Hands-on lab: Creating a Sample resource provider

# Goal

The goal of this hands-on lab is to have the reader create a sample Resource Provider (RP) which can then be registered with Windows Azure and be made available in the Store. The sample RP will enable the reader to understand the interaction between Windows Azure and the service providing the resource.

# Overview of the Sample

The sample contains the following major components:

1. **Routes**: These are defined in <file name> and describe the endpoints that Windows Azure will be calling. The service will respond to these requests appropriately.
2. **DataModel**: this implements a sample functionality which demonstrates how the data layer could handle requests coming from Windows Azure. The actual implementation here would be dependent on the provider. For example, for a database service, this layer would be responsible for creating, deleting, databases etc.
3. **Controllers**: There are three controllers which handle incoming events:
   1. CloudServices: This handles interactions related to Cloud Services and allows the user to
      1. Get all the Cloud Services
      2. Get resources within a Cloud Service
      3. Delete a Cloud Service.
   2. Resources: This handles interactions at a resource level and allows the user to :
      1. Get details of a resource that can then be displayed in the Azure Developer Portal
      2. Provision a resource
      3. Update a resource
      4. Delete a resource
      5. Generate a SSO token so that the user be single signed-on to the partner’s own management portal

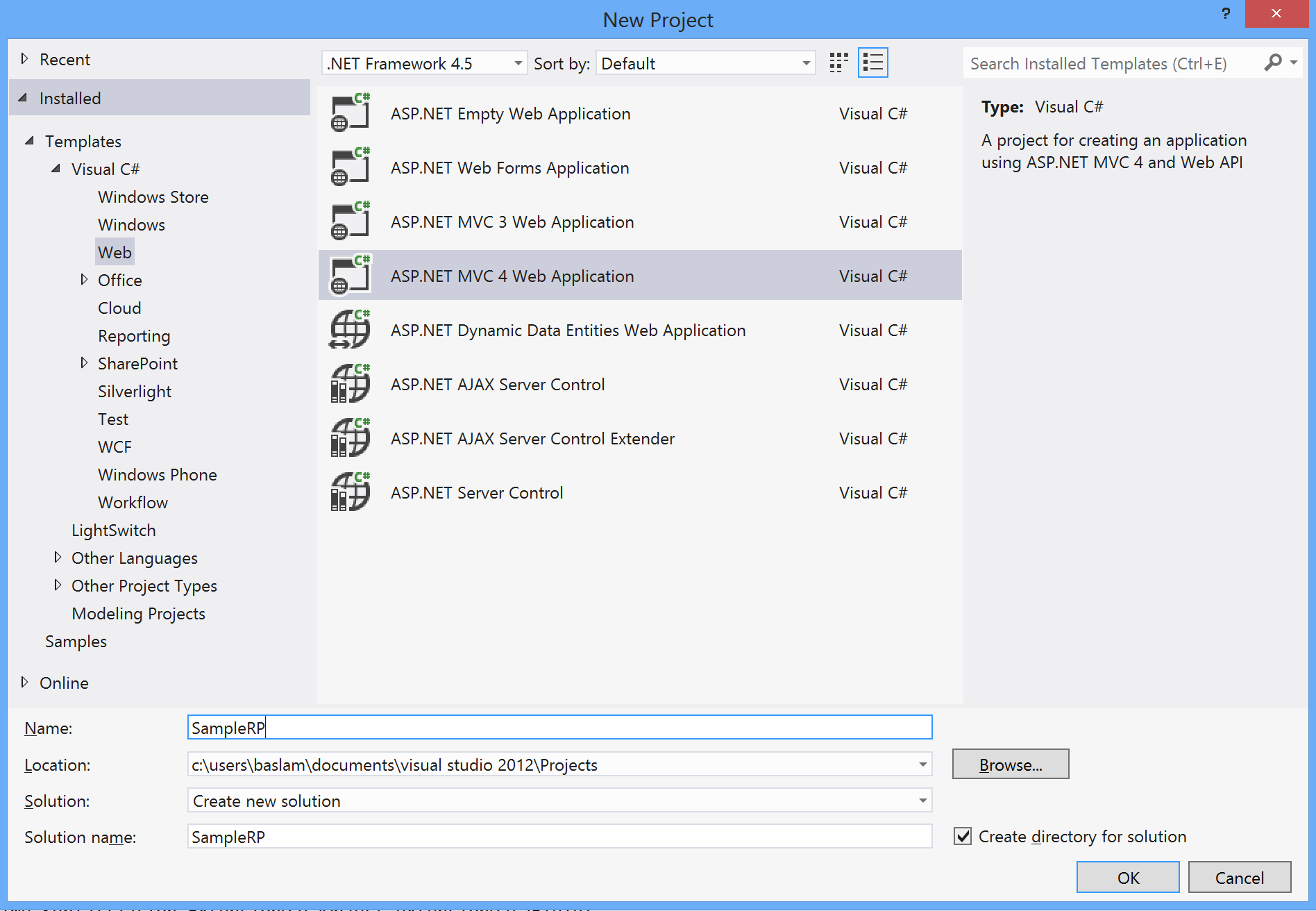
The sample is implemented as an [ASP.NET WebAPI](http://www.asp.net/web-api) project and requires Visual Studio 2012.

