Office365Mon Subscription Management API

Office365Mon provides a set of APIs for managing subscriptions in our service. With it you can do things like create a subscription, change the details about the subscription, modify the list of administrators and notifications for a subscription, configure the resources being monitored for the subscription, and more.

Using the subscription management API requires you to first create an application in your Azure Active Directory and add the Office365Mon applications to it. You'll then reference your application information when requesting an access token that you can use to work with the subscription management API. This process is explained and illustrated in great detail in our API documentation for accessing report data, which you can download from https://www.office365mon.com/Office365Mon_AccessToken_And_API.pdf. It's highly recommended to download that document first to ensure your environment is set up correctly before using this subscription management API.

For resellers there are some special APIs just for you. You need to be registered with Office365Mon.Com to use the reseller APIs, which you can do by contacting us at support@office365mon.com. Once you're registered then you use the Reseller* APIs described at the end of this document to create new subscriptions and add and remove plans for those subscriptions. Once a subscription is actually created though, you can use

the same Subscription Management APIs as everyone else to manage it – change the list of Admins, add or remove notification info, etc.

Contents

Change History 7
Overview 9
Error Handling 10
API Endpoints 10
Subscriptions 10
Create a Subscription 10
Get a Subscription 12
Get My Subscriptions 14
Update Basic Subscription Details 16
Delete Historical Data 17
Administrators 18
Get Subscription Administrators 18
Add a Subscription Administrator 19
Delete a Subscription Administrator 21
Notifications 22
Get Subscription Notifications 22
Add a Subscription Notification 24
Delete a Subscription Notification 25
Get Outage Duration for Notifications 27
Update Outage Duration for Notifications 28
Send a Subscription Notification 29
Webhooks 30
Adding a Webhook 40
Webhook Notification Data Model 40
Processing a Webhook Notification 40
Test a Webhook 41
Office 365 Service Status Changes 42
Get Service Status Values 43
Get the Monitored Service Statuses 44
Set the Monitored Service Statuses 45
Resources 47
Get Subscription Resources 47

Add a Subscription Resource 49
Update the Access Token for a Subscription Resource 51
Delete a Subscription Resource 55
Copy an Access Token for a Subscription Resource 56
Copy All Access Tokens for Subscription Resources 58
Issue a Health Probe for a Resource 59
Health Score Notifications 61
Get Health Score Notification Configuration 61
Update Health Score Notification Configuration 63
Web Sites 65
Get Monitored Web Sites 65
Add Monitored Web Site 67
Delete Monitored Web Site 68
Office 365 Search Monitor 69
Get the Search Monitoring Configuration 69
Update the Search Monitoring Configuration 71
Delete the Search Monitoring Configuration 73
Validate the Search Monitoring SharePoint App Installation 74
Distributed Probes and Diagnostics 75
Get Access Code 75
Delete Access Code 76
Delete All Access Codes 77
Get Remote Agent Monitoring Configuration 78
Update Remote Agent Monitoring Configuration 80
Get Geo Aware Alerts Configuration 81
Update Geo Aware Alerts Configuration 83
Get Document Probe Configuration 85
Update Document Probe Configuration 86
Exchange Online Version Changes 88
Is Version Monitoring Enabled 88
Enable Version Monitor 89
Disable Version Monitor 90
Dashboard Reports 91

Get Dashboard Report Keys 91
Add Dashboard Report Key 93
Delete a Dashboard Report Key 94
Delete All Dashboard Report Keys 95
Email Transport Monitoring 96
Get Email Transport Monitoring Configuration 96
Update Email Transport Monitoring Configuration 98
List Monitoring 100
Monitored List Data Format 100
Get List Monitoring Configuration 101
Set List Monitoring Configuration 102
Stop Monitoring Lists 104
Get Monitored Lists 105
Get Site Lists 107
Add a Monitored List 108
Delete a Monitored List 110
Threat Intelligence Monitoring 111
Get Threat Intelligence Configuration 112
Start Threat Intelligence Monitoring 114
Update Threat Intelligence Configuration 116
Stop Threat Intelligence Monitoring 118
Usage Monitoring 120
Get Usage Monitoring Configuration 120
Update Usage Monitoring Configuration 122
Log Shipping 123
Get Log Shipping Status 124
Toggle Log Shipping Status 126
Internet Egress Change Notifications 127
Get Egress Notification Status 127
Toggle Egress Notification Status 129
Microsoft Teams Monitoring 130
Get Teams IDs 130
Add Or Update Teams Monitoring Configuration 132

Get Teams Monitoring Configuration 133
Remove Teams Monitoring Configuration 135
Resellers 136
Reseller Branding Requirements 140
Associated Subscriptions 141
Create a Subscription 142
Delete a Subscription 144
Get Associated Subscriptions 145
Get My Reseller Subscriptions 147
Get Plans 149
Add a Plan to a Subscription 151
Delete a Plan from a Subscription 153
Get Customer Plans 154
Convert a Trial Subscription to a Paid Subscription 156
Get Distributed Probe Configuration for One Subscription 158
Get Distributed Probe Configuration for All Subscriptions 160

Change History

June 8, 2015	Initial release
September 3, 2015	Add support for Reseller APIs
January 11, 2016	Refactored Add Subscription Resource, Update Access Token and Issue Health Probe for any Office 365 Resource because of unannounced breaking changes by the Microsoft Azure and ADAL teams. Added new APIs to Copy Token and Copy Tokens.
January 19, 2016	Added the Delete Historical Data REST API.
February 29, 2016	Added support for webhooks.
March 14, 2016	Added support for managing the Office 365 service status changes.
April 13, 2016	Added support for managing the Office 365 Search Monitor feature.
August 13, 2016	Added support for managing the Distributed Probes and Diagnostics access codes as well as the Office 365 Help Desk feature.
November 9, 2016	Added additional support in UpdateAccessToken for Web Sites. Added support for managing Web Sites.
December 2, 2016	Added support for checking, enabling and disabling monitoring version changes in Exchange Online.
December 7, 2016	Added support for managing Dashboard Report keys.
January 7, 2017	Added additional info on using the Reseller APIs.
January 12, 2017	Added support for managing Distributed Probe agent monitoring.
February 12, 2017	Added support for managing Email Transport monitoring.
February 27, 2017	Added support for having notes with each Dashboard Reports key.
March 20, 2017	Added support for managing List Monitoring. Removed documentation for Office 365 Help Desk features since it has been deprecated.
May 10, 2017	Added support for OneDrive sites as a monitored resource.
June 27, 2017	Documented support for additional regional data centers for Office365Mon customers.
August 14, 2017	Added support for resellers to add Associated Subscriptions.
August 23, 2017	Added support to manage AzureServiceMon subscriptions.
September 21, 2017	Added support to manage metric monitoring in AzureServiceMon subscriptions.
November 30, 2017	Added support to manage Threat Intelligence monitoring.
January 2, 2018	Added an additional webhook to track issues with email transport monitoring.
January 25, 2018	Added support for additional Threat Intelligence monitoring on SharePoint and OneDrive for Business.
January 30, 2018	Added new API for Resellers to let them convert a trial subscription to a paid subscription.
March 2, 2018	Added support for managing Log Shipping of Office 365 SharePoint and OneDrive logs to Office365Mon for historical preservation and reporting.
March 8, 2018	Added additional webhook notification types.
August 8, 2018	Added support for configuring slow network notifications. Added support for configuring Geo Aware Alerts. Added additional webhook notification types.
November 30, 2018	Added support for Office365Mon Private deployments.
February 4, 2019	Added support for managing Internet egress change notifications.
May 13, 2019	Added support for managing Microsoft Teams monitoring configuration.

July 22, 2019	Updated the webhook notification information section to reflect the new types of alerts you can get with call quality monitoring. Removed the AzureServiceMon documentation.
October 21, 2019	Added new APIs for resellers to get Distributed Probe configuration information for the subscriptions they own.
December 13, 2019	Added new APIs and webhook notification types for document probes.
January 10, 2020	Added new webhook notifications for daily processing jobs.

Overview

The subscription management API is designed to let you do all of the required steps to create and maintain Office365Mon, and now, AzureServiceMon subscriptions. The order of the document follows the same steps you go through when creating and/or managing a subscription in the web site at https://www.office365mon.com.

Support has been added for AzureServiceMon (https://azureservicemon.com) because virtually all of the same concepts are the same. You create subscriptions, and you configure them for monitoring. You can manage the notifications, subscription admins, etc. the same exact way as you do for Office365Mon.

The basic steps you'll want to go through to create a new subscription are:

1. Create a subscription. If you're an Office 365 reseller remember to use the ResellerCreateSub API to create subscriptions.
2. Add additional subscription administrators, if needed. If you're a reseller then typically you add one or more accounts that you own, and at least one account in the new Office 365 tenant.
3. Add the notification options you want for the subscription – e.g. the email addresses and texting phone numbers that should be alerted in the event of an outage.
4. Add the resources that you want to be monitored by the subscription. When you add a resource, you'll want to provide the credentials of an account that has permissions to the resource. However, that account has to located in an Azure Active Directory tenant that has already gone through the Azure consent process for the Office365Mon application, otherwise we won't be able to successfully get an access token for the resource. You can always set up the resources and then later after the account has gone through the Azure consent process, you can use the update an access token API to fetch a new access token for that resource.

We've also included additional APIs for the management of the subscription, such as changing any of the information described above, as well as manually issuing health probes to both a resource we are monitoring for you, as well as any Office 365 resource to which you have permission.

Error Handling

Some of the APIs simply return an HTTP error response when there's an issue – typically a 409 – Conflict and some additional information in the ReasonPhrase property of an HttpResponseMessage (if you are developing with .NET). Other times when there may be multiple reasons why a call failed, the API returns more detailed error information – both an error number and message. The code samples illustrate when more detailed error information is available, and how to extract both the error number and message. Following is a list of possible error numbers and what they mean:

Error Number	Description
100 – Unexpected error	A general error; details are included in the ErrorMessage.
230 – Parameter out of bounds	The value for one or more of the parameters provided was out of bounds of possible values. The valid values are described in the documentation for each parameter in each REST API.
240 – Configuration Incomplete	A feature is not fully configured yet, and the REST API you are calling requires it to be configured first.

API Endpoints

All of the API endpoints are described below for the default data center location, which most customers will be in. However, if your subscription is in a regional data center then you'll need to replace the default root URL with the one for your data center. For example, you would replace *https://www.office365mon.com* as follows:

- Germany Data Center: *https://office365mon.de*

Subscriptions

The operations in this section all have to do with managing general subscription details.

Create a Subscription

Use this method to create a new subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/createsub

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
UPN	String	The UPN for the initial administrator of the Office365Mon subscription.
CompanyName	String	The company whose name the subscription is for.
ContactEmail	String	An email address that can be used for pricing changes, feature changes, etc. for the subscription
SubscriptionType	Int	**OPTIONAL.** If this parameter is not included, or if it is 0, an Office365Mon subscription is created. If it is 1, an AzureServiceMon subscription is created.

Returns:

Name	Type	Description
SubscriptionId	Guid	The subscription ID for the new Office365Mon subscription.

The call returns a subscription ID if it works. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("UPN", "fred@contoso.com"));
vals.Add(new KeyValuePair<string, string>("CompanyName", "Contoso"));
vals.Add(new KeyValuePair<string, string>("ContactEmail", "fred@contoso.com"));


//we can leave this out to make an Office365Mon subscription, but in this
//case we're going to make it an AzureServiceMon subscription
vals.Add(new KeyValuePair<string, string>("SubscriptionType", "1"));
```

```
//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "createsub");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	string SubscriptionId = data["SubscriptionId"].ToString();
}
else
{
	//the operation failed
}
```

Get a Subscription

Use this method to get subscription details.

REST Endpoint:

https://www.office365mon.com/api/developers/getsub

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being queried.

Returns:

Name	Type	Description
CompanyName	String	The name of the company for the subscription.
ContactEmail	String	The email address that will be used to contact the subscription owner in the case of pricing changes,

		feature changes, licensing changes, general announcements, etc.
DateCreated	DateTime	The date the subscription was created.
IsActive	Int	An integer representing the active state of the subscription.

Active State	Code
Inactive	0
Active	1

MonitorVersions	Int	An integer describing whether version changes for Office 365 resources should be monitored

Monitoring State	Code
Don't monitor for and notify me of changes in the version of my Office 365 resources.	0
Do monitor for and notify me of changes in the version of my Office 365 resources.	1

If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "getsub");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string subData = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(subData);
```

```
        string CompanyName = data["CompanyName"].ToString();
        string ContactEmail = data["ContactEmail"].ToString();
        DateTime DateCreated DateTime.Parse(data["DateCreated"]);
        string IsActive = data["IsActive"].ToString();
        string MonitorVersions = items ["MonitorVersions"].ToString();
}
else
{
        //the operation failed
}
```

Get My Subscriptions

Use this method to get details of all subscriptions on which you are a subscription administrator.

REST Endpoint:

https://www.office365mon.com/api/developers/mysubs

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.

Returns:

An array of information about each subscription:

Name	Type	Description	
CompanyName	String	The name of the company for the subscription.	
ContactEmail	String	The email address that will be used to contact the subscription owner in the case of pricing changes, feature changes, licensing changes, general announcements, etc.	
DateCreated	DateTime	The date the subscription was created.	
IsActive	Int	An integer representing the active state of the subscription.	
		Active State	**Code**
		Inactive	0
		Active	1

MonitorVersions	Int	An integer describing whether version changes for Office 365 resources should be monitored	
		Monitoring State	**Code**
		Don't monitor for and notify me of changes in the version of my Office 365 resources.	0
		Do monitor for and notify me of changes in the version of my Office 365 resources.	1

If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "mysubs");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string subData = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	Array data = jss.DeserializeObject(subData) as Array;

	foreach (var sub in data)
	{
		Dictionary<string, object> items = sub as Dictionary<string, object>;

		string CompanyName = items["CompanyName"].ToString();
		string ContactEmail = items ["ContactEmail"].ToString();
		DateTime DateCreated DateTime.Parse(items ["DateCreated"]);
		string IsActive = items ["IsActive"].ToString();
		string MonitorVersions = items ["MonitorVersions"].ToString();
	}
}
```

```
else
{
        //the operation failed
}
```

Update Basic Subscription Details

Use this method to update the company name and/or contact email for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/updatesub

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
CompanyName	String	The company whose name the subscription is for.
ContactEmail	String	An email address that can be used for pricing changes, feature changes, etc. for the subscription
MonitorVersions	Int	A value of 0 means don't monitor for and notify me of changes in the version of my Office 365 resources; a value of 1 means notify me.

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
```

```
vals.Add(new KeyValuePair<string, string>("CompanyName", "Contoso"));
vals.Add(new KeyValuePair<string, string>("ContactEmail", "fred@contoso.com"));
vals.Add(new KeyValuePair<string, string>("MonitorVersions", "1"));

//make the post to update the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "updatesub");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Delete Historical Data

Use this method to delete the historical data for a subscription. It deletes the following items:

- Recent performance data of health probes against the subscription resources
- All monthly performance statistics for subscription resources
- All outages for subscription resources
- All monthly outage statistics for subscription resources

REST Endpoint:

https://www.office365mon.com/api/developers/deletehistoricaldata

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";
```

```
//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletehistoricaldata");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Administrators

The operations in this section all have to do with managing administrators for a subscription.

Get Subscription Administrators

This method returns a semi-colon delimited list of administrators for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getadmins

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being queried.

Returns:

Name	Type	Description
AdminUPNs	ArrayList	An ArrayList of administrator UPNs for the subscription.

If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the subscription admins
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "getadmins");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string admins = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(admins);

	ArrayList adminList = data["AdminUPNs"] as ArrayList;

	foreach (var admin in adminList)
	{
		//admin = a UPN of an admin
	}
}
else
{
	//the operation failed
}
```

Add a Subscription Administrator

Use this method to add a subscription administrator.

