pictorially depicts the sequential flow of Read operations.

Procedure

1. Create an instance of Online ODataStore (Refer earlier procedure on initiating an Online Store).

2. Pass the instance of SAP.Net.Http.HttpClient to call OpenAsync by executing the below sub-steps:
   c. Use SAP.Net.Http.HttpClient to pass in the headers (for example, X-SUP-APPCID).

   ```csharp
   var client = new SAP.Net.Http.HttpClient(
       NetworkCredential("user", "password") }, true);
   client.DefaultRequestHeaders.TryAddWithoutValidation("X-SMP-APPCID", appconf);
   await store.OpenAsync(client);
   ```

3. Call ScheduleReadEntitySet passing in the collection name, which is outlined in the following code snippet:

   ```csharp
   // Read the collection and wait for the response
   // Note: The collection name can include filters. For example,
   // Suppliers?$top=5
   ```
var execution = store.ScheduleReadEntitySet(collectionName);
IODataResponse response = await execution.Response;
// Subsequent calls to read collections
var execution = store.ScheduleReadEntitySet(anotherCollectionName);
IODataResponse response = await execution.Response;

The Response property is called asynchronously.

Supplementary section: How to make HTTP requests from Whitelisted connections
You can make HTTP Get requests from either whitelisted OData or JSON service connections.

Prerequisites
Define a whitelisted back-end connection used by the application in SMP Server using Management Cockpit.

Procedure

1. If the whitelisted connection is an OData service:
   a. Create a new instance of ODataStore: var store = new ODataStore("Uri");
   b. Follow the steps Making HTTP Get Requests.

2. If the whitelisted connection is a JSON service:
   a. Create a new instance of SAP.Net.Http.HttpClient that is initialized using the credentials from the DataVault, add the application connection ID as a header, and submit the request asynchronously.
   b. In order to parse response, use native JSON libraries to parse response. Follow the below code snippet.

```
SAP.Net.Http.HttpClient()
//to pass in the credentials
var client = new SAP.Net.Http.HttpClient(
    {
      Credentials =
        new System.Net.NetworkCredential(
         Globals.LogonCore.LogonContext.RegistrationContext.BackendUserName,
    },
    true); // will be disposed by the store!
client.DefaultRequestHeaders.TryAddWithoutValidation("X-SMP-APPCID",
connectionId);
// Send a request asynchronously continue when complete
HttpResponseMessage response = await client.GetAsync(url);
// Check that response was successful or throw exception
response.EnsureSuccessStatusCode();
// Read response asynchronously as JsonValue and write out top facts for each
country
var content = await response.Content.ReadAsStringAsync();
// Parse it into an JObject
var jsonContent = JObject.Parse(content);
var airport = jsonContent.GetNamedValue("name").GetString();
```
Step 4 - Bind responses to UI controls

Overview

This procedure implies binding the OData response to a UI control. The ODataEntitySet is a major component of the response from SMP server. You need to take care of the following points when you bind a response to a UI control:

- ODataEntitySet represents an OData collection. Ex: All BusinessPartners that match a specified criteria.
- ODataEntitySet is also an IObservableCollection, which implies UI controls are bound to the ODataEntitySet that is automatically updated when you add or remove entities from the set.
- ODataEntity denotes a single row of an OData collection. Ex: A single BusinessPartner.
- ODataEntity implements the INotifyPropertyChanged interface.
- Some helpful methods that operate on ODataEntity include DeepCopy, GetNavigationProperty, and so on.