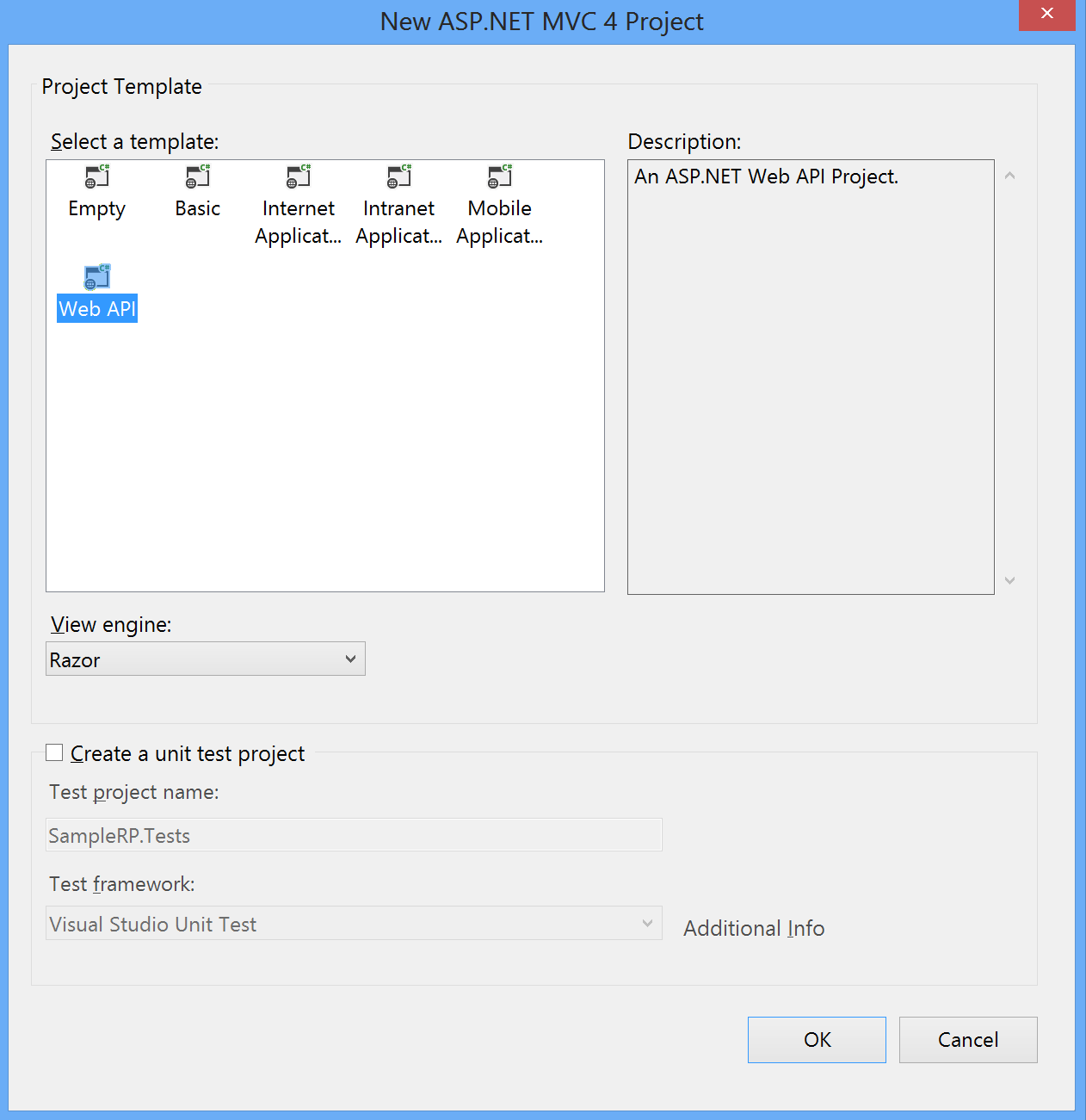
This document does not cover the following resource provider functionalities.