REST Endpoint:

https://www.office365mon.com/api/developers/addadmins

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
AdminUPNs	String	A JSON-formated array of UPNs to add as administrators of the subscription, i.e. ["speschka@office365mon.com"] or ["speschka@office365mon.com","stevep@office365mon.com"].

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//this is just one method for supporting adding multiple items at once; you can use
//whatever method works for you
string[] admins = new string[2] { "speschka@officew365mon.com", "stevep@office365mon.com"
};

//this is used to convert a .NET string array into a JSON-formatted single string for
POST'ing
JavaScriptSerializer jss = new JavaScriptSerializer();

vals.Add(new KeyValuePair<string, string>("AdminUPNs",
jss.Serialize(admins).ToString()));

//make the post to add the admins
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "addadmins");
```

```
//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Delete a Subscription Administrator

Use this method to delete a subscription administrator.

REST Endpoint:

https://www.office365mon.com/api/developers/deleteadmin

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
AdminUPNs	String	The UPN of the admin to delete from the subscription. This accepts a single value only, not an array.

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
```

```
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("AdminUPNs", "fred@contoso.com"));

//make the post to delete the admin
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "deleteadmin");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Notifications

The operations in this section all have to do with managing who or what is notified when there is an outage, such as email addresses and phone numbers for text messages. You can also use a webhook for notifications, allowing you to programmatically process a notification for outages or Office 365 service status changes.

Get Subscription Notifications

Use this method to get all of the notification points for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getnotifications

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being queried.

Returns:

An array of objects that each contain information about one notification point: address, description, notification type description, and notification type code.

Name	Type	Description
Address	String	The address the notification should go to. It can be an email address, a mobile phone number that can receive text messages, or the Url to a webhook endpoint.
Description	String	A description of the notification endpoint.
TypeDescription	String	A description of the notification type.
TypeCode	Int	An int that represents the type of address it is. The possible TypeCodes are:

Address Type	Code
Email	0
Text	1
Webhook	2

If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list of notifications
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getnotifications");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string notifications = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(notifications);
```

```
        if (data.ContainsKey("NotificationList"))
        {
                ArrayList noteList = data["NotificationList"] as ArrayList;

                List<NotificationInfo> notificationInfos = new List<NotificationInfo>();

                foreach (var note in noteList)
                {
                        Dictionary<string, object> noteInfo = note as Dictionary<string,
                        object>;

                        string address = noteInfo["Address"].ToString();
                        string description = noteInfo["Description"].ToString();
                        string typeDescription = noteInfo["TypeDescription"].ToString();
                        int typeCode = (int)noteInfo["TypeCode"];
                }
        }
        else
        {
                //there aren't any
        }
}
else
{
        //the operation failed
}
```

Add a Subscription Notification

Use this method to add a notification endpoint to the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/addnotification

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
Address	String	The address the notification should go to. It can be an email address, a mobile phone number that can receive text messages, or the Url to a webhook endpoint.
Description	String	A description of the notification endpoint.
TypeCode	Int	An int that represents the type of address it is. The possible TypeCodes are:

Address Type	Code

		Email	0
		Text	1
		Webhook	2

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("Address", "fred@contoso.com"));
vals.Add(new KeyValuePair<string, string>("Description", "Home email address"));
vals.Add(new KeyValuePair<string, string>("TypeCode", "0"));

//make the post to add the Email Address notification (because TypeCode = 0)
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "addnotification");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Delete a Subscription Notification

Use this method to delete a notification endpoint for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/deletenotification

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
Address	String	The address the notification should go to. It can be an email address, a mobile phone number that can receive text messages, or the Url to a webhook endpoint.

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("Address", "fred@contoso.com"));

//make the post to delete the notification
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletenotification");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Get Outage Duration for Notifications

Use this method to get the duration, in minutes, that an outage must have before sending notifications. A value of 0 means that notifications are sent on every outage.

REST Endpoint:

https://www.office365mon.com/api/developers/getoutageduration

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being queried.

Returns:

An integer value – in minutes – of the minimum outage duration before notifications are sent. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list of notifications
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getoutageduration");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//this is used to convert our JSON data into a dictionary of values
	string notifications = await hrm.Content.ReadAsStringAsync();

	//you can convert this value into an integer
}
else
{
```

```
        //the operation failed
}
```

Update Outage Duration for Notifications

Use this method to update the duration, in minutes, that an outage must have before sending notifications. A value of 0 means that notifications are sent on every outage.

REST Endpoint:

https://www.office365mon.com/api/developers/updateoutageduration

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being updated.
OutageDuration	Int	An int that represents how long an outage duration needs to run before notifications are sent out. If the value is zero then notifications are sent on every outage, even if it's less than a minute.

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("OutageDuration","2"));

//make the post to get the list of notifications
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"updateoutageduration");
```

```
//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked!
}
else
{
	//the operation failed
}
```

Send a Subscription Notification

Use this method to send a notification to all of the email addresses and text phone numbers for the subscription. This is typically used when you have a custom application developed for Office 365 running in a tenant being monitored by Office365Mon. It allows you to use the Office365Mon notification infrastructure for your own custom events.

REST Endpoint:

https://www.office365mon.com/api/developers/sendnotification

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
EmailSubject	String	The Subject of the email being sent. If you are only sending a text message this can be blank.
EmailBody	String	The body of the email message being sent. If you are only sending a text message this can be blank.
TextMessage	String	The text message to send. If you are only sending an email message this can be blank.
MessageType	Int	An int that represents the type of message to send. The possible MessageType codes are:

Address Type	Code
Both	0
Email only	1
Text only	2

Returns:

If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 400 – Bad Request. You can look at the content from the response to parse out more details about the error.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("EmailSubject", "New Email from Me"));
vals.Add(new KeyValuePair<string, string>("EmailBody", "This is the email body!"));
vals.Add(new KeyValuePair<string, string>("TextMessage", "Here's a text message!"));
vals.Add(new KeyValuePair<string, string>("MessageType", "0"));

//make the post to add the Email Address notification (because TypeCode = 0)
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"sendnotification");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//when there is an error you will get info about it in the results
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	//get the error number and message from the API
	int errNum = int.Parse(data["ErrorNumber"].ToString());
	string errMsg = data["ErrorMessage"].ToString();
	//the HTTP response code is here: (int)hrm.StatusCode

	//this DIDN'T work - take the appropriate action
}
```

Webhooks

Webhooks allow you to programmatically receive notifications. The webhooks from Office365Mon notify you when an outage is starting, when an outage is ending, and when events you have signed up to

monitor using the Office 365 Service info occur; for example when an Office 365 feature changes to the ***Service Interruption*** status. To use a webhook with Office365Mon, you can need to host an anonymous HTTPS endpoint that supports POST'ing a JSON payload. You will be able to verify the payload by the SubscriptionId attribute – it is the SubscriptionId of the Office365Mon subscription that has triggered the notification.

Your webhook should return an HTTP 200 status as quickly as possible, but Office365Mon does not act upon the return value in any way. Webhooks are fired for every event and you can choose whether or not to process a particular notification. However, if your webhook fails to respond or otherwise returns an HTTP status code of something other than 200, Office365Mon will not attempt to send the webhook again. Where the status code you return is used is when your webhook is tested from Office365Mon; the HTTP code and reason from your webhook endpoint is returned from the Office365Mon test webhook method.

The JSON payload is based on this class definition:

```
public class WebhookNotification
{
    public enum NotificationType
    {
        Test,
        Outage,
        ServiceStatusChange,
        QueryLatency,
        QueryResultsChanged,
        QueryNoResults,
        LongRunningProbe,
        VersionChange,
        DpdHostOffline,
        InboundMessageDelay,
        OutboundMessageDelay,
        MonitoredListOverMaxSize,
        MonitoredListOverMaxRenderTime,
        AzureMetricAlert,
        ThreatIntelligenceAlert,
        EmailTransportMonitoringIssue,
        MonitoredResourceDeleted,
        AccessTokenIssue,
        SharePointHealthInfo,
        SlowNetwork,
        DistributedGeoProximityOutage,
        UsageExceeded,
        SlowPing,
        IpEgressChanged,
        MediaDestinationIssue,
        MediaQualityIssue,
        ProbeForbidden,
        DailyProcessingJob
    }


    public Guid SubscriptionId { get; set; }
    public string CompanyName { get; set; }
    public int WebhookNotificationType { get; set; }
    public OutageInfo NotificationOutageInfo { get; set; }
```

```
public ServiceStatusInfo NotificationServiceStatusInfo { get; set; }
public QueryInfo NotificationQueryInfo { get; set; }
public LongRunningProbeInfo NotificationLongRunningProbeInfo { get; set; }
public VersionInfo NotificationVersionInfo { get; set; }
public OfflineHostInfo NotificationOfflineHostInfo { get; set; }
public MailTransportDelay NotificationMailTransportInfo { get; set; }
public MonitoredListInfo NotificationMonitoredListInfo { get; set; }
public AzureMetricAlertInfo NotificationAzureMetricAlertInfo { get; set; }
public ThreatIntelligenceAlertInfo NotificationThreatIntelligenceInfo { get; set; }
public MonitoredResourceDeletedInfo
    NotificationMonitoredResourceDeletedInfo { get; set; }
public AccessTokenIssueInfo NotificationAccessTokenIssueInfo { get; set; }
public SharePointHealthInfo NotificationSharePointHealthInfo { get; set; }
public DistributedGeoProximityOutageInfo
    NotificationDistributedGeoProximityOutageInfo { get; set; }
public UsageExceededInfo NotificationUsageExceededInfo { get; set; }
public SlowPingInfo NotificationSlowPingInfo { get; set; }
public IpEgressChangeInfo NotificationIpEgressChangeInfo { get; set; }
public MediaDestinationIssueInfo NotificationMediaDestinationIssueInfo { get; set; }
public MediaQualityIssueInfo NotificationMediaQualityIssueInfo { get; set; }
public ProbeForbiddenInfo NotificationProbeForbidden { get; set; }
public DailyProcessingJobInfo NotificationDailyProcessingJobInfo { get; set; }


public class OutageInfo
{
    public enum NotificationType
    {
        OutageStarting,
        OutageEnding
    }

    public enum OutageResourceType
    {
        SharePoint = 0,
        Exchange = 1,
        SkypePresence = 4,
        SkypeIM = 5,
        WebSite = 6,
        OneDrive = 10,
        PowerBI = 11,
        AzureResource = 12
    }

    public string Resource { get; set; }
    public int ResourceType { get; set; }
    public int OutageNotificationType { get; set; }
    public string DistributedProbeHostName { get; set; }
}

public class ServiceStatusInfo
{
    public string ServiceName { get; set; }
    public string FeatureName { get; set; }
    public string CurrentStatus { get; set; }
    public string PreviousStatus { get; set; }
}
```

```
public class QueryInfo
{
    public string Resource { get; set; }
    public int NumResults { get; set; }
    public string Results { get; set; }
}

public class LongRunningProbeInfo
{
    public string DistributedProbeHostName { get; set; }
    public int Duration { get; set; }
    public int Threshold { get; set; }
}

public class VersionInfo
{
    public string OldVersion { get; set; }
    public string NewVersion { get; set; }
    public int ResourceType { get; set; }
    public string ResourceAddress { get; set; }
}

public class OfflineHostInfo
{
    public string DistributedProbeHostName { get; set; }
    public DateTime LastProbeTime { get; set; }
    public int MaxOfflineTime { get; set; }
}

public class MailTransportDelay
{
    public enum NotificationType
    {
        MaxDeliveryTimeExceeded,
        MessageDeliveryCompleted,
        TransportTestNotCompleted
    }

    public DateTime TimeSent { get; set; }
    public int MinutesToDeliver { get; set; }
    public NotificationType DelayInformationType { get; set; }
}

public class MonitoredListInfo
{
    public string ListName { get; set; }
    public string SiteAddress { get; set; }
    public int MaxListSize { get; set; }
    public int ActualListSize { get; set; }
    public int MaxRenderTime { get; set; }
    public int ActualRenderTime { get; set; }
}

public class AzureMetricAlertInfo
{
    public string ResourceName { get; set; }
    public string ResourceId { get; set; }
    public Guid AzureSubscriptionId { get; set; }
```

```
    public string MetricFriendlyPropertyName { get; set; }
    public string MetricInternalPropertyName { get; set; }
    public double MetricValue { get; set; }
    public double MetricAlertValue { get; set; }
}

public class ThreatIntelligenceAlertInfo
{
    public enum ThreatNotificationType
    {
        FirstOccurrence,
        ExceededGeneralMalwareCount,
        ExceededUserMalwareCount,
        ExceededUploadMalwareCount
    }

    public int NotificationType { get; set; }
    public string MalwareFamily { get; set; }
    public int MalwareCount { get; set; }
    public DateTime MessageTime { get; set; }
    public string Recipients { get; set; }
    public string Sender { get; set; }
    public string SenderIp { get; set; }
    public string DetectionType { get; set; }
    public string Verdict { get; set; }
}

public class MonitoredResourceDeletedInfo
{
    public string MrAddress { get; set; }
    public int MrResourceType { get; set; }
    public DateTime MrDeletedOn { get; set; }
}

public class AccessTokenIssueInfo
{
    public int AtiIssueType { get; set; }
    public string AtiAddress { get; set; }

    /* IssueType will be one of the following:
    *NoAuthenticationResult,
    *RefreshTokenNearExpiration,
    *RefreshTokenExpired,
    *AccessExpiredOrRevoked,
    *UnknownIssue,
    *WrongUserAccessToken,
    *SkypePoolChanged,
    *AzureSecuredSiteRefreshTokenNearExpiration,
    *EmailTransportRefreshTokenNearExpiration
    */
}

public class SharePointHealthInfo
{
    public int ShnResourceType { get; set; }
    public int ShnHealthLimit { get; set; }
    public int ShnHealthScore { get; set; }
    public int ShnRequestLimit { get; set; }
```

```
    public int ShnRequestScore { get; set; }
}


public class DpdGeoAlert
{

    public enum AlertRadiusUnits
    {
        Miles,
        Kilometers
    }


    public Guid SubscriptionId { get; set; }
    public double Distance { get; set; }
    public int AlertRadius { get; set; }
    public int Units { get; set; }
    public string UnitType { get; set; }
}


public class DistributedGeoProximityOutageInfo
{
    public List<DpdGeoAlert> DgpoImpactedHosts { get; set; }
}

public class UsageExceededInfo
{
    public int UeiStorageType { get; set; }
    public double UeiNotificationThreshold { get; set; }
    public List<MailboxUsageDetail> UeiMailboxes { get; set; }
    public List<SharePointSiteUsageDetail> UeiSharePointSites { get; set; }
    public List<OneDriveSiteUsageDetail> UeiOneDriveSites { get; set; }
}

public class MailboxUsageDetail
{
    public string UserPrincipalName { get; set; }
    public string DisplayName { get; set; }
    public string DeletedDateValue { get; set; }
    public DateTime CreatedDate { get; set; }
    public long ItemCount { get; set; }
    public double IssueWarningQuota { get; set; }
    public double ProhibitSendQuota { get; set; }
    public double ProhibitSendReceiveQuota { get; set; }
    public DateTime ReportRefreshDate { get; set; }
    public bool IsDeleted { get; set; }
    public string LastActivityDateValue { get; set; }
    public DateTime LastActivityDate { get; set; }
    public double StorageUsed { get; set; }
    public int ReportPeriod { get; set; }
}

public class SharePointSiteUsageDetail
{
    public long PageViewCount { get; set; }
    public long VisitedPageCount { get; set; }
```

```
    public string RootWebTemplate { get; set; }
    public string SiteUrl { get; set; }
    public string OwnerDisplayName { get; set; }
    public long FileCount { get; set; }
    public long ActiveFileCount { get; set; }
    public double StorageAllocated { get; set; }
    public DateTime ReportRefreshDate { get; set; }
    public bool IsDeleted { get; set; }
    public string LastActivityDateValue { get; set; }
    public DateTime LastActivityDate { get; set; }
    public double StorageUsed { get; set; }
    public int ReportPeriod { get; set; }
}

public class SharePointSiteUsageDetail
{
    public string SiteUrl { get; set; }
    public string OwnerDisplayName { get; set; }
    public long FileCount { get; set; }
    public long ActiveFileCount { get; set; }
    public double StorageAllocated { get; set; }
    public DateTime ReportRefreshDate { get; set; }
    public bool IsDeleted { get; set; }
    public string LastActivityDateValue { get; set; }
    public DateTime LastActivityDate { get; set; }
    public double StorageUsed { get; set; }
    public int ReportPeriod { get; set; }
}

public class SlowPingInfo
{
    public int SpiServiceType { get; set; }
    public string SpiSiteAddress { get; set; }
    public long SpiDuration { get; set; }
    public long SpiThreshold { get; set; }
    public string SpiDistributedProbeHostName { get; set; }
}

public class IpEgressChangeInfo
{
    public string IeciOldEntryPoint { get; set; }
    public string IeciNewEntryPoint { get; set; }
    public string IeciDistributedProbeHostName { get; set; }
}

public class MediaDestinationIssueInfo
{
    public string MdiHostName { get; set; }
    public List<MediaDestinationIssue> MdiIssues { get; set; }

    public MediaDestinationIssueInfo()
    {
        this.MdiIssues = new List<MediaDestinationIssue>();
    }
}

public class MediaQualityIssueInfo
{
```

```
    public int MqiIssueType { get; set; }
    public string MqiIssueDescription { get; set; }
    public double MqiDuration { get; set; }
    public double MqiThreshold { get; set; }
    public string MqiDistributedProbeHostName { get; set; }
}

public class ProbeForbiddenInfo
{
    public string PfSiteAddress { get; set; }
}

public class DailyProcessingJobInfo
{
    public enum StatusChangeType
    {
        ServiceProcessStarting,
        JobStarting,
        JobCompleted
    }

    public int DpjJobStatusChangeType { get; set; }
}
```