Procedure

1. Pass CollectionName as parameter and use Await on the Response property to call ScheduleReadEntitySet method. If an exception is returned, display an error message; else, cast response.Payload to ODataEntitySet and store in variable <X>.

```csharp
// Cast response.Payload to IODataEntitySet and store in variable X
var execution = Store.ScheduleReadEntitySet(collectionName);
try
{
    IODataResponse response = await execution.Response;
    this.EntitySet = (IODataEntitySet)((IODataResponseSingle)response).Payload;
}
catch (Exception)
{
    throw;
}
```

2. Set the DataContext of the XAML page to the variable <X> at design time or runtime. Using the Properties member field (also referred as an indexer), bind the properties of a single entity in the set.

```csharp
this.DataContext = SharedContext.Context = new DataContext();
<GridView x:Name="ItemGridView"
    ItemsSource="{Binding EntitySet}"
    <DataTemplate x:Key="Standard250x250ItemTemplate">
        <TextBlock Text="{Binding Properties[Name].Value}"
```

Supplementary section: Bind responses to UI controls using LINQ

You can use Language-Integrated Query (LINQ) to bind a response to a UI control. LINQ enables you to manipulate the ODataEntitySet, and write complex filters.
Procedure

1. Use LINQ to filter the ODataEntitySet (use either lambda expression or SQL) and then display the results:

   // LINQ extension method using the lambda style
   var filteredEntitySet = entitySet.Where(e =>
      ((string)e.Properties["Address/State"].Value) == "WA");

   // LINQ extension method using the SQL style
   var filteredEntitySet = from e in entitySet
      where ((string)e.Properties["Address/State"].Value) == "WA"
      select e;

   foreach (var e in filteredEntitySet)
   {
      Console.WriteLine(e.Properties["Name"].Value);
   }

Step 5 - Perform CUD (Create, Update, Delete) Operations

You can perform the CUD and other operations on the OData source.

Create Operations

You can use the ScheduleCreateEntity method to create an entity in the backend. This method uses ODataEntity and the CollectionName as parameters. An entity is created locally and passed in as a parameter to the ScheduleCreateEntity method.

The following steps illustrates how you can execute the Create operations -

1. Create a local ODataEntity and set values for the properties.
2. Set the PropertyCreationMode with any of the available options like All, Keys, Mandatory and Optional. You can combine the PropertyCreationMode using the binary | operator

   var entity = new SAP.Data.OData.Online.ODataEntity("TypeName");
   entity.Properties["ID"].Value = XYZ;
   entity.Properties["Name"].Value = "XYZ";
   Store.AllocateProperties(entity,
      SAP.Data.OData.Store.PropertyCreationMode.All);
   var execution = Store.ScheduleCreateEntity(entity, collectionName);
   await execution.Response;
   //You can also store the Response object in a variable and perform
   //operations against it,
   //if you know that the server returns something when an entity is created
   var response = await execution.Response;

Update Operations

You can update an ODataEntity using the ScheduleUpdateEntity method on the ODataStore. Apply the DeepCopy() method to make a real copy (in contrast to a reference copy) of an entity. Make the necessary updates on the real copy of the ODataEntity. This ensures that the original entity remains intact and the app remains in a consistent state, even if the update operation fails in any of the situations including
network disconnection and server error. The following are the steps:

1. Create a deep copy of an existing entity for a temporary ODataEntity variable.
2. Update the properties of the temporary ODataEntity variable (copiedEntity).

Note that DeepCopy() only makes a shallow copy of any child objects, in case they are available. For instance, if your ODataEntitySet is created using the $expand option, then you have parent and child entities such as SalesOrder and SalesOrderItems respectively. When you execute a DeepCopy(), it makes a deep copy of the parent object and a shallow copy of the child object.

```csharp
var copiedEntity = entity.DeepCopy();
copiedEntity.Properties["Name"].Value = newName;
var execution = Store.ScheduleUpdateEntity(copiedEntity);
await execution.Response;
```

### Delete Operations

You can delete an ODataEntity using the ScheduleDeleteEntity method on the ODataStore. The following is the code snippet for performing this operation.

```csharp
var execution = Store.ScheduleDeleteEntity(entity);
await execution.Response;
```