1. Handling subscription events
2. Securing the connection between Windows Azure and the resource provider service.
3. Integrating with Azure Portal using SSO.

# Detailed Steps

## Create a Web API project in Visual Studio 2012

1. Start Visual Studio 2012
2. Click File > New > Project…
3. Click on the Web template group and choose ASP.NET MVC 4 Web Application
4. Click OK
5. Choose “Web API” in the template dialog:



1. Click OK

## Routes

In the App\_Start folder, create the following routes in WebAPIConfig.cs

### Handle subscription events

In this section you register routes for two calls that are invoked on a subscription.

1. Handling subscription events – This will map to the Events Controller.

// Subscription notifications

config.Routes.MapHttpRoute(

name: "Events",

routeTemplate: "events",

defaults: new { controller = "Events", action = "HandleSubscriptionNotifications" }

);

1. Handling a Get request on a Cloud Service resource. This will map to the Cloud Services controller.

config.Routes.MapHttpRoute(

name: "CloudService-Management-Subscription",

routeTemplate: "subscriptions/{subscriptionId}/cloudservices",

defaults: new

{

controller = "CloudServices",

action = "GetAllCloudServices"

}

);

### Handle Cloud Service events

In this section you would register a route for responding to requests at a cloud service level. There is only one route that is registered here and it will handle requests on a cloud service.

config.Routes.MapHttpRoute(

name: "CloudService-Management",

routeTemplate: "subscriptions/{subscriptionId}/cloudservices/{cloudServiceName}",

defaults: new {

controller = "CloudServices"

}

);

### Handle resource events

In this section you would register a route for handling all resource level interactions. These will be mapped to the resources controller

// Resource level management

config.Routes.MapHttpRoute(

name: "Resource-Management",

//routeTemplate: "subscriptions/{subscriptionId}/cloudservices/{cloudServiceName}/resources/{resourceType}/{resourceName}",

routeTemplate: "subscriptions/{subscriptionId}/cloudservices/{cloudServiceName}/resources/{resourceName}",

defaults: new

{

controller = "Resources"

}

);

Once you have registered these routes, you will create controllers that will handle requests on these URLs and respond appropriately.

## Data Model

We will now implement a sample data model with trivial implementation of business logic.

Create a file called DataModel.cs under the Models folder in your project. Declare a namespace named <Your RP Name>.Models.<Model name> (In the sample this is ResourceProvidR.Models.RPDataModel)

The primary objects that you will be working with are:

### CloudService

This class will contain the data members associated with the Cloudservice. This includes the name, Incarnation ID and geo-location of the service. It will also contain the method using for listing resources in the Cloud Service.

internal class CloudService

{

public string Name { get; set; }

public int IncarnationId { get; set; }

public string GeoLocation { get; set; }

public List<ResourceOutput> Resources { get; private set; }

public CloudService()

{

IncarnationId = 0;

GeoLocation = "";

Resources = new List<ResourceOutput>();

}

}

### Subscription

This will contain data members associated with the Subscription, specifically the subscription ID and a method for listing Cloud Services within the subscription.

internal class Subscription

{

public string Id { get; set; }

public List<CloudService> CloudServices { get; private set; }

public Subscription()

{

CloudServices = new List<CloudService>();

}

}

### DataModel

You will also need the Datamodel class which will handle the actual interactions. As this service could be called multiple times, it is a good idea on using thread safe code. In the sample it is done using a lock object to ensure that only one thread can access the code block. In the declaration of the datamodel, you should declare a lock object. You will also need to use a dictionary object which keeps a list of all the subscriptions and their unique ids.

public static class DataModel

{

private static object theMassiveLock = new object();

private static Dictionary<string, Subscription> allSubscriptions = new Dictionary<Guid, Subscription>();

Next you would need to handle the actual interactions with Windows Azure.

1. Insert or update resource:

In this method, the resource provider should do the following high level tasks.

1. Check if the subscription had been passed earlier or not.
2. Check if the service has been passed earlier or not
3. Check if the resource being passed is a new resource or not. This would be typically done on the name being passed in.
4. If it is an existing resource then check if the resource Incarnation ID passed in is different from the one stored in your database. Incarnation ids are used to tracking if the resource has been updated. Windows Azure will call this invoke if the service containing the resource has been updated. In some cases your resource may not have been updated so using this incarnation id will ensure that you don’t do any unnecessary updates on your end.
5. If the incarnation id is different, update the resource
6. If the resource did not exist create a new one.
7. You can create new outputitems containing the output values that you want to display in the portal and add it to the resource output object.
8. Return the resource output object back in the response body.
9. Ensure that you set the right response status as well.

public static ResourceOutput InsertOrUpdateResource(string subscriptionId, string cloudServiceName, string resourceType, string resourceName, ResourceInput resource)

{

ResourceOutput output;

lock (theMassiveLock)

{

Subscription subscription;

if (!allSubscriptions.TryGetValue(subscriptionId, out subscription))

{

subscription = new Subscription() { Id = subscriptionId };

allSubscriptions[subscriptionId] = subscription;

}

CloudService theMatchingCloudService = subscription.CloudServices.SingleOrDefault<CloudService>(cs => String.CompareOrdinal(cs.Name, cloudServiceName) == 0);

if (theMatchingCloudService == null)

{

theMatchingCloudService = new CloudService() { Name = cloudServiceName };

subscription.CloudServices.Add(theMatchingCloudService);

}

ResourceOutput theMatchingResource = theMatchingCloudService.Resources.FirstOrDefault(r => String.Compare(r.Name, resourceName) == 0);

if (theMatchingResource != null)

{

if (theMatchingResource.IncarnationId != resource.IncarnationId)

{

theMatchingResource.CloudServiceSettings = resource.CloudServiceSettings;

theMatchingResource.IncarnationId = resource.IncarnationId;

theMatchingResource.IntrinsicSettings = resource.IntrinsicSettings;

theMatchingResource.Name = resourceName;

theMatchingResource.OperationStatus = new OperationStatus()

{

Error = new ErrorData() { HttpCode = 200, Message="OK" },

Result = OperationResult.Succeeded

};

theMatchingResource.OutputItems = new OutputItemList(new List<OutputItem>() {

new OutputItem() { Key="ProvidR-Output-1", Value = DateTime.Now.ToShortDateString() },

new OutputItem() { Key="ProvidR-Output-2", Value = Environment.MachineName }

});

theMatchingResource.Plan = resource.Plan;

theMatchingResource.SchemaVersion = resource.SchemaVersion;

theMatchingResource.State = ResourceState.Started;

theMatchingResource.SubState = "";

theMatchingResource.Type = resource.Type;

}

output = theMatchingResource;

}

else

{

output = new ResourceOutput()

{

CloudServiceSettings = resource.CloudServiceSettings,

IncarnationId = resource.IncarnationId,

IntrinsicSettings = resource.IntrinsicSettings,

Name = resourceName,

OperationStatus = new OperationStatus()

{

Error = new ErrorData() { HttpCode = 200, Message="OK" },

Result = OperationResult.Succeeded

},

OutputItems =GenerateOutputItems(),

Plan = resource.Plan,

SchemaVersion = resource.SchemaVersion,

State = ResourceState.Started,

SubState = "",

Type = resource.Type

};

theMatchingCloudService.Resources.Add(output);

}

}

return output;

}

You will also need to add a function to generate the output items that is called within the above method.

private static OutputItemList GenerateOutputItems()

{

List<OutputItem> outputItems = new List<OutputItem>();

// Add Well Known output values

outputItems.Add(new OutputItem() { Key = WellKnownOutputValues.ConnectionServerName, Value = "db1234567.contoso.com" });

outputItems.Add(new OutputItem() { Key = WellKnownOutputValues.ConnectionDatabaseName, Value = "asxr210i" });

outputItems.Add(new OutputItem() { Key = WellKnownOutputValues.ConnectionUserName, Value = "user-1" });

outputItems.Add(new OutputItem() { Key = WellKnownOutputValues.ConnectionPassword, Value = "a\_pa$s\_word" });

// Add some additional output values

outputItems.Add(new OutputItem() { Key="ProvidR-Output-1", Value = DateTime.Now.ToShortDateString() });

outputItems.Add(new OutputItem() { Key="ProvidR-Output-2", Value = Environment.MachineName });

return new OutputItemList(outputItems);

}

1. Get resource: In this call the resource provider should return details about the resource identified by the combination of subscription, service and resource name. If any of these are not found, the resource provider should return a not found (404)HTTP status code

public static ResourceOutput GetResource(string subscriptionId, string cloudServiceName, string resourceName)

{

lock (theMassiveLock)

{

Subscription subscription;

if (!allSubscriptions.TryGetValue(subscriptionId, out subscription))

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

CloudService theMatchingCloudService = subscription.CloudServices.SingleOrDefault<CloudService>(cs => String.CompareOrdinal(cs.Name, cloudServiceName) == 0);

if (theMatchingCloudService == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

ResourceOutput theMatchingResource = theMatchingCloudService.Resources.FirstOrDefault(r => String.Compare(r.Name, resourceName) == 0);

if (theMatchingResource != null)

{

return theMatchingResource;

}

else

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

}

}

1. Delete resource: the resource provider should delete the resource identified by the service name,subscription id and resource name.

public static void DeleteResource(string subscriptionId, string cloudServiceName, string resourceName)

{

lock (theMassiveLock)

{

Subscription subscription;

if (!allSubscriptions.TryGetValue(subscriptionId, out subscription))

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

CloudService theMatchingCloudService = subscription.CloudServices.SingleOrDefault<CloudService>(cs => String.CompareOrdinal(cs.Name, cloudServiceName) == 0);

if (theMatchingCloudService == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

ResourceOutput theMatchingResource = theMatchingCloudService.Resources.FirstOrDefault(r => String.Compare(r.Name, resourceName) == 0);

if (theMatchingResource != null)

{

// Do not increment incarnation ids

//theMatchingCloudService.IncarnationId++;

theMatchingCloudService.Resources.Remove(theMatchingResource);

}

else

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

}

}

1. Delete cloud service: The resource provider should delete the cloud service and all the resources within it.

public static void DeleteCloudService(string subscriptionId, string cloudServiceName)

{

lock (theMassiveLock)

{

Subscription subscription;

if (!allSubscriptions.TryGetValue(subscriptionId, out subscription))

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

CloudService theMatchingCloudService = subscription.CloudServices.SingleOrDefault<CloudService>(cs => String.CompareOrdinal(cs.Name, cloudServiceName) == 0);

if (theMatchingCloudService == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

subscription.CloudServices.Remove(theMatchingCloudService);

}

}

1. GetCloudService: The resource provider should find the cloud service by name and subscription id and return it back as a response.

public static CloudServiceOutput GetCloudServiceBySubscriptionIdAndName(string subscriptionId, string cloudServiceName)

{

lock (theMassiveLock)

{

Subscription subscription;

if (!allSubscriptions.TryGetValue(subscriptionId, out subscription))

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

CloudService theMatchingCloudService = subscription.CloudServices.SingleOrDefault<CloudService>(cs => String.CompareOrdinal(cs.Name, cloudServiceName) == 0);

if (theMatchingCloudService == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

CloudServiceOutput cloudServiceOutput = new CloudServiceOutput()

{

GeoLocation = theMatchingCloudService.Resources.Count > 0 ? theMatchingCloudService.Resources[0].CloudServiceSettings.GeoLocation : String.Empty,

IncarnationId = theMatchingCloudService.IncarnationId,

Resources = new ResourceOutputCollection(theMatchingCloudService.Resources)

};

return cloudServiceOutput;

}

}

## Resource provider contract

To be able to serialize/deserialize objects from the REST call, we need to provide a data contract. In the sample this is implemented in the ResourceProviderContracts.cs in the models folder.

First lets provide the contract for the well known output values.

namespace Microsoft.WindowsAzure.CloudServiceManagement.ResourceProviderCommunication

{

/// <summary>

/// Name of well known Output Values displayed in the Windows Azure Management Portal.

/// </summary>

public static class WellKnownOutputValues

{

public const string ConnectionServerName = "connectionServerName";

public const string ConnectionDatabaseName = "connectionDatabaseName";

public const string ConnectionUserName = "connectionUserName";

public const string ConnectionPassword = "connectionPassword";

}

Next, lets define the operation status data contract

/// <summary>

/// The possible result from an operation.

/// </summary>

[DataContract(Namespace = "http://schemas.microsoft.com/windowsazure")]

public enum OperationResult

{

[EnumMember]

InProgress,

[EnumMember]

Succeeded,

[EnumMember]

Failed

}

### Cloud service settings

These are passed in resource operations

/// <summary>

/// Settings of the cloud service that are sent in the resource-level operations.

/// </summary>

[DataContract(Namespace = "http://schemas.microsoft.com/windowsazure")]

public class CloudServiceSettings : IExtensibleDataObject

{

/// <summary>

/// The geo location of the cloud service.

/// </summary>

[DataMember]

public string GeoLocation { get; set; }

public ExtensionDataObject ExtensionData { get; set; }

}

### Error information

The information sent by any failed operation

/// <summary>

/// Error information about a failed operation.

/// </summary>

[DataContract(Name = "Error", Namespace = "http://schemas.microsoft.com/windowsazure")]

public class ErrorData : IExtensibleDataObject

{

/// <summary>

/// The HTTP error code.

/// </summary>

[DataMember(IsRequired = true)]

public int HttpCode { get; set; }

/// <summary>

/// The error message.

/// </summary>

[DataMember(IsRequired = true)]

public string Message { get; set; }

public ExtensionDataObject ExtensionData { get; set; }

}

### Status information

/// <summary>

/// Status about an operation.

/// </summary>

[DataContract(Namespace = "http://schemas.microsoft.com/windowsazure")]

public class OperationStatus : IExtensibleDataObject

{

/// <summary>

/// The error code. This is not necessarily an integer.

/// </summary>

[DataMember(IsRequired = true)]

public OperationResult Result { get; set; }

/// <summary>

/// The error information for an unhealthy resource.

/// CS manager only passes this field to callers.

/// </summary>

[DataMember]

public ErrorData Error { get; set; }

public ExtensionDataObject ExtensionData { get; set; }

}

### Resource information

Sent: Resource information which is sent to the provider with every resource level operation

/// <summary>

/// Resource information, as sent to the resource provider for the resource-level operations.

/// </summary>

[DataContract(Name = "Resource", Namespace = "http://schemas.microsoft.com/windowsazure")]

public class ResourceInput : IExtensibleDataObject

{

/// <summary>

/// The cloud service settings sent along with this resource input.

/// </summary>

[DataMember]

public CloudServiceSettings CloudServiceSettings { get; set; }

/// <summary>

/// The type of the resource.

/// </summary>

[DataMember]

public string Type { get; set; }

/// <summary>

/// The plan of the resource.

/// </summary>

[DataMember]

public string Plan { get; set; }

/// <summary>

/// The incarnation ID of the resource.

/// </summary>

[DataMember]

public int IncarnationId { get; set; }

/// <summary>

/// The schema version of the intrinsic settings.

/// </summary>

[DataMember]

public string SchemaVersion { get; set; }

/// <summary>

/// The intrinsic settings of the resource.

/// The values and schema of this field are defined by the resource provider.

/// </summary>

[DataMember]

public XmlNode[] IntrinsicSettings { get; set; }

public ExtensionDataObject ExtensionData { get; set; }

}

Received:

The resource information sent back by the resource provider:

/// <summary>

/// Resource information, as returned by the resource provider.

/// </summary>

[DataContract(Name = "Resource", Namespace = "http://schemas.microsoft.com/windowsazure")]

public class ResourceOutput : IExtensibleDataObject

{

/// <summary>

/// The cloud service settings.

/// This field is only filled in by the resource-level APIs (PUT/DELETE resource), not by the cloud-service-level APIs.

/// </summary>

[DataMember]

public CloudServiceSettings CloudServiceSettings { get; set; }

/// <summary>

/// The name of the resource. It is unique within a cloud service.

/// This field is only filled in by the cloud-service-level APIs (PUT/GET/DELETE cloud service).

/// In the resource-level APIs, the resource name is already present in the URI of the request.

/// </summary>

[DataMember]

public string Name { get; set; }

/// <summary>

/// The type of the resource.

/// </summary>

[DataMember]

public string Type { get; set; }

/// <summary>

/// The plan of the resource.

/// </summary>

[DataMember]

public string Plan { get; set; }

/// <summary>

/// The incarnation ID of the resource.

/// </summary>

[DataMember]

public int IncarnationId { get; set; }

/// <summary>

/// The schema version of the intrinsic settings.

/// </summary>

[DataMember]

public string SchemaVersion { get; set; }

/// <summary>

/// The intrinsic settings of the resource.

/// The values and schema of this field are defined by the resource provider.

/// </summary>

[DataMember]

public XmlNode[] IntrinsicSettings { get; set; }

/// <summary>

/// The output of of a resource, can be null.

/// The values and schema of this field are defined by the resource provider.

/// </summary>

[DataMember]

public OutputItemList OutputItems { get; set; }

/// <summary>

/// The state of the resource.

/// </summary>

[DataMember]

public ResourceState State { get; set; }

/// <summary>

/// The sub-state of the resource. The possible values to this field are defined by the resource provider.

/// </summary>

[DataMember]

public string SubState { get; set; }

/// <summary>

/// Status about an operation on this resource.

/// </summary>

[DataMember(IsRequired = true)]

public OperationStatus OperationStatus { get; set; }

public ExtensionDataObject ExtensionData { get; set; }

}

### List of resources

List of resources as returned by the resource provider

/// <summary>

/// List of resources, as returned by the resource provider.

/// </summary>

[CollectionDataContract(Name = "Resources", Namespace = "http://schemas.microsoft.com/windowsazure")]

public class ResourceOutputCollection : List<ResourceOutput>

{

public ResourceOutputCollection()

{

}

public ResourceOutputCollection(IEnumerable<ResourceOutput> resources)

: base(resources)

{

}

public ResourceOutput Find(string resourceType, string resourceName)

{

return this.Find(res =>

string.Equals(res.Type, resourceType, StringComparison.InvariantCultureIgnoreCase) &&

string.Equals(res.Name, resourceName, StringComparison.InvariantCultureIgnoreCase));

}

}

### Cloud Service information:

This is the cloud service information as returned by the resource provider.

[DataContract(Name = "CloudService", Namespace = "http://schemas.microsoft.com/windowsazure")]

public class CloudServiceOutput : IExtensibleDataObject

{

/// <summary>

/// The incarnation ID of this request.

/// </summary>

[DataMember(IsRequired = true)]

public int IncarnationId { get; set; }

/// <summary>

/// The geo location of the cloud service.

/// </summary>

[DataMember(IsRequired = true)]

public string GeoLocation { get; set; }

/// <summary>

/// The resources of the cloud service.

/// </summary>

[DataMember(IsRequired = true)]

public ResourceOutputCollection Resources { get; set; }

public ExtensionDataObject ExtensionData { get; set; }

}

[CollectionDataContract(Name = "CloudServices", Namespace = "http://schemas.microsoft.com/windowsazure")]

public class CloudServiceOutputCollection : List<CloudServiceOutput>

{

public CloudServiceOutputCollection()

{

}

public CloudServiceOutputCollection(IEnumerable<CloudServiceOutput> cloudServices)

: base(cloudServices)

{

}

}

[DataContract(Namespace = "http://schemas.microsoft.com/windowsazure")]

public class OutputItem : IExtensibleDataObject

{

public ExtensionDataObject ExtensionData { get; set; }

[DataMember(EmitDefaultValue = false)]

public string Key { get; set; }

[DataMember(EmitDefaultValue = false)]

public string Value { get; set; }

}

[CollectionDataContract(Name = "Output", ItemName = "OutputItem", Namespace = "http://schemas.microsoft.com/windowsazure")]

public class OutputItemList : List<OutputItem>

{

public OutputItemList()

{

}

public OutputItemList(IEnumerable<OutputItem> outputs)

: base(outputs)

{

}

}

### Resource state enumeration

Lets now add the state enumeration. The system does not enforce this enumeration but will only handle the following values.

public enum ResourceState

{

/// <summary>

/// The resource state is unkown because an error occurred when calling the resource provider.

/// </summary>

Unknown,

/// <summary>

/// The resource provider has no record of this resource.

/// </summary>

NotFound,

/// <summary>

/// The resource is started.

/// </summary>

Started,

/// <summary>

/// The resource is stopped.

/// </summary>

Stopped,

/// <summary>

/// The resource is paused.

/// </summary>

Paused,

}

### Operations contract

As the final step, now add the resource provider operation contract

/// <summary>

/// Resource provider operations.

/// </summary>

[ServiceContract]

public interface IResourceProvider

{

[OperationContract(AsyncPattern = true)]

[WebInvoke(Method = "PUT", UriTemplate = "/subscriptions/{subscriptionId}/cloudservices/{cloudServiceName}/resources/{resourceType}/{resourceName}")]

IAsyncResult BeginPutResource(string subscriptionId, string cloudServiceName, string resourceType, string resourceName, ResourceInput resourceInput, AsyncCallback callback, object state);

ResourceOutput EndPutResource(IAsyncResult result);

[OperationContract(AsyncPattern = true)]

[WebGet(UriTemplate = "/subscriptions/{subscriptionId}/cloudservices/{cloudServiceName}")]

IAsyncResult BeginGetCloudServiceResources(string subscriptionId, string cloudServiceName, AsyncCallback callback, object state);

CloudServiceOutput EndGetCloudServiceResources(IAsyncResult result);

[OperationContract(AsyncPattern = true)]

[WebInvoke(Method = "DELETE", UriTemplate = "/subscriptions/{subscriptionId}/cloudservices/{cloudServiceName}/resources/{resourceType}/{resourceName}")]

IAsyncResult BeginDeleteResource(string subscriptionId, string cloudServiceName, string resourceType, string resourceName, AsyncCallback callback, object state);

ResourceOutput EndDeleteResource(IAsyncResult result);

}

With this the models are now complete and we can move to adding the controllers which will be invoked by the incoming requests and will in turn call the right methods in the models.

## Controllers

### CloudServices controller

Next, let’s add a controller named CloudServices. This controller will manage the CloudServices calls that will be invoked by Windows Azure. Add the following functions to the CloudServices controller:

### GetAllCloudServices(SubscriptionId)

This method will be invoked by Windows Azure when the user has requested to see the list of Cloud Services in a subscription. The response to this call should contain all the Cloud Services that the RP has created resources in.

In most cases, the RP will do some validation and then query the data layer it tracks resources and return the list of Cloud Services.

[HttpGet]

public CloudServiceOutputCollection GetAllCloudServices(string subscriptionId)

{

return DataModel.GetCloudServicesForSubscription(subscriptionId);

}

### GetAllResourcesInCloudService(SubscriptionId, Cloudservicename)

This method is invoked when the user has requested to see all the resources that they have in a particular Cloud Service. This call will be made for each Cloud Service that the user has so that the Azure Developer portal can display a single list of resources across all subscriptions.

In most cases, the resource provider will do some validation and then query their data store and return a list of resources.

[HttpGet]

public CloudServiceOutput GetAllResourcesInCloudService(string subscriptionId, string cloudServiceName)

{

if (String.IsNullOrEmpty(cloudServiceName))

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

return DataModel.GetCloudServiceBySubscriptionIdAndName(subscriptionId, cloudServiceName);

}

### DeleteAllResourcesInCloudService(subscriptionId,cloudservicename)

This method is invoked when the user has requested to delete a CloudService. The resource provider should validate the input parameters and then do the cleanup on their end and delete the resource entries in their tracking database.

[HttpDelete]

public void DeleteAllResourcesInCloudService(string subscriptionId, string cloudServiceName)

{

if (String.IsNullOrEmpty(cloudServiceName))

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

DataModel.DeleteCloudService(subscriptionId, cloudServiceName);

}

With these methods, your Cloud Service controller is finished and your service can handle all required Cloud Service level interactions.

### ResourcesController

This controller will handle all resource level interactions. Add the following methods to it.

#### GetResource(SubscriptionId,CloudServicename,resourcename)

This function is called to display the data for a particular resource. This will be used by the portal to show resource-specific properties. The data returned is controlled by the provider and can contain any data that they want to display.

For the current release, we will be imposing certain limitations on the data that can be returned, such as length limits and key names. These restrictions will be documented soon.

In most cases, the RP will do some validation on the parameters being passed before returning data about a resource:

[HttpGet]

public ResourceOutput GetResource(string subscriptionId, string cloudServiceName, /\*string resourceType,\*/ string resourceName)

{

if (String.IsNullOrEmpty(cloudServiceName) || String.IsNullOrEmpty(resourceName))

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

return DataModel.GetResource(subscriptionId, cloudServiceName, resourceName);

}

#### ProvisionOrUpdateResource(SubscriptionId,Servicename,Resourcename)

This method will be invoked when a resource is provisioned or when it is updated by the user. The RP would do validations on parameters and then provision the resource. The RP will also return a result which will contain the output values as defined by them when the RP’s schema was registered.

If the RP wants, it can also return success immediately and provision the resource asynchronously.

[HttpPut]

public ResourceOutput ProvisionOrUpdateResource(string subscriptionId, string cloudServiceName, /\*string resourceType,\*/ string resourceName, ResourceInput resource)

{

if (String.IsNullOrEmpty(cloudServiceName) /\*|| String.IsNullOrEmpty(resourceType)\*/ || String.IsNullOrEmpty(resourceName) || (resource == null))

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

return DataModel.InsertOrUpdateResource(subscriptionId, cloudServiceName, null, /\*resourceType,\*/ resourceName, resource);

}

#### DeleteResource(SubscriptionId,CloudServicename,resourcename)

This method is invoked by Windows Azure when a user deletes a resource. The RP should do validations on parameters, de-provision the resource from its data layer and delete the tracking records.

[HttpDelete]

public void DeleteResource(string subscriptionId, string cloudServiceName, /\*string resourceType,\*/ string resourceName)

{

if (String.IsNullOrEmpty(cloudServiceName) /\*|| String.IsNullOrEmpty(resourceType)\*/ || String.IsNullOrEmpty(resourceName))

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

DataModel.DeleteResource(subscriptionId, cloudServiceName, resourceName);

}

#### GenerateSSOToken(SubscriptionId,CloudServicename,resourcename)

This method is invoked by Windows Azure when the user requests to be taken to the partner’s management portal. The RP should generate a token based on subscription ID, timestamp (UTC), Cloud Service name, Resource name and a salt that is known only to the RP.

This token should be returned in the response to this call. The Azure portal will then redirect to the SSO URL and provide this token. The RP can then validate the token and return the HTML payload to be displayed in the user’s browser.

TBD