When a webhook is sent to your endpoint, the common attributes you can check first are the Office365Mon SubscriptionId, the CompanyName (which is what you provided for the subscription), and the WebhookNotificationType. You can tell what kind of notification it is based on the WebhookNotificationType:

- 0 = this is a test webhook notification. You could have sent this from the Configure Office365Mon web page on the Office365Mon site, or using the Subscription Management API.
- 1 = this is an outage notification. An outage has either started or ended. For more details look in the **OutageInfo** data.
- 2 = this is a service status change notification. For example, the Office 365 Service Info integration API has reported that a feature has entered the ***Service Interruption*** status. For more details look in the **ServiceStatusInfo** data.
- 3 = this is a query latency notification. A query has taken longer to return results than the threshold you defined in the Configure Office 365 Search Monitoring page. For more details look in the **QueryInfo** data.
- 4 = query results changed. This means that the set of results returned from querying the SharePoint Online site have changed since the last time a query was executed. For more details look in the **QueryInfo** data.
- 5 = no query results. This means that no results were returned from the last query executed against the SharePoint Online site. For more details look in the **QueryInfo** data.
- 6 = you have a long running probe. This is based on the threshold you set when you installed the Office365Mon Distributed Probes and Diagnostics Service. At each location you install it you can define how long a health probe can take before it's considered to be "long running". If the probe takes longer than that, and if there's a working Internet connection at that location, then this webhook notification is sent. The payload will include information about the host which sent out the notification. For more details look in the **LongRunningProbeInfo** data.

- 7 = the version of one of the tenant resources has changed. That means a new version of software has been pushed out by Microsoft to either your SharePoint Online or Exchange Online tenant. For more details look in the **VersionInfo** data.
- 8 = one of the Distributed Probe and Diagnostics agents has entered an offline state. That happens when it hasn't successfully recorded a health probe with the cloud service for specific number of minutes. The exact number of minutes is configured on the Configure Office 365 Distributed Probes page in the web site, or using the updatedpdmonitorconfig API. For more details look in the **OfflineHostInfo** data.
- 9 = an email sent to monitor the inbound email transport infrastructure has either not been delivered within the maximum time configured before a notification is sent, or, an email that triggered a notification previously has been delivered. The MailTransportDelay NotificationType property indicates which of these two conditions triggered the webhook. For more details look in the **MailTransportDelay** data.
- 10 = an email sent to monitor the outbound email transport infrastructure has either not been delivered within the maximum time configured before a notification is sent, or, an email that triggered a notification previously has been delivered. The MailTransportDelay NotificationType property indicates which of these two conditions triggered the webhook. For more details look in the **MailTransportDelay** data.
- 11 = a health probe was run on a monitored list, and it took longer to render the default view than the maximum time according to the list monitoring configuration. The exact number of seconds is configured on the Configure List Monitoring page in the web site, or using the updatelistmonconfig API. For more details look in the **MonitoredListInfo** data.
- 12 = a monitored list has more items than the maximum desired according to the list monitoring configuration. The exact number of items is configured on the Configure List Monitoring page in the web site, or using the updatelistmonconfig API. For more details look in the **MonitoredListInfo** data.
- 13 = an Azure metric has returned a value in excess of the threshold at which you asked to be notified. For more details look in the **AzureMetricAlertInfo** data.
- 14 = a Threat Intelligence alert. It could be because a malware has been detected for the first time in your organization, you've exceeded a threshold for number of malwares detected in a given time period, or a user has received more than a certain number of malwares during the current day. The thresholds for overall malwares in a given time period and number per user per day are configurable for the subscription. For more details look in the **ThreatIntelligenceAlertInfo** data.
- 15 = an email transport monitoring test has not completed successfully in at least 12 hours. If you are monitoring the email transport and either an inbound or outbound test has not completed successfully in the last 12 hours, this notification will be fired. Generally when this happens you should look at Recent Email Transport Errors report in the Office365Mon.Com Advanced Reports gallery for errors. When this webhook is fired, the TimeSent property indicates the last time a test completed successfully. Note that if you are monitoring both inbound and outbound, this webhook fires when ***either*** has not completed in the last 12 hours. That means that one direction may be working but the other may not. For more details look in the **MailTransportDelay** data.
- 16 = a monitored resource was deleted by the Office365Mon service. Resources are deleted for the following reasons:
    - your monitoring feature has expired

    - the access token you obtained for monitoring a resource was obtained by the wrong person, i.e. someone that doesn't have rights to the resource
    - your access token used to monitor the resource has been expired for several days and we can no longer monitor it for you
- 17 = we are having access token issues with a resource. The most common causes are an access token is about to expire, has expired, and/or has been revoked.
- 18 = a SharePoint health score metric exceeds the user configured notification value. For example, notifications are requested for a Request Duration exceeding 3000 milliseconds, and a request took 3500 milliseconds to process. Both the notification levels and the actual values are returned in the data.
- 19 = a slow network response time was detected. "Slow" means the network time was longer than the configured alert threshold for a slow network either for cloud-based probes or any Distributed Probe agent.
- 20 = an outage was reported by an Office365Mon customer's Distributed Probe agent that was within "x" number of miles or kilometers of one or more of your Distributed Probe agents. "x" is a value that has been configured on the Office365Mon web site on the Configure Distributed Probes page or via the Office365Mon REST API.
- 21 = one or more resources with storage limits – Exchange, SharePoint, and/or OneDrive for Business – are within the thresholds configured of reaching their storage limit.
- 22 = a resource – Dns, Proxy, and/or Office 365 service – recorded a ping test that is slower than the notification value configured for it at a particular location.
- 23 = the Internet egress point at a location where a Distributed Probe and Diagnostics agent is installed has changed.
- 24 = a Distributed Probe agent was unable to reach one or more Office 365 telephony service endpoints using one or more port and protocol combinations. These endpoints are used to for Skype for Business and Microsoft Teams calling. If you are unable to reach a service endpoint via the port and protocol then users may not be able to complete calls, call connection time may suffer, etc.
- 25 = one of the call quality metrics reported a value above the threshold you configured for a particular agent. For example, after running a test there may have been 40 milliseconds of jitter, and you asked to be notified if it exceeded 30 milliseconds.
- 26 = a document probe was attempted, but was denied with a409 – Forbidden error. That typically means that the Office365Mon SharePoint App that is used for document probes and Search monitoring has been uninstalled.
- 27 = the daily processing job for all subscriptions has started, or for a specific subscription has started or ended. Daily processing jobs run for each subscription to summarize the current day's performance and availability data into daily totals, and daily totals into monthly totals. This webhook gives you notification so that you can run your own code if desired when the daily processing job starts, because when it is completed the current day's performance and availability data is deleted. The ServiceProcessStarting hook fires first when the overall process for the service starts, and then the JobStarting hook is fired for each subscription when its data is being processed, and then the JobCompleted hook fires when that subscription's processing is complete.

If the notification is for an outage starting or ending, the notification includes the resource experiencing the outage – meaning, the SharePoint site Url or Exchange mailbox UPN. If the notification is for a service status change, it includes the service name (e.g. Exchange Online), the feature name (e.g. Sign-

in), the current status now, and the status it had prior to entering the current status. If the notification is 4 – query results changed, the Results property is populated with the search results that were returned, **but only if** search monitoring is configured to *Include results in notifications when results change*.

Adding a Webhook

You can add a webhook in the Office365Mon web site when you Configure Office365Mon, or you can use the Subscription Management API to Add a Subscription Notification. A webhook is just another type of notification channel, like an email address or phone number that gets text messages. Conversely you can use the same approach for deleting a webhook – either use the Office365Mon web site or the Delete a Subscription Notification API.

Webhook Notification Data Model

The set of possible values for the data that is sent in a webhook is mostly described above in the JSON payload class definition. In addition to that, there are different values that can appear for the service status change notifications. The list of ServiceName and FeatureName attributes is not a part of the Office365Mon API; those values come directly from Microsoft via the Office 365 Service Communications API so any questions regarding those values should be obtained from them.

The values for the CurrentStatus and PreviousStatus are part of the Office365Mon API and are as follows:

- Investigating
- RestoringService
- VerifyingService
- ServiceRestored
- PostIncidentReviewPublished
- ServiceDegradation
- ServiceInterruption
- ExtendedRecovery
- Scheduled
- InProgress
- Completed
- Canceled
- ServiceOperational

You can test your webhook endpoint either using the Office365Mon web site on the Configure Office365Mon page, or using the Subscription Management API, as described next.

Processing a Webhook Notification

The code you write to process your webhook notification will vary of course based upon the language and tools you are using, as well as what type of workflow you actually build once you receive the data. What follows here though is one example of a .NET Web API endpoint that is used as a webhook. The point of this example is to illustrate how simple it can be to have the data POST'ed to your webhook endpoint and break it down into its various properties. Note that the "WebhookNotification" class in this code sample is documented above as the JSON payload.

```
[System.Web.Http.HttpPost]
[System.Web.Http.AllowAnonymous]
public HttpResponseMessage Office365MonWebhook(WebhookNotification message)
{
    HttpResponseMessage hrm = new HttpResponseMessage(HttpStatusCode.OK);

    //process the data here - your integration with your systems
    Debug.WriteLine(message.SubscriptionId.ToString());
    Debug.WriteLine(message.CompanyName);
    Debug.WriteLine(message.WebhookNotificationType.ToString());

    Debug.WriteLine(message.NotificationOutageInfo.OutageNotificationType.ToString());
    Debug.WriteLine(message.NotificationOutageInfo.Resource);
    Debug.WriteLine(message.NotificationServiceStatusInfo.ServiceName);
    Debug.WriteLine(message.NotificationServiceStatusInfo.FeatureName);
    Debug.WriteLine(message.NotificationServiceStatusInfo.CurrentStatus);
    Debug.WriteLine(message.NotificationServiceStatusInfo.PreviousStatus);

    //return an HTTP OK code
    return hrm;
}
```

Test a Webhook

Use this method to test a notification webhook.

REST Endpoint:

https://www.office365mon.com/api/developers/testwebhook

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
Address	String	The Url of the webhook endpoint. A webhook JSON payload will be posted to that endpoint with a WebhookNotificationType of 0, which means that it is a test POST. The endpoint must be anonymous.

Returns:

If the call works it will return content with the HTTP status code and reason that was returned from your webhook. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("Address",
"https://contoso.com/api/office365mon/webhook"));

//make the post to delete the notification
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "testwebhook");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//when there is an error you will get info about it in the results
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	//get the HTTPS status number and message from the API
	//NOTE: even though the fields are called ErrorNumber and
	//ErrorMessage that does NOT mean an error has occurred.  Those
	//are just the field names
	int statusNum = int.Parse(data["ErrorNumber"].ToString());
	string statusMsg = data["ErrorMessage"].ToString();
}
else
{
	//the operation failed
}
```

Office 365 Service Status Changes

The operations in this section all have to do with managing the Office 365 service status changes that are being monitored. This allows you to have a notification issued

whenever the feature of an Office 365 service enters a particular status. Services are monitored for entering any of the following statuses:

- Investigating
- RestoringService
- VerifyingService
- ServiceRestored
- PostIncidentReviewPublished
- ServiceDegradation
- ServiceInterruption
- ExtendedRecovery
- Scheduled
- InProgress
- Completed
- Canceled
- ServiceOperational

Get Service Status Values

Use this method to get a list of all of the possible service statuses that can be monitored. You can use this to allow your users to select the service statuses they want to monitor, or as values to send to the Subscription Management API to set the statuses that are being monitored for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getservicestatusvalues

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.

Returns:

The call returns a dictionary of <string, string> with the service status values and descriptions that can be monitored. If the call fails, the status code will be 400 – Bad Request.

Example:

```
//all REST endpoints start here
```

```
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getservicestatusvalues");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, string>>(results);

	//enumerate the dictionary; the Key is the numeric value for the
	//the service status; the Value is the description
}
else
{
	//the operation failed
}
```

Get the Monitored Service Statuses

Use this method to get the service statuses that are being monitored for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getservicestatus

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.

Returns:

The call returns a dictionary of <string, string> with the service status values and descriptions that are being monitored. If the call fails, the status code will be 400 – Bad Request.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete the notification
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getservicestatus");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, string>>(results);

	//enumerate the dictionary; the Key is the numeric value for the
	//the service status; the Value is the description
}
else
{
	//the operation failed
}
```

Set the Monitored Service Statuses

Use this method to set the service statuses that are being monitored for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/setservicestatus

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
StatusEvents	String	A semi-colon delimited list of service status values that should be monitored. To get the list of possible values use the getservicestatusvalues REST API.

Returns:

The call returns no data and a 200 status code if it works. If the call fails, the status code will be 400 – Bad Request or 409 – Conflict and details are provided in the response contents about the error.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//set this subscription to monitor for the Investigating and
//ServiceOperational statuses
string theStatus = "0;12;";

vals.Add(new KeyValuePair<string, string>("StatusEvents", theStatus));

//make the post to delete the notification
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"setservicestatus");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
}
else
```

```
{
        //when there is an error you will get info about it in the results
        string results = await hrm.Content.ReadAsStringAsync();

        //this is used to convert our JSON data into a dictionary of values
        JavaScriptSerializer jss = new JavaScriptSerializer();

        //set the maximum length value because some of the data,
        //especially health pings, can be quite long
        jss.MaxJsonLength = int.MaxValue;

        var data = jss.Deserialize<Dictionary<string, object>>(results);

        //get the error number and message from the API
        int errNum = int.Parse(data["ErrorNumber"].ToString());
        string errMsg = data["ErrorMessage"].ToString();
        //the HTTP response code is here: (int)hrm.StatusCode

        //this DIDN'T work - take the appropriate action
}
```

Resources

The operations in this section all have to do with managing the resources that are going to be monitored for the subscription.

Get Subscription Resources

Use this method to get all of the resources being monitored for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getresources

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being queried.

Returns:

An array of objects that each contain information about one resource: address and resource type code.

Name	Type	Description
Address	String	The address of the resource. It can be either a SharePoint Online url, or an Exchange Online email address.
TypeCode	Int	An int that represents the type of resource it is. The possible TypeCodes are:

Type Code	Code
SharePoint Online	0
Exchange Online	1
Skype Presence	4
Skype Instant Messaging	5

If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the resources
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getnotifications");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string resources = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(resources);

	if (data.ContainsKey("ResourceList"))
	{
		ArrayList resourceList = data["ResourceList"] as ArrayList;

		List<ResourceInfo> resourceInfos = new List<ResourceInfo>();

		foreach (var resource in resourceList)
```

```
                {
                        Dictionary<string, object> resourceInfo = resource as
                        Dictionary<string, object>;

                        string address = resourceInfo["Address"].ToString();
                        int typeCode = (int)resourceInfo["TypeCode"];
                }
        }
        else
                MessageBox.Show("No resources were found");
}
else
{
        //the operation failed
}
```

Add a Subscription Resource

Use this method to add a resource to a subscription. **NOTE: This method was updated January 2016 to account for unannounced breaking changes by the Microsoft Azure and ADAL teams.**

As a result of the Azure breaking changes, when you add a resource it cannot be immediately monitored because there is no access token associated with it. You can add an access token to the resource by the following methods:

- Navigating to the Office365Mon subscription configuration page and clicking the Update button next to the resource that was added.
- Using the method described in the ***Update the Access Token for a Subscription Resource*** section of this API documentation.
- Using the method described in the ***Copy an Access Token for a Subscription Resource*** section of this API documentation.
- Using the method described in the ***Copy All Access Tokens for Subscription Resources*** section of this API documentation.

Also, please note that you can add a resource and the access token for it in one step if you have an access code. Using the method described in the ***Update the Access Token for a Subscription Resource*** section of this API documentation will add the resource if it doesn't exist, and store the access token for it.

> ***NOTE:*** *There is not a method to add Skype Presence and Instant Messaging as resources to a subscription via this API. That's because of the way the Skype team has designed their service, each user account can have a different address with which to communicate. This can only be determined through a series of requests and follows, and thus is only available in the browser.*

REST Endpoint:

https://www.office365mon.com/api/developers/addresource

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
Address	String	The address of the resource to be monitored. It should be either a SharePoint Online url or an Exchange Online email address.
TypeCode	Int	An int that represents the type of address it is. The possible TypeCodes are:
		Type Code / **Code**
		SharePoint Online / 0
		Exchange Online / 1
		OneDrive / 10
		Power BI / 11
*Username	String	The username of an Azure Active Directory account that has rights to access the resource. **NOTE: This information is NEVER stored; it is just used one time to get an access token.**
*Password	String	The password for the Azure Active Directory account username. **NOTE: This information is NEVER stored; it is just used one time to get an access token.**
*TenantId	Guid	The tenant ID of the Azure Active Directory tenant to which the Username belongs.

*** Deprecated and no longer used because of changes made by Azure and ADAL teams.**

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
```

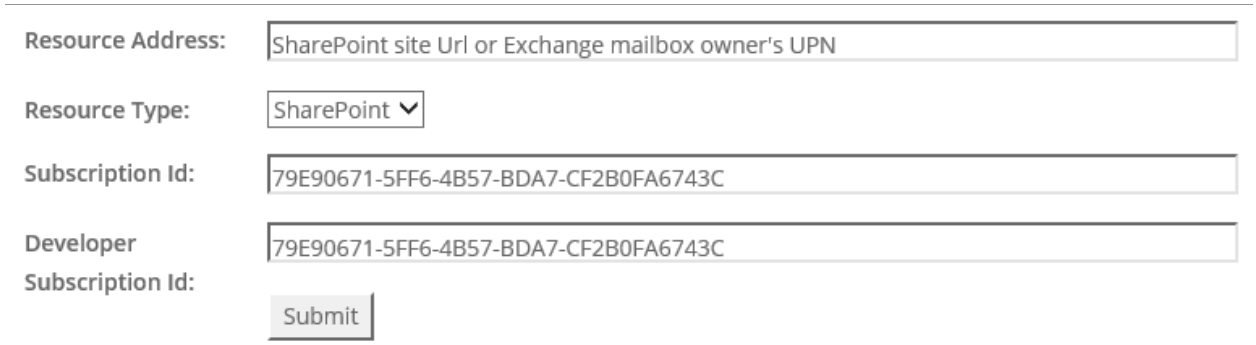
```
vals.Add(new KeyValuePair<string, string>("Address", "https://contoso-
public.sharepoint.com"));
vals.Add(new KeyValuePair<string, string>("TypeCode", "0"));

//make the post to add the SharePoint Online resource (because TypeCode = 0)
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "addresource");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Update the Access Token for a Subscription Resource

Use this method to update the access token for a subscription resource. Access tokens are issued by Azure Active Directory and expire approximately every 90 days. If you do not refresh the access tokens in your subscription, Office365Mon will no longer be able to monitor those resources for you when the token expires. **NOTE: This method was updated January 2016 to account for unannounced breaking changes by the Microsoft Azure and ADAL teams.**

Because of the breaking changes by the Azure team, this method now requires an access code and a redirect URI as method parameters. You obtain a code with a web application that requests a code from Azure AD. Your web application needs to redirect back to the Office365Mon.Com web site in order to retrieve the access code, which you can then use with this API.

The sample code for the Subscription Management API includes a zip file called MvcSample.zip. If you extract it and look in there you will see a sample ASP.NET MVC project that includes a controller called DevAccessCodeController and view called Index.cshtml. When you add this code to an ASP.NET web application that is secured by Azure AD you can navigate to the view and fill in a few simple fields to request an access code. When you first navigate to the view it appears like this:



The fields should contain this information:

- Resource Address: if you want to get an access code for a SharePoint site, enter the site Url. If you want an access code for an Exchange mailbox, enter the UPN of the user that has a mailbox. For a Web Site or REST API use its Url. Other resource types do not use the Resource Address and will ignore it if included.
- Resource Type: the type of resource for which you want a code – SharePoint Online, Exchange Online, Office 365 Service Info integration, a Web Site or REST API, OneDrive, Office 365 Service Info, Email Transport monitoring, Power BI, Threat Intelligence monitoring, Usage monitoring, or Microsoft Teams.
- Subscription Id: the Id of the Office365Mon subscription that is monitoring the resource.
- Developer Subscription Id: the Id of the Office365Mon subscription that has the Subscription Management API feature
- If you are updating the token for a Web Site or REST API you must also provide the ClientId, ClientSecret, and AppIdUri parameters as described below.

After you enter your values and click the Submit button, the controller redirects you to Azure where retrieves an access code. You are then redirected back to https://office365mon.com and the access code and other details you need to use in the API call to update the access token for the resource are displayed. Note that because of limitations in how Azure works, you can only be redirected back to Office365Mon.Com. The page details look like this:

Here are the values you need for the Office365Mon Subscription Management API to update the access token:

- SubscriptionId: 79E90671-5FF6-4B5A-BDA7-CF2B0FA6743C
- DevSubscriptionId: 79E90671-5FF6-4B5A-BDA7-CF2B0FA6743C
- Address: https://office365mon.sharepoint.com/sites/steve
- TypeCode: 0
- AccessCode: AAABAAAAiL9Kn2Z27UubvWFPbm0gLXuEnL1cyo9j015_JjYyrcbPKSHS0dmscUv6Y7LTDiIkusYT-A02nkELQMoKKSTUwslzU2kwDcW8Fdra0Y2jpv_raTRsdPr0HISDqfbQyUS80e-0jlwb4u8xrPHxll6AHV2dhqhL41047tN2XTOegAuyqixq7vr1BNav1wTcPQskLGB5XY7r4tXP9feoF976NWtVDM4GjKa
dT48IU9BuQWhY8ddVDOi6ge8wP9jXJvfQqC1pNL5Beg50aZoYX7XdOFr3PByDopzbeF3pXBJDL8eCavlQZNsynKIksc
wBKoUnu1ND8HSV3dAaWL2nLhtD19Blof2r7FhQDvO5YbUuBvS0KHpV7JtK_ms35dAUrhaOkra-hMO6kaNN6ICo0Yt5ANMhvUjtdeySmsYYV5HyjYAUge2hb_80bg7trTIXULxNMM5R1oIxADUkNd6GqjHB3ELYlGQ9Z

There is one other thing worth noting for use with this API – if the resource you are updating the token for is a SharePoint or OneDrive site, Exchange mailbox, Power BI, Email Transport monitor, or Microsoft Teams and the Monitored Resource has not been created in Office365Mon yet, it will be created and the access token stored for it. So, you can create your Monitored Resource and set the access token for it with a single call using the UpdateAccessToken method. Note that this does not apply to monitored Web Sites; for them you MUST create the Web Site in the Office365Mon subscription before you can use the UpdateAccessToken method.

> ***NOTE:*** *As described in the Add a Subscription Resource section, there is not a method to add Skype Presence and Instant Messaging as resources to a subscription via this API.*

REST Endpoint:

https://www.office365mon.com/api/developers/updateaccesstoken

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
Address	String	The address of the resource whose access token is being updated. It should be either a SharePoint Online or OneDrive for Business Url, or an Exchange Online email address. For all other TypeCode options, you can use a blank string.
TypeCode	Int	An int that represents the type of address it is. The possible TypeCodes are:

Type Code	Code
SharePoint Online	0
Exchange Online	1
Office 365 Admin Info	2
Monitored Web Site or API	3
Email Transport	4
Power BI	5
Threat Intelligence Monitoring	7
Usage Monitoring	8
Microsoft Teams	9

AccessCode	String	The access code you received after being redirected to https://www.office365mon.com/devaccesscode/processcode.
ClientId	String	The ClientId for an application used to monitor a Web Site or REST API that is secured with Azure Active Directory. This parameter is required if your TypeCode is 3 for a Monitored Web Site or API.
ClientSecret	String	The ClientSecret for an application used to monitor a Web Site or REST API that is secured with Azure Active Directory. This parameter is required if your TypeCode is 3 for a Monitored Web Site or API.
AppIdUri	String	The App ID URI for an application used to monitor a Web Site or REST API that is secured with Azure Active Directory. This parameter is required if your TypeCode is 3 for a Monitored Web Site or API.
*UPN	String	The UPN of the user that authenticated with Azure Active Directory to create an access code, if different from the user used to connect to the REST API. If the same user authenticated with Azure Active Directory and connected to the REST API, this parameter can be omitted.
*RedirectUri	String	The RedirectUri parameter used with Azure Active Directory when obtaining the access code. This parameter is only needed if you requested the token from your own web

		application and are using the Office365Mon Full White Label service; otherwise it **MUST** be omitted for the call will fail.
*Username	String	The username of an Azure Active Directory account that has rights to access the resource. **NOTE: This information is NEVER stored; it is just used one time to get an access token.**
*Password	String	The password for the Azure Active Directory account username. **NOTE: This information is NEVER stored; it is just used one time to get an access token.**
*TenantId	Guid	The tenant ID of the Azure Active Directory tenant to which the Username belongs.

* These items are only available to partners that are using the Full White Label service. If you try and pass these parameters and are **not** using the Full White Label service then the call will fail and/or the token will be invalid for monitoring.

*** Deprecated and no longer used because of changes made by Azure and ADAL teams.**

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict and you may find more information about the failure in the Reason Phrase.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("Address", "https://contoso-
public.sharepoint.com"));
vals.Add(new KeyValuePair<string, string>("TypeCode", "0"));
vals.Add(new KeyValuePair<string, string>("AccessCode", "reallyLongString"));
vals.Add(new KeyValuePair<string, string>("RedirectUri",
"https://yourWebAppUrlWhereTheTokenWasObtained"));

//make the post to add the SharePoint Online resource (because TypeCode = 0)
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"updateaccesstoken");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
```

```
}
else
{
        //the operation failed
        //you can get more detailed information about why
        //the call failed by looking at the Content property
        //of the HttpResponseMessage, like this:
        //string errMsg = hrm.Content.ReadAsStringAsync().Result;
}
```

Delete a Subscription Resource

Use this method to delete a subscription resource. This means that the resource will no longer be monitored by Office365Mon, it does nothing to the resource itself, i.e. it will not delete a SharePoint Online site or Exchange Online mailbox.

REST Endpoint:

https://www.office365mon.com/api/developers/deleteresource

Input Parameters:

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>DevSubscriptionId</td><td>Guid</td><td>The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.</td></tr>
<tr><td>SubscriptionId</td><td>Guid</td><td>The ID of the subscription being modified.</td></tr>
<tr><td>TypeCode</td><td>Int</td><td>An int that represents the type of address to delete. Please note that Skype Presence and Instant Messaging are BOTH deleted when you choose to delete Skype. The possible TypeCodes are:
<table>
<tr><th>Type Code</th><th>Code</th></tr>
<tr><td>SharePoint Online</td><td>0</td></tr>
<tr><td>Exchange Online</td><td>1</td></tr>
<tr><td>Skype</td><td>4</td></tr>
<tr><td>OneDrive</td><td>10</td></tr>
<tr><td>Power BI</td><td>11</td></tr>
</table></td></tr>
</table>

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("TypeCode", "0"));

//make the post to delete the SharePoint Online resource (because TypeCode = 0)
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "deleteresource");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Copy an Access Token for a Subscription Resource

Use this method to copy an access token from one subscription to another subscription. This can currently only be used to copy tokens that meet the following criteria:

- The resource is a SharePoint Online site.
- The resource the token is being copied to is part of the same tenant as the resource it's being copied from, i.e. the same SharePoint Online tenant.

The limitation of it only working with SharePoint Online is currently inherent in Office 365 and cannot be altered until they change their support in Exchange Online for delegate mailboxes. If you try and call this API and request to copy an Exchange Online access token the method will fail and return a 403 – Forbidden HTTP response.

REST Endpoint:

https://www.office365mon.com/api/developers/copytoken

Input Parameters:

Name	Type	Description

DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.
TargetSubscriptionId	Guid	The ID of the subscription to which the access token should be copied
TypeCode	Int	An int that represents the type of resource for which to copy the access token. The possible TypeCodes are:

Address Type	Code
SharePoint Online	0

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails, the status code will be 409 – Conflict and the ReasonPhrase will contain additional details about why it failed.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("TargetSubscriptionId", "12345678-1234-1234-
1234-123456789012"));
vals.Add(new KeyValuePair<string, string>("TypeCode", "0"));

//make the post to copy the SharePoint Online access token
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "copytoken");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed
}
```

Copy All Access Tokens for Subscription Resources

Use this method to copy an access token from one Office365Mon subscription to all other Office365Mon subscriptions for which the authenticated user is an admin. This can currently only be used to copy tokens that meet the following criteria:

- The resource is a SharePoint Online site.
- The resource the token is being copied to is part of the same tenant as the resource it's being copied from, i.e. the same SharePoint Online tenant.

The limitation of it only working with SharePoint Online is currently inherent in Office 365 and cannot be altered until they change their support in Exchange Online for delegate mailboxes. If you try and call this API and request to copy an Exchange Online access token the method will fail and return a 403 – Forbidden HTTP response.

If the token cannot be copied to one or more subscriptions, then HTTP response will be 409 – Conflict. The HTTP response content will contain a semi-colon delimited list of all of the subscription IDs that it could not copy the token to. In most cases this will happen if the subscription the token is being copied to does not have a resource added to it, i.e. it is not configured to monitor a SharePoint Online site. If you believe the copy should have completed successfully you can use the failed subscription IDs and query the resources for each one to see whether it has a SharePoint Online resource. The ***Get Subscription Resources*** section of this API documentation describes how to do that.

REST Endpoint:

https://www.office365mon.com/api/developers/copytokens

Input Parameters:

Name	Type	Description	
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.	
SubscriptionId	Guid	The ID of the subscription being modified.	
TypeCode	Int	An int that represents the type of resource for which to copy tokens. The possible TypeCodes are:	
		Type Code	**Code**
		SharePoint Online	0

Returns:

Nothing. If the call works, the Http Status Code in the response will be 200. If it fails copying to **any** of the subscriptions, the status code will be 409 – Conflict. For additional details on which subscriptions it failed to copy to look at the HTTP response content.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("TypeCode", "0"));

//make the post to copy the SharePoint Online access token from the subscription
//passed in the SubscriptionId parameter to all other subscriptions for which
//the currently authenticated user is an administrator
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "copytokens");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the operation worked
}
else
{
	//the operation failed – here's how you can find out more about the failure:
	string failedSubCopies = hrm.Content.ReadAsStringAsync().Result;
	string[] failedSubs = failedSubCopies.Split(";".ToCharArray(),
		StringSplitOptions.RemoveEmptyEntries);

}
```

Issue a Health Probe for a Resource

Use this method to manually issue a health probe against a resource and check the Http status code it returns.

REST Endpoint:

https://www.office365mon.com/api/developers/issuehealthprobe

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being modified.

TypeCode	Int	An int that represents the type of address to issue the probe against. The possible TypeCodes are:

Type Code	Code
SharePoint Online	0
Exchange Online	1
Skype Presence	4
Skype IM	5
OneDrive	10
Power BI	11
Teams	15
Teams Channel	16

Returns:

Name	Type	Description
StatusCode	Int	The Http status code that was returned from the health probe.
StatusReason	String	The Http reason code returned in response to the health probe.

If the call fails, the status code will be 409 – Conflict. For Skype IM, a successful probe is a StatusCode of 201; for the others it is 200.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("TypeCode", "0"));

//make the post to test the SharePoint Online resource (since TypeCode = 0)
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"issuehealthprobe");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();
```

```
        //set the maximum length value because some of the data,
        //especially health pings, can be quite long
        jss.MaxJsonLength = int.MaxValue;

        var data = jss.Deserialize<Dictionary<string, object>>(results);

        int statusCode = int.Parse(data["StatusCode"].ToString());
        string statusReason = data["StatusReason"].ToString();
}
else
{
        //the operation failed
}
```

Health Score Notifications

The operations in this section are used to configure the health score notifications feature. This feature is used with SharePoint Online and Skype for Business, and measures two metrics that Microsoft provides when you make a request to the service – the health score and the request duration. These values are captured every time a monitoring request is made to one of these services, and can optionally send out notifications if they exceed values configured by the user, or using these REST APIs.

Get Health Score Notification Configuration

Use this method to get the configuration of health score notifications for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/gethealthscoreconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
ResourceType	Int	The type of resource for which you are getting the configuration. Currently the only supported value

		is 0, but this is subject to change in subsequent releases.

Returns:

Name	Type	Description
HealthNotifications	Bool	Boolean indicating whether health score notifications are enabled.
HealthScore	Int	The Health Score value above which a notification will be sent.
RequestDuration	Int	The RequestDuration value, in milliseconds, above which a notification will be sent.
NetworkRequestDuration	Int	The NetworkRequestDuration value, in milliseconds, above which a notification will be sent.
ResourceType	Int	The type of resource for this health score notification configuration. Currently the only supported value is 0, but this is subject to change in subsequent releases.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//currently only 0 is supported; that is subject to change in future releases
vals.Add(new KeyValuePair<string, string>("ResourceType", "0"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"gethealthscoreconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
   string info = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
```

```
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);

    bool notificationsEnabled = bool.Parse(data["HealthNotifications"].ToString());
    int healthScore = data["HealthScore"].ToString();
    int requestDuration = data["RequestDuration"].ToString();
    int networkRequestDuration = data["NetworkRequestDuration"].ToString();

    //for illustration purposes only at this time
    int ResourceType = int.Parse(data["ResourceType"].ToString());
}
else
{
    //the operation failed
}
```

Update Health Score Notification Configuration

Use this method to add or update the health score notification configuration, or to turn it off, for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/configurehealthscores

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
HealthNotifications	Bool	Boolean indicating whether health score notifications are enabled.
HealthScore	Int	The Health Score value above which a notification will be sent.
RequestDuration	Int	The RequestDuration value, in milliseconds, above which a notification will be sent.
NetworkRequestDuration	Int	The NetworkRequestDuration value, in milliseconds, above which a notification will be sent.
ResourceType	Int	The type of resource for which you are getting the configuration. Currently the only supported value

		is 0, but this is subject to change in subsequent releases.

Returns:

The call returns nothing if it was successful; it returns a 409 – Conflict status if there was a problem, and when appropriate additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("HealthNotifications", "true"));
vals.Add(new KeyValuePair<string, string>("HealthScore", "8"));
vals.Add(new KeyValuePair<string, string>("RequestDuration", "3500"));
vals.Add(new KeyValuePair<string, string>("NetworkRequestDuration", "5000"));

//currently only 0 is supported; that is subject to change in future releases
vals.Add(new KeyValuePair<string, string>("ResourceType", "0"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"configurehealthscores");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
{
   //the operation failed

   //when there is an error you *may* get additional info in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;
```

```
    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Web Sites

The operations in this section all have to do with managing a monitored Web Site or REST API for the subscription.

Get Monitored Web Sites

Use this method to get all of the Web Sites and REST APIs being monitored for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getmonitoredsites

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being queried.

Returns:

An array of Dictionary objects that each contain information about a monitored Web Site or REST API.

Name	Type	Description
SiteAddress	String	The Url of the Web Site or REST API.
ClientId	String	The ClientId for the Azure Active Directory application used in monitoring the Web Site or REST API.
ClientSecret	String	The ClientSecret for the Azure Active Directory application used in monitoring the Web Site or REST API.

AppIdUri	String	The App ID URI for the Azure Active Directory application used in monitoring the Web Site or REST API.
IsSecure	Bool	A Boolean indicating whether the monitored Web Site or REST API is secured with Azure Active Directory.

If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the resources
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getmonitoredsites");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data can be long
    jss.MaxJsonLength = int.MaxValue;

    Array data = jss.DeserializeObject(results) as Array;

    if (data.Length != 0)
    {
        foreach (Dictionary<string, object> item in data)
        {
            //get values out the dictionary like so:
            //(string)item["SiteAddress"]
    }
    else
        MessageBox.Show("No monitored sites were found");
}
else
{
        //the operation failed
}
```

Add Monitored Web Site

Use this method to add a monitored Web Sites or REST API. Note that when you call this method, if successful it also creates a Monitored Resource in the subscription for it as well.

REST Endpoint:

https://www.office365mon.com/api/developers/addmonitoredsite

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being queried.
SiteAddress	String	The Url of the Web Site or REST API. **IMPORTANT:** You cannot add a Url that includes a query string!
ClientId	String	The ClientId for the Azure Active Directory application used in monitoring the Web Site or REST API.
ClientSecret	String	The ClientSecret for the Azure Active Directory application used in monitoring the Web Site or REST API.
AppIdUri	String	The App ID URI for the Azure Active Directory application used in monitoring the Web Site or REST API.
IsSecure	Bool	A Boolean indicating whether the monitored Web Site or REST API is secured with Azure Active Directory.

Returns:

A status code of 200 OK if it succeeds. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();
```

```
//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SiteAddress", "https://web.steve.com/login"));
vals.Add(new KeyValuePair<string, string>("ClientId", "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx"));
vals.Add(new KeyValuePair<string, string>("ClientSecret", "Avc+qq5bmiMZBxmPwLfjLBM="));
vals.Add(new KeyValuePair<string, string>("AppIdUri", "http://localhost/steveweb"));
vals.Add(new KeyValuePair<string, string>("IsSecure", "true"));

//make the post to add the notification
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"addmonitoredsite");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   MessageBox.Show("Monitored site was added.");
}
else
{
       //the operation failed
}
```

Delete Monitored Web Site

Use this method to delete a monitored Web Sites or REST API. Note that when you call this method, if successful it also deletes the Monitored Resource associated with it in the subscription for it as well.

REST Endpoint:

https://www.office365mon.com/api/developers/deletemonitoredsite

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being queried.
SiteAddress	String	The Url of the Web Site or REST API.

Returns:

A status code of 200 OK if it succeeds. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SiteAddress", "https://web.steve.com/login"));

//make the post to add the notification
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletemonitoredsite");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   MessageBox.Show("Monitored site was deleted.");
}
else
{
       //the operation failed
}
```

Office 365 Search Monitor

The operations in this section all have to do with managing the Office 356 Search Monitor feature.

Get the Search Monitoring Configuration

Use this method to get the configuration for monitoring the Search service.

REST Endpoint:

https://www.office365mon.com/api/developers/getsearchmonconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.

SubscriptionId	Guid	The ID of the subscription for which you want to retrieve the search configuration.

Returns:

Name	Type	Description
KQL	String	The KQL used to issue queries that check for latency, changed results and no results.
SendNotifications	Bool	A flag that determines whether notifications are sent out when the query latency exceeds the value of the configuration's Duration.
Duration	Int	The number of seconds a query can take without sending notifications. If a query takes longer than this number to return results, then notifications are sent.
NotifyOnChange	Bool	A flag that determines whether notifications are sent if the KQL query returns different results than the last time the query was executed.
IncludeResultsInNotifications	Bool	A flag that determines whether the search results themselves are returned in notifications when the results change.
NotifyIfNoResults	Bool	A flag that determines whether notifications are sent if the KQL query returns no results.
MonitorCrawl	Bool	A flag that determines whether crawl times should be monitored
CrawlSiteUrl	String	The Url of the site being crawled if monitoring crawl latency.

If the call fails, the status code will be 409 – Conflict and you can look at the Content of the HttpResponseMessage as a string to get additional details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to create the subscription
```

```
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getsearchmonconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	if (data["KQL"] != null)
		string kql = data["KQL"].ToString();

	bool sendNotifications = bool.Parse(data["SendNotifications"].ToString());
	int duration = data["Duration"].ToString();
	bool sendOnChange = bool.Parse(data["NotifyOnChange"].ToString());
	bool includeResults =
		bool.Parse(data["IncludeResultsInNotifications"].ToString());
	bool sendOnNoResults = bool.Parse(data["NotifyIfNoResults"].ToString());
	bool monitorCrawl = bool.Parse(data["MonitorCrawl"].ToString());

	  if (data["CrawlSiteUrl"] != null)
		string crawlSiteUrl = data["CrawlSiteUrl"].ToString();
}
else
{
	//the operation failed
}
```

Update the Search Monitoring Configuration

Use this method to update the configuration for monitoring the Search service.

REST Endpoint:

https://www.office365mon.com/api/developers/updatesearchmonconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.

SubscriptionId	Guid	The ID of the subscription for which you want to retrieve the search configuration.
KQL	String	The KQL used to issue queries that check for latency, changed results and no results.
SendNotifications	Bool	A flag that determines whether notifications are sent out when the query latency exceeds the value of the configuration's Duration.
Duration	Int	The number of seconds a query can take without sending notifications. If a query takes longer than this number to return results, then notifications are sent.
NotifyOnChange	Bool	A flag that determines whether notifications are sent if the KQL query returns different results than the last time the query was executed.
IncludeResultsInNotifications	Bool	A flag that determines whether the search results themselves are returned in notifications when the results change.
NotifyIfNoResults	Bool	A flag that determines whether notifications are sent if the KQL query returns no results.
MonitorCrawl	Bool	A flag that determines whether crawl times should be monitored

Returns:

If the call works, it returns an HTTP Status Code of 200 and no data. If the call fails, the status code will be 409 – Conflict and you can look at the Content of the HttpResponseMessage as a string to get additional details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("KQL", "querytext='sharepoint'"));
vals.Add(new KeyValuePair<string, string>("SendNotifications", "true"));
vals.Add(new KeyValuePair<string, string>("Duration", "8"));
vals.Add(new KeyValuePair<string, string>("NotifyOnChange", "true"));
vals.Add(new KeyValuePair<string, string>("IncludeResultsInNotifications", "true"));
vals.Add(new KeyValuePair<string, string>("NotifyIfNoResults", "true"));
vals.Add(new KeyValuePair<string, string>("MonitorCrawl", "true"));
```

```
//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"updatesearchmonconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
else
{
	//the operation failed
}
```

Delete the Search Monitoring Configuration

Use this method to delete the configuration for monitoring the Search service.

REST Endpoint:

https://www.office365mon.com/api/developers/deletesearchmonconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription for which you want to retrieve the search configuration.

Returns:

If the call works, it returns an HTTP Status Code of 200 and no data. If the call fails, the status code will be 409 – Conflict and you can look at the Content of the HttpResponseMessage as a string to get additional details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();
```

```
//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletesearchmonconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
else
{
	//the operation failed
}
```

Validate the Search Monitoring SharePoint App Installation

In order for Office365Mon to monitor crawls as well as do document probes, you must install the Office365Mon Search Monitor app in the site that is being monitored or doing document probes. You can use this method to determine whether the app has been installed in a monitored SharePoint or OneDrive site.

REST Endpoint:

https://www.office365mon.com/api/developers/validatesearchmon

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription for which you want to retrieve the search configuration.
ResourceType	Int	OPTIONAL: if you don't supply a parameter, it will check the monitored SharePoint site. Otherwise, use 0 to check the monitored SharePoint site, or 10 to check the monitored OneDrive site.

Returns:

If the call works, it returns an HTTP Status Code of 200 and you can look at the Content of the HttpResponseMessage to get a Boolean value indicating whether the app has been installed. If the call fails, the status code will be 409 – Conflict and you can look at the Content of the HttpResponseMessage as a string to get additional details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"validatesearchmon");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string result = await hrm.Content.ReadAsStringAsync();
	bool siteValid = bool.Parse(result);
else
{
	//the operation failed
}
```

Distributed Probes and Diagnostics

The operations in this section all have to do with managing access codes for the Distributed Probes and Diagnostics feature.

Get Access Code

This method creates a new access code and returns it.

REST Endpoint:

https://www.office365mon.com/api/developers/getdpdaccesscode

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription to which you are adding the access code.

Returns:

Name	Type	Description
	String	The content of the HttpResponseMessage contains the new access code.

If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to add the access code
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getdpdaccesscode");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//the new code is in the content of the response message
	string code = await hrm.Content.ReadAsStringAsync();
}
else
{
	//the operation failed
}
```

Delete Access Code

This method deletes one individual access code.

REST Endpoint:

https://www.office365mon.com/api/developers/deletedpdaccesscode

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription from which you are deleting the access code.
AccessCode	Guid	The access code you wish to delete.

Returns:

Nothing. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("AccessCode", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete the access code
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletedpdaccesscode");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
}
else
{
	//the operation failed
}
```

Delete All Access Codes

This method deletes all Distributed Probes and Diagnostics access codes.

REST Endpoint:

https://www.office365mon.com/api/developers/deletealldpdaccesscodes

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription from which you are deleting the access code.

Returns:

Nothing. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletealldpdaccesscodes");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
}
else
{
	//the operation failed
}
```

Get Remote Agent Monitoring Configuration

This method retrieves the configuration used to monitor remote agents running the Distributed Probes and Diagnostics service.

REST Endpoint:

https://www.office365mon.com/api/developers/getdpdmonitorconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription from which you are retrieving the configuration.

Returns:

Returns information about whether monitoring for remote agents is enabled to send alerts, and what the maximum offline time, in minutes, is before an alert is sent.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getdpdmonitorconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    string info = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(info);

    bool shouldAlert = bool.Parse(data["ShouldAlert"].ToString());
    int maxOfflineTime = int.Parse(data["MaxOfflineTime"].ToString());
```

```
}
else
{
        //the operation failed
}
```

Update Remote Agent Monitoring Configuration

This method updates the configuration used to monitor remote agents running the Distributed Probes and Diagnostics service.

REST Endpoint:

https://www.office365mon.com/api/developers/updatedpdmonitorconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription for which you are updating the configuration.
ShouldAlert	Bool	Set to True in order to send alerts when an agent has not issued a health probe for more than MaxOfflineTime, as defined below.
MaxOfflineTime	Int	The amount of time, in minutes, that an agent must fail to report a health probe to the cloud service before it is considered offline. **The MaxOfflineTime value must be at least 10 and no more than 1440.**

Returns:

If it works it returns an HTTP status code 200 response with no data. If the MaxOfflineTime parameter is not within the values described above, it returns a 409 – Conflict HTTP status code along with additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();
```

```
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("ShouldAlert", "true"));
vals.Add(new KeyValuePair<string, string>("MaxOfflineTime", "120"));


//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"updatedpdmonitorconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //the operation worked
}
else
{
   //the operation failed

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
}
```

Get Geo Aware Alerts Configuration

This method gets the configuration for the Geo Aware Alerts feature.

REST Endpoint:

https://www.office365mon.com/api/developers/getgeoawarealertconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.

SubscriptionId	Guid	The ID of the subscription for which you are updating the configuration.

Returns:

If the SubscriptionId does not have the Distributed Probes and Diagnostics feature the HTTP return status will be 401 – Unauthorized. Otherwise, it will return the following data:

Name	Type	Description
DoNotifications	Bool	Boolean indicating whether Geo Aware Alerts are enabled.
Distance	Int	The distance that one or more of the customer's Distributed Probe and Diagnostics agents must be within of an outage in order to trigger an alert. The distance is either Miles or Kilometers, depending on the value of the Units property.
Units	Int	The units the Distance property is measured in. Miles = 0; Kilometers = 1.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));


//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getgeoawarealertconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //the operation worked
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data can be long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);
```

```
    bool doNotifications = bool.Parse(data["DoNotifications"].ToString());
    int distance = int.Parse(data["Distance"].ToString());
    int units = int.Parse(data["Units"].ToString());
}
else
{
    //the operation failed

    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Update Geo Aware Alerts Configuration

This method updates the Geo Aware Alerts feature configuration.

REST Endpoint:

https://www.office365mon.com/api/developers/configuregeoawarealertconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription for which you are updating the configuration.
DoNotifications	Bool	Set to True in order to send alerts when an outage is reported by another Office365Mon customer within the distance described by the Distance and Units properties, as defined below.
Distance	Int	The distance, in Miles or Kilometers as defined by the Units property, in which an outage by another Office365Mon customer will trigger a Geo Aware Alert.

		The Distance value must be greater than 0.
Units	Int	The units the Distance should be measured in when determining whether to trigger a Geo Aware Alert. Miles = 0; Kilometers = 1.

Returns:

If it works it returns an HTTP status code 200 response with no data. If the Distance parameter is not within the values described above, it returns a 409 – Conflict HTTP status code along with additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>(" DoNotifications", "true"));
vals.Add(new KeyValuePair<string, string>("Distance", "100"));
vals.Add(new KeyValuePair<string, string>("Units", "0"));


//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"configuregeoawarealertconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //the operation worked
}
else
{
   //the operation failed

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);
```

```
    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Get Document Probe Configuration

This method gets the configuration for monitoring document uploads and downloads.

REST Endpoint:

https://www.office365mon.com/api/developers/getdocprobeconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription for which you are updating the configuration.

Returns:

If it works it returns a list of ResourceType items that are configured to monitor document uploads and downloads. The ResourceType value corresponds to a MonitoredResourceType. The types of resources that support document probes are SharePoint (0) and OneDrive (10). If it fails, it returns a 409 – Conflict response.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));


//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getdocprobeconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
```

```
{
    //the operation worked

    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    //DocProbeConfig is a class the REST API sample application; it contains
    //just two fields: SubscriptionId and ResourceType
    var data = jss.Deserialize<List<DocProbeConfig>>(results);
}
else
{
    //the operation failed
}
```

Update Document Probe Configuration

This method is used to delete document probing for a resource type (like SharePoint and OneDrive). In the future, it may be used to modify other configuration properties.

> ***IMPORTANT:*** *In order for Office365Mon to do document probes, you must install the Office365Mon Search Monitor app in the SharePoint and/or OneDrive site that is being monitored. Visit the site for more information on how to do that. Also, you can use the ValidateSearchMon REST API to determine if it has been installed in a monitored site.*

REST Endpoint:

https://www.office365mon.com/api/developers/updatedocprobeconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription for which you are updating the configuration.
ResourceType	Int	The resource type for which the document probe configuration is being updated. 0 = SharePoint, 10 = OneDrive.
IsEnabled	Bool	Set to true to enable document probes for the resource type; set to false to turn off document probes for the resource type.

Returns:

If it works it returns an HTTP StatusCode of OK. If it fails, it returns a 409 – Conflict response.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("ResourceType", "0"));
vals.Add(new KeyValuePair<string, string>("IsEnabled", "true"));


//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"updatedocprobeconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //the operation worked

    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    //DocProbeConfig is a class the REST API sample application; it contains
    //just two fields: SubscriptionId and ResourceType
    var data = jss.Deserialize<List<DocProbeConfig>>(results);
}
else
{
    //the operation failed
}
```

Exchange Online Version Changes

The operations in this section all have to do with checking, enabling and disabling monitoring Exchange Online for version changes. Note that enabling and disabling Exchange version monitoring ***won't do anything*** unless the subscription has an access token for the Office365Mon app that monitors both email and versions. In the Configure Office365Mon page on the Office365Mon web site, that is done by checking the box that says "Also monitor Exchange Online version changes." If you are getting an access token yourself and passing it to the UpdateAccessToken method, then the client ID needed is e32a9177-2fca-4e82-8c27-12af8c6c3c5b. This is also covered in our MVC Sample application that is included with this download.

Is Version Monitoring Enabled

Use this method to see if monitoring for version changes in Exchange Online has been enabled.

REST Endpoint:

https://www.office365mon.com/api/developers/isexoversionmonitoring

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns a string if successful. If version monitoring is enabled the return value is "true"; if not it is "false".

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();
```

```
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"isexoversionmonitoring");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
	string results = await hrm.Content.ReadAsStringAsync();
	bool isMonitoring = bool.Parse(results);
}
else
{
	//the operation failed
}
```

Enable Version Monitor

Use this method to enable monitoring for version changes in Exchange Online.

REST Endpoint:

https://www.office365mon.com/api/developers/enableexoversionmonitor

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are enabling Exchange version monitoring.

Returns:

Nothing. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
```

```
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"enableexoversionmonitor");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
}
else
{
	//the operation failed
}
```

Disable Version Monitor

Use this method to disable monitoring for version changes in Exchange Online.

REST Endpoint:

https://www.office365mon.com/api/developers/disableexoversionmonitor

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are disabling Exchange version monitoring.

Returns:

Nothing. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"disableexoversionmonitor");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
}
else
{
	//the operation failed
}
```

Dashboard Reports

The operations in this section all have to do with getting, adding and deleting Dashboard Report keys. Report keys are used with the Dashboard Reports feature, which allows you to display reports from the Office365 Basic and Advanced Reports galleries in any web site. To get more information on configuring and using the Dashboard Reports feature please see our configuration guide at https://www.office365mon.com/Office365Mon_DashboardReports_Configuration.pdf or go to our site to set it up at https://www.office365mon.com/Configure/Dashboard.

Get Dashboard Report Keys

Use this method to get all of the Dashboard Report keys for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getdashboardreportkeys

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns an array of Dictionary<string, object> items that describe the Dashboard Report keys collection.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getdashboardreportkeys");

//look at the results
if (hrm.IsSuccessStatusCode)
{
       //it worked
       string keys = await hrm.Content.ReadAsStringAsync();

       //this is used to convert our JSON data into a dictionary of values
       JavaScriptSerializer jss = new JavaScriptSerializer();

       //set the maximum length value because some of the data,
       //especially health pings, can be quite long
       jss.MaxJsonLength = int.MaxValue;

       Array data = jss.DeserializeObject(keys) as Array;

       foreach (var keyData in data)
       {
          Dictionary<string, object> keyItem = keyData as Dictionary<string, object>;
```

```
        }
}
else
{
        //the operation failed
}
```

Add Dashboard Report Key

Use this method to add a new Dashboard Report Key.

REST Endpoint:

https://www.office365mon.com/api/developers/adddashboardreportkey

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are enabling Exchange version monitoring.
Notes	String	Any notes you want associated with the key. The notes show up as a description for the key when viewed in the Office365Mon web site.

Returns:

The new Dashboard Report key. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("Notes", "this key is for admins only"));
```

```
//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"adddashboardreportkey");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked – the new key is in the response message content
	string newKey = await hrm.Content.ReadAsStringAsync();
}
else
{
	//the operation failed
}
```

Delete a Dashboard Report Key

Use this method to delete one Dashboard Report key.

REST Endpoint:

https://www.office365mon.com/api/developers/deletedashboardreportkey

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are disabling Exchange version monitoring.
ReportKey	String	The Dashboard Report key that should be deleted.

Returns:

Nothing. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();
```

```
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("ReportKey", "
AMtzf4vSi1hMR8bqwAR+6wHu1buO0Excb5T1KjP8RgYX4XCLE6apiDjOjgBBrHA=="));


//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletedashboardreportkey");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
}
else
{
	//the operation failed
}
```

Delete All Dashboard Report Keys

Use this method to delete all Dashboard Report keys for a subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/deletealldashboardreportkeys

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are disabling Exchange version monitoring.
ReportKey	String	The Dashboard Report key that should be deleted.

Returns:

Nothing. If the call fails, the status code will be 409 – Conflict.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));


//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletealldashboardreportkeys");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
}
else
{
	//the operation failed
}
```

Email Transport Monitoring

The operations in this section are used to configure monitoring the email transport for an Office 365 tenant. They use properties to refer to Inbound monitoring – messages that are coming into Office 365 – and Outbound monitoring – messages that are going out from Office 365.

Get Email Transport Monitoring Configuration

Use this method to get the current email transport monitoring configuration for a subscription. Note that it's possible a subscription doesn't have any configuration created yet, in which case no data will be returned.

REST Endpoint:

https://www.office365mon.com/api/developers/getemailtransportconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns an array of Dictionary<string, object> items that describe the Email Transport monitoring configuration.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getemailtransportconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //it worked
   string info = await hrm.Content.ReadAsStringAsync();

    if (!string.IsNullOrEmpty(info))
    {
        //this is used to convert our JSON data into a dictionary of values
        JavaScriptSerializer jss = new JavaScriptSerializer();

        //set the maximum length value because some of the data,
        //especially health pings, can be quite long
        jss.MaxJsonLength = int.MaxValue;

        var data = jss.Deserialize<Dictionary<string, object>>(info);

        bool inboundAlert = bool.Parse(data["InboundShouldAlert"].ToString());
        string inboundMaxTime = data["InboundMaxOfflineTime"].ToString();
        string inboundRecipient = data["InboundRecipient"].ToString();
        bool outboundAlert = bool.Parse(data["OutboundShouldAlert"].ToString());
        string outboundMaxTime = data["OutboundMaxOfflineTime"].ToString();
```

```
    }
    else
        MessageBox.Show("This subscription has not configured Email Transport
            monitoring yet.");
}
else
{
    //the operation failed
}
```

Update Email Transport Monitoring Configuration

Use this method to add or update the Email Transport monitoring configuration.

REST Endpoint:

https://www.office365mon.com/api/developers/updateemailtransportconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are enabling Exchange version monitoring.
InboundShouldAlert	Bool	Whether inbound email transport should be monitored. If you don't ask for notifications, it won't be monitored.
InboundMaxOfflineTime	Int	The number of minutes after which if an inbound message isn't delivered, a notification will be sent. This must be no less than 5, and no more than 1440.
OutboundShouldAlert	Bool	Whether outbound email transport should be monitored. If you don't ask for notifications, it won't be monitored.
OutboundMaxOfflineTime	Int	The number of minutes after which if an outbound message isn't delivered, a notification will be sent. This must be no less than 5, and no more than 1440.

Returns:

Nothing if it works. If the call fails, the status code will be 409 – Conflict along with additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("InboundShouldAlert", "true"));
vals.Add(new KeyValuePair<string, string>("InboundMaxOfflineTime", "15"));
vals.Add(new KeyValuePair<string, string>("OutboundShouldAlert", "true"));
vals.Add(new KeyValuePair<string, string>("OutboundMaxOfflineTime", "20"));

//make the post to delete all access codes
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"updateemailtransportconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	//it worked
}
else
{
	//the operation failed

	//when there is an error you will get info about it in the results
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	//get the error number and message from the API
	int errNum = int.Parse(data["ErrorNumber"].ToString());
	string errMsg = data["ErrorMessage"].ToString();
}
```

List Monitoring

The operations in this section are used to configure monitoring large lists in SharePoint Online. Large lists have traditionally been a problem for organizations using SharePoint. A large list can become very difficult to work with and slow down all sorts of things that are dependent up it. Office365Mon allows you to monitor large lists in SharePoint Online. You can set thresholds to be alerted when it takes too long to view a list. You can also get notified after you lists reach a certain size. One of the top user complaints about SharePoint is around performance, and using List Monitoring allows you to find out about it ahead of time. Then you can use one of the many methods for mitigating large list performance and get in front the problem before it is out of hands with your users.

Monitored List Data Format

The API provides two methods that are used to return list data – lists that are in a monitored SharePoint Online site, and lists that are configured to be monitored. The sample application includes a simple class definition that illustrates how to work with the data that is returned in these two cases:

```
public class MonitoredListInfo
{
	public string Title { get; set; }
	public Guid ListId { get; set; }
	public Guid ViewId { get; set; }
	public string ServerRelativeUrl { get; set; }

	public MonitoredListInfo(string Title, Guid ListId, Guid ViewId, string
ServerRelativeUrl)
	{
		this.Title = Title;
		this.ListId = ListId;
		this.ViewId = ViewId;
		this.ServerRelativeUrl = ServerRelativeUrl;
	}

	public MonitoredListInfo(string Title, Guid ListId, string ServerRelativeUrl)
	{
		this.Title = Title;
		this.ListId = ListId;
		this.ServerRelativeUrl = ServerRelativeUrl;
	}
}
```

Get List Monitoring Configuration

Use this method to get the current list monitoring configuration for a subscription. Note that it's possible a subscription doesn't have any configuration created yet, in which case no data will be returned.

REST Endpoint:

https://www.office365mon.com/api/developers/getlistmonconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns an array of Dictionary<string, object> items that describe the list monitoring configuration.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getlistmonconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
   string info = await hrm.Content.ReadAsStringAsync();

   if (!string.IsNullOrEmpty(info))
```

```
    {
        //this is used to convert our JSON data into a dictionary of values
        JavaScriptSerializer jss = new JavaScriptSerializer();

        //set the maximum length value because some of the data,
        //especially health pings, can be quite long
        jss.MaxJsonLength = int.MaxValue;

        var data = jss.Deserialize<Dictionary<string, object>>(info);

        string maxRenderTimeTxt = data["MaxRenderTime"].ToString();
        string maxItemsTxt.Text = data["MaxListSize"].ToString();
    }
    else
        MessageBox.Show("This subscription has not configured list monitoring yet.");
}
else
{
    //the operation failed
}
```

Set List Monitoring Configuration

Use this method to set (i.e. create or update) the current list monitoring configuration for a subscription. **NOTE:** The subscription **must** have monitoring configured for a SharePoint Online site **first**, before you can configure list monitoring. The lists that are monitored come from the same SharePoint Online site that is being monitored.

REST Endpoint:

https://www.office365mon.com/api/developers/updatelistmonconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
MaxRenderTime	Int	The maximum time, in seconds, for the default view of a list to be rendered. Anything longer will send a notification. This must be no less than 1 and no more than 300.
MaxListSize	Int	The maximum number of records allowed in a list; when a list grows bigger than that a notification will be sent. This must be no less than 1000 and no more than 500,000.

Returns:

The call returns nothing if it's successful; if it fails it returns additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("MaxRenderTime", "10"));
vals.Add(new KeyValuePair<string, string>("MaxListSize", "4800"));

//make the post to add or update list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"updatelistmonconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
{
   //the operation failed

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
}
```

Stop Monitoring Lists

Use this method to stop monitoring all lists for a subscription, including deleting the list monitoring configuration.

REST Endpoint:

https://www.office365mon.com/api/developers/stoplistmonitoring

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are stopping list monitoring.

Returns:

The call returns nothing if it's successful; if it fails it returns additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to stop monitoring lists
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"stoplistmonitoring");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //it worked
}
else
{
    //the operation failed
```

```
    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Get Monitored Lists

Use this method to get all of the lists that are being monitored for this subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getmonitoredlists

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are getting the list of monitored lists.

Returns:

The call returns a Dictionary<string, object>, that is a list of all the monitored sites; if it fails it returns additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
```

```
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list of sites
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getmonitoredlists");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //it worked – this uses the MonitoredListInfo class defined in the SDK

    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data can be long
    jss.MaxJsonLength = int.MaxValue;

    Array data = jss.DeserializeObject(results) as Array;

    if (data.Length != 0)
    {
        List<MonitoredListInfo> lists = new List<MonitoredListInfo>();
        foreach (Dictionary<string, object> item in data)
        {
            lists.Add(new MonitoredListInfo((string)item["Title"],
Guid.Parse((string)item["ListId"]), (string)item["ServerRelativeUrl"]));
        }
    }
}
else
{
    //the operation failed

    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Get Site Lists

Use this method to get all of the lists in the SharePoint Online site that is being monitored for this subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getsitelists

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are getting the list of monitored lists.

Returns:

The call returns a Dictionary<string, object>, that is a list of all the monitored sites; if it fails it returns additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list of sites
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + " getsitelists");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //it worked – this uses the MonitoredListInfo class defined in the SDK

    string results = await hrm.Content.ReadAsStringAsync();
```

```
    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data can be long
    jss.MaxJsonLength = int.MaxValue;

    Array data = jss.DeserializeObject(results) as Array;

    if (data.Length != 0)
    {
        List<MonitoredListInfo> lists = new List<MonitoredListInfo>();
        foreach (Dictionary<string, object> item in data)
        {
            lists.Add(new MonitoredListInfo((string)item["Title"],
Guid.Parse((string)item["ListId"]), Guid.Parse((string)item["ViewId"]),
(string)item["ServerRelativeUrl"]));
        }
    }
}
else
{
    //the operation failed

    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Add a Monitored List

Use this method to add a new list to be monitored. Note that you cannot add a list to be monitored unless the subscription already as a SharePoint Online site configured to be monitored, and the list monitoring configuration for the subscription has been created. Note that all of the parameters required for this method can be obtained by calling the Get Site Lists method.

REST Endpoint:

https://www.office365mon.com/api/developers/addmonitoredlist

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are getting the list of monitored lists.
Title	String	The title of the list to be monitored.
ListId	Guid	The ID of the list to be monitored.
ViewId	Guid	The ID of the default view of the list to be monitored.
ServerRelativeUrl	String	The server relative URL of the list to be monitored.

Returns:

The call returns nothing if successful; if it fails it returns additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("Title", "My Custom List"));
vals.Add(new KeyValuePair<string, string>("ListId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("ViewId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("ServerRelativeUrl", "/sites/dev/Lists/My
Custom List"));

//make the post to add monitoring for a list
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"addmonitoredlist");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
```

```
{
    //the operation failed

    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Delete a Monitored List

Use this method to delete monitoring for a list.

REST Endpoint:

https://www.office365mon.com/api/developers/deletemonitoredlist

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for which you are getting the list of monitored lists.
ListId	Guid	The ID of the list for which monitoring is being stopped.

Returns:

The call returns nothing if successful; if it fails it returns additional error information.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("ListId", "12345678-1234-1234-1234-
123456789012"));

//make the post to stop monitoring a list
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deletemonitoredlist");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
{
   //the operation failed

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
}
```

Threat Intelligence Monitoring

The operations in this section are used to configure Threat Intelligence monitoring. This type of monitoring allows you to be notified the first time a new malware is found trying

to enter your organization, when you receive an excessive number of malware messages in a particular time frame, or when specific users appear being targeted at malware.

> IMPORTANT: If the user has not consented to the Office365Mon.Com Management Activity Monitor application then monitoring threat intelligence will fail. Consent can be granted either through the Configure Threat Monitoring page on the Office365Mon web site, or through our REST API if you are White Label partner. After consent is granted, like all application access, you need to update the token once every 90 days. If you are a White Label partner, get an updated token using the Office365Mon client ID for Threat Intelligence monitoring.

Assuming the user has provided consent for Threat Intelligence monitoring, these are the scenarios in which you can use the REST API:

- Get the monitoring configuration – if no configuration has been created yet, the string "null" is returned (based on JSON serialization).
- If there isn't any monitoring configuration yet, then use the API to start Threat Intelligence monitoring.
- If you try starting Threat Intelligence monitoring but get a 409 – Conflict response, look at the ReasonPhrase of the HttpResponseMessage. In almost all cases it's because monitoring has already been started. You can use the REST API to stop Threat Intelligence monitoring to stop it and optionally start it again.
- If you just want to change the configuration after monitoring has been started – such as thresholds for sending notifications, whether to send notifications for certain events, etc. – then use the REST API to update the Threat Intelligence monitoring configuration.

Get Threat Intelligence Configuration

Use this method to get the current threat intelligence monitoring configuration for a subscription. Note that it's possible a subscription doesn't have any configuration created yet, in which case a string of "null" is returned.

REST Endpoint:

https://www.office365mon.com/api/developers/getthreatintelconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns an array of Dictionary<string, object> items that describe the Threat Intelligence monitoring configuration.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getthreatintelconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
   string info = await hrm.Content.ReadAsStringAsync();

   if (info != "null"))
   {
      //this is used to convert our JSON data into a dictionary of values
      JavaScriptSerializer jss = new JavaScriptSerializer();

      //set the maximum length value because some of the data,
      //especially health pings, can be quite long
      jss.MaxJsonLength = int.MaxValue;

      var data = jss.Deserialize<Dictionary<string, object>>(info);

      string tenantIdTxt.Text = data["TenantId"].ToString();
```

```
        string generalAnomalyCountTxt.Text = data["GeneralAnomalyCount"].ToString();
        //much more; look at the data for complete details
    }
    else
        MessageBox.Show("This subscription has not configured threat monitoring yet.");
}
else
{
    //the operation failed
}
```

Start Threat Intelligence Monitoring

Use this method to start Threat Intelligence monitoring. This method registers a "hook" with Microsoft to read the stream of data for an Office 365 tenant. If that "hook" has already been registered, then the call will typically fail and return a 409 – Conflict error, and the ReasonPhrase will have more information about the issue.

REST Endpoint:

https://www.office365mon.com/api/developers/startthreatintelmonitoring

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
GeneralAnomalyCount	Int	The number of anomalies to detect within x minutes in order to send notifications. The value "x" is from GeneralAnomalyInterval.
GeneralAnomalyInterval	Int	The number of minutes in which to measure the number of malwares that have been detected and determine whether to send notifications.
GeneralAnomalyNotify	Bool	If true, will send notifications if the GeneralAnomalyCount number of messages is received in the GeneralAnomalyInterval number of minutes.
NotifyOnFirstCase	Bool	If true, will send a notification the first time a malware is detected attempting to come into the organization.
UserAnomalyCount	Int	The number of anomalies to detect within the current day for individual users before sending notifications, if UserAnomalyNotify is true.

UserAnomalyNotify	Bool	If true, will send notifications if the UserAnomalyCount number of malwares are detected for any user in the current day.
UserUploadNotify	Bool	If true, will send notifications if the UserUploadCount number of malwares are uploaded by any user during any single day.
UserUploadCount	Int	The number of times a user can upload malware infected items to SharePoint and/or OneDrive for Business that cause a notification to be triggered. For example, if this value is 3, when it is detected that the same user has uploaded 3 infected items in any day, a notification is sent along with information about who the person uploading the items is.

Returns:

The call returns nothing if successful. If not successful, look at the return data and ReasonPhrase for details on what happened.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("GeneralAnomalyCount", "5"));
vals.Add(new KeyValuePair<string, string>("GeneralAnomalyInterval", "30"));
vals.Add(new KeyValuePair<string, string>("GeneralAnomalyNotify", "true"));
vals.Add(new KeyValuePair<string, string>("NotifyOnFirstCase", "true"));
vals.Add(new KeyValuePair<string, string>("UserAnomalyCount", "5"));
vals.Add(new KeyValuePair<string, string>("UserAnomalyNotify", "true"));
vals.Add(new KeyValuePair<string, string>("UserUploadNotify", "true"));
vals.Add(new KeyValuePair<string, string>("UserUploadCount", "3"));


//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"startthreatintelmonitoring");

//look at the results
if (hrm.IsSuccessStatusCode)
{
```

```
    //it worked
}
else
{
    //the operation failed

    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Update Threat Intelligence Configuration

Use this method to update the Threat Intelligence monitoring configuration. Note that this method differs from the REST API to start Threat Intelligence monitoring in that it one requires one other parameter – the tenant's TenantId value. This value should be retrieved using the REST API to get the Threat Intelligence monitoring configuration. This ensures that the correct configuration record is updated.

REST Endpoint:

https://www.office365mon.com/api/developers/updatethreatintelconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
GeneralAnomalyCount	Int	The number of anomalies to detect within x minutes in order to send notifications. The value "x" is from GeneralAnomalyInterval.
GeneralAnomalyInterval	Int	The number of minutes in which to measure the number of malwares that have been detected and determine whether to send notifications.

GeneralAnomalyNotify	Bool	If true, will send notifications if the GeneralAnomalyCount number of messages is received in the GeneralAnomalyInterval number of minutes.
NotifyOnFirstCase	Bool	If true, will send a notification the first time a malware is detected attempting to come into the organization.
UserAnomalyCount	Int	The number of anomalies to detect within the current day for individual users before sending notifications, if UserAnomalyNotify is true.
UserAnomalyNotify	Bool	If true, will send notifications if the UserAnomalyCount number of malwares are detected for any user in the current day.
UserUploadNotify	Bool	If true, will send notifications if the UserUploadCount number of malwares are uploaded by any user during any single day.
UserUploadCount	Int	The number of times a user can upload malware infected items to SharePoint and/or OneDrive for Business that cause a notification to be triggered. For example, if this value is 3, when it is detected that the same user has uploaded 3 infected items in any day, a notification is sent along with information about who the person uploading the items is.
TenantId	Guid	The Tenant ID for the Office 365 tenant that is being monitored.

Returns:

The call returns nothing if successful. If not successful, look at the return data and ReasonPhrase for details on what happened.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("GeneralAnomalyCount", "5"));
vals.Add(new KeyValuePair<string, string>("GeneralAnomalyInterval", "30"));
```

```
vals.Add(new KeyValuePair<string, string>("GeneralAnomalyNotify", "true"));
vals.Add(new KeyValuePair<string, string>("NotifyOnFirstCase", "true"));
vals.Add(new KeyValuePair<string, string>("UserAnomalyCount", "5"));
vals.Add(new KeyValuePair<string, string>("UserAnomalyNotify", "true"));
vals.Add(new KeyValuePair<string, string>("UserUploadNotify", "true"));
vals.Add(new KeyValuePair<string, string>("UserUploadCount", "3"));
vals.Add(new KeyValuePair<string, string>("TenantId", "12345678-1234-1234-1234-
123456789012"));


//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"updatethreatintelconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
{
   //the operation failed

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
}
```

Stop Threat Intelligence Monitoring

Use this method to stop Threat Intelligence monitoring.

REST Endpoint:

https://www.office365mon.com/api/developers/stopthreatintelmonitoring

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns nothing if successful. If not successful, look at the return data and ReasonPhrase for details on what happened.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));


//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"stopthreatintelmonitoring");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
{
   //the operation failed

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
}
```

Usage Monitoring

The operations in this section are used to configure the Usage Monitoring feature. blah

> IMPORTANT: If the user has not consented to the Office365Mon Office 365 Usage Reports application then usage monitoring will fail. Consent can be granted either through the Configure Usage Monitoring page on the Office365Mon web site, or through our REST API if you are White Label partner. After consent is granted, like all application access, you need to update the token once every 90 days. If you are a White Label partner, get an updated token using the Office365Mon client ID for Usage Monitoring.

Assuming the user has provided consent for Usage Monitoring, these are the scenarios in which you can use the REST API:

- Get the usage monitoring status to find out what monitoring thresholds have been set for Office 365 resources with storage limits (SharePoint, OneDrive and Exchange).
- Toggle the Usage Monitoring status to turn it on or off.

Get Usage Monitoring Configuration

Use this method to get the Usage Monitoring configuration for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getusagemonconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns the usage monitoring configuration data, if any.

Example:

```
public class UsageMonitoringConfiguration
{
    public Guid SubscriptionId { get; set; }
    public int StorageType { get; set; }
    public double Level1Warning { get; set; }
    public double Level2Warning { get; set; }
    public double Level3Warning { get; set; }
}


//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getusagemonconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //it worked
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data can be long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<List<UsageMonitoringConfiguration>>(results);
}
```

```
else
{
    //the operation failed
}
```

Update Usage Monitoring Configuration

Use this method to update the Usage Monitoring configuration for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/configureusagemonconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
StorageType	Int	An enum for the type of storage monitoring you are setting: 0 = Exchange, 1 = SharePoint, 2 = OneDrive, and 3 = ReportOnly, which means you aren't monitoring for storage consumption but you just want to get the tenant consumption reports.
Level1Warning	Double	This is the percentage of storage allocated, when a notification will be fired. For example, if this value is .95, then a notification will be fired when 95% or more of the storage for a resource type has been reached. For StorageType of ReportOnly, this value is ignored and can be zero.
Level2Warning	Double	This is for future use and can be ignored by passing in zero.
Level3Warning	Double	This is for future use and can be ignored by passing in zero.

Returns:

The call returns nothing if updating the configuration was successful. If there was an error, it returns error information in the results.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("EnableLogShipping", "true"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"configureusagemonconfig ");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
{
   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
   //the HTTP response code is here: (int)hrm.StatusCode

   //this DIDN'T work - take the appropriate action
}
```

Log Shipping

Please Note: This feature has been deprecated and is no longer supported.

The operations in this section are used to configure the Log Shipping feature. The Log Shipping feature **works in conjunction** with the Threat Intelligence monitoring feature. It is designed to create a copy of all of the Office 365 activity logs for SharePoint and OneDrive and store it at Office365Mon.Com. For organizations that are required to keep copies of these log files for several months for compliance reasons, it meets that standard and doesn't require any manual management of the log files. In addition to that, you can also get statistical reporting data from the log files for the current month, or for all months for which data has been log shipped to Office365Mon.Com.

> **IMPORTANT:** If the user has not consented to the Office365Mon.Com Management Activity Monitor application then log shipping will fail. Consent can be granted either through the Configure Threat Monitoring page on the Office365Mon web site, or through our REST API if you are White Label partner. After consent is granted, like all application access, you need to update the token once every 90 days. If you are a White Label partner, get an updated token using the Office365Mon client ID for Threat Intelligence monitoring.

Assuming the user has provided consent for Threat Intelligence monitoring, these are the scenarios in which you can use the REST API:

- Get the log shipping status to find out whether we are already doing log shipping for a subscription.
- Toggle the log shipping status to turn it on or off.

Get Log Shipping Status

Use this method to get the status of whether log shipping is enabled for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getlogshipstatus

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns a bool indicating whether log shipping is enabled.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getlogshipstatus");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
   string info = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<bool>(info);
}
else
{
   //the operation failed
}
```

Toggle Log Shipping Status

Use this method to set the status of whether log shipping is enabled for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/togglelogshipstatus

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
EnableLogShipping	Int	Use **0** to disable log shipping; use **1** to enable log shipping.

Returns:

The call returns a bool indicating whether setting the log shipping status was successful.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("EnableLogShipping", "true"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"togglelogshipstatus");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
   string info = await hrm.Content.ReadAsStringAsync();
```

```
    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    //if this is true the change worked; if it's false, it did not
    var data = jss.Deserialize<bool>(info);
}
else
{
    //the operation failed
}
```

Internet Egress Change Notifications

The operations in this section are used to configure notifications for when an Internet egress point changes in a location where a Distributed Probes and Diagnostics agent is deployed. The egress location is checked once a day by the agent, and the current egress information is sent to the Office365Mon.Com REST service. If it has changed since it was last checked, then the new information is added to the historical log of egress points for that particular location. Additionally, a user can configure whether or not to receive notifications when that happens in the Configure Office 365 Distributed Probes page on the Office365Mon.Com web site. This particular API allows you to also configure that setting programmatically.

These are the scenarios in which you can use the REST API:

- Get the egress notification status to find out whether they are enabled for a subscription.
- Toggle the egress notification status to turn it on or off.

Get Egress Notification Status

Use this method to get the status of whether egress notifications enabled for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/getinteretegressnotifystatus

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns a bool indicating whether egress notifications enabled.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getinteretegressnotifystatus");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
   string info = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<bool>(info);
}
else
{
   //the operation failed
}
```

Toggle Egress Notification Status

Use this method to set the status of whether egress notifications enabled for the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/toggleinteretegressnotifystatus

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
EnableLogShipping	Int	Use **0** to disable egress notifications; use **1** to enable egress notifications.

Returns:

The call returns a bool indicating whether setting the egress notification status was successful.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("EnableFeature", "true"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"toggleinteretegressnotifystatus");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
   string info = await hrm.Content.ReadAsStringAsync();
```

```
    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    //if this is true the change worked; if it's false, it did not
    var data = jss.Deserialize<bool>(info);
}
else
{
    //the operation failed
}
```

Microsoft Teams Monitoring

The operations in this section are used to configure monitoring features for Microsoft Teams. Please note that "Microsoft Teams" is an umbrella term that describes several different features, and the monitoring options for them vary based on the Office365Mon license you have. If you try and configure a Teams monitoring feature that is not part of a license for the subscription, it will fail and generate an exception that it returns to your calling code.

Get Teams IDs

Use this method to get a list of Ids for the Microsoft Teams to which the user who obtained the access token for monitoring is joined. The Team ID is needed to add or remove monitoring for Microsoft Teams.

REST Endpoint:

https://www.office365mon.com/api/developers/getteamsgroupids

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.

SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns a list of Team IDs. If an access token has not been obtained for Teams monitoring in the subscription, then the list will be empty. The return data is structured as follows:

Name	Type	Description
Id	Guid	The Id for the Team.
Description	String	The description for the Team.
DisplayName	String	The name of the Team.
EmailAddress	String	The email address for the Team, if applicable.
MailEnabled	Bool	A Boolean indicating whether the Team is mail-enabled.
MailNickname	String	The email nickname for the Team, if applicable.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"getteamsgroupids");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
   string info = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   //this line of code assumes you created a class called
   //MicrosofTeamGroup elsewhere that is defined as described above
   var data = jss.Deserialize<List<MicrosoftTeamGroup>>(results);
```

```
}
else
{
    //the operation failed
}
```

Add Or Update Teams Monitoring Configuration

Use this method to add Teams monitoring to the subscription, and/or update the Microsoft Team that is used for monitoring Teams.

REST Endpoint:

https://www.office365mon.com/api/developers/addorupdateteamsmonitoringconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.
GroupId	Guid	The ID of the Microsoft Team you want to use for monitoring Teams.

Returns:

A 200 HTTP response if the call works; an error number and message if it fails.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("GroupId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list monitoring configuration
```

```
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"addorupdateteamsmonitoringconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
{
   //the operation failed

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
   //the HTTP response code is here: (int)hrm.StatusCode

   //this DIDN'T work - take the appropriate action
}
```

Get Teams Monitoring Configuration

Use this method to get Teams monitoring configuration from the subscription. Currently it just returns a record for each type of Teams feature monitoring that is enabled. You can look at the MonitoredResourceType value in the return data for each configuration record to determine which feature is being monitored. The current list of features that can be monitored is:

- 15 – Microsoft Teams
- 16 – Microsoft Teams Channel

REST Endpoint:

https://www.office365mon.com/api/developers/getteamsmonitoringconfigs

Input Parameters:

Name	Type	Description

DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

The call returns a list of Teams monitoring configuration records when it works; if it fails you will get an HTTP response with a standard status code. The return data is structured as follows:

Name	Type	Description
SubscriptionId	Guid	The ID for Subscription being monitored.
GroupId	Guid	The ID of the Microsoft Team being monitored.
IsBuilding	Bool	A Boolean indicating whether the list of joined Teams for the monitoring