### Miscellaneous Operations

There are a number of miscellaneous operations that you can call on an entity. The following is the list of operations (methods):

```csharp
public IODataRequestExecution SchedulePatchEntity(IODataEntity entity,
Dictionary<string, string> options = null);

public IODataRequestExecution ScheduleReadEntity(IODataEntity entity,
Dictionary<string, string> options = null);

public IODataRequestExecution ScheduleReadEntity(string resourcePath,
Dictionary<string, string> options = null);

public IODataRequestExecution ScheduleReadPropertyComplex(string resourcePath,
Dictionary<string, string> options = null);

public IODataRequestExecution ScheduleReadPropertyLink(string resourcePath,
Dictionary<string, string> options = null);

public IODataRequestExecution ScheduleReadPropertyLinkSet(string resourcePath,
Dictionary<string, string> options = null);

public IODataRequestExecution ScheduleReadPropertyPrimitive(string resourcePath,
Dictionary<string, string> options = null);

public IODataRequestExecution ScheduleReadPropertyRaw(string resourcePath,
Dictionary<string, string> options = null);
```
Function Imports

Function imports are helpful in providing custom functionality and are not bounded to any collection. However, having prior knowledge of return type of the function import would be useful.

```csharp
var resourcePath = “GetProductsByRating?rating=5”;
var execution = Store.ScheduleFunction(resourcePath);
var response = (await execution.Response) as IODataResponseSingle;
if (response != null)
{
    if (response.PayloadType == ODataType.EntitySet)
    {
        this.EntitySet =
            (IODataEntitySet)((IODataResponseSingle)response).Payload;
    }
}
```

Step 6 - Logging operations

Overview

SMP provides support through logs and traces that lets administrators, developers and support professionals to troubleshoot application issues. All logs have a common format and are stored in the SMP server database. All log entries for a specific business or application flow (for instance, OData request or a registration) are associated across the client and server stack. This lets you visualize and comprehend the end-to-end flow, helping you in identifying the source of an application problem. Apart from logs and traces, you can even activate end to end tracing on the basis of a single user or device to offer support to execute end-to-end diagnostics to identify performance or functional application issues.

Administrators and support professionals can identify problem areas with the help of system logs that collect log messages. Similarly, developers can identify code problems by tracking debug level log messages. You can set the log level for individual logging components to specify the amount of captured information.

Application tracing captures additional business data for a request. Typical example include message data, payloads, HTTP headers and URIs, which can be used for troubleshooting application problems. How much business data is captured in application traces is assessed by the application developer. You can enable tracing for individual logging components, as per your specific business requirements.

This is the last stage of the windows based application development, wherein you can log various application events by specifying a log level and then subsequently upload the client log to SMP server for analysis.

Prerequisites

Define the client log and trace policies in SMP server using the cockpit.

Conclusion

In the light of SMP becoming the standard for developing applications that can integrate to any backend including SAP, the aim of this guide is to provide you a conceptual and a procedural framework of how you can get started to create a simple windows based application integrated to SAP.
For more information

To learn more about Innovapptive's mPower™ App Suite, Enterprise Mobility Strategy and Consulting, Rapid Deployment Solutions and Custom Development Services, contact your Innovapptive sales representative, visit us at www.innovapptive.com or you can email Innovapptive directly at sales@innovapptive.com

About Innovapptive

Innovapptive is among the fastest-growing enterprise mobile and user experience (UX) solution companies and is an industry leader among rapid mobile application development (RMAD) solution providers. Innovapptive’s Rapid App Configurator Engine (RACE) technology redefines traditional approaches to digitize the enterprise and is empowering organizations to rapidly deliver ready-to-run mobile apps across the enterprise. Innovapptive’s portfolio of 100+ mobile solutions that run across a wide array of devices and systems is creating a connected enterprise across employees, customers, and suppliers. Innovapptive was named a 2016 SAP Pinnacle Award winner and has won these recognition three years in a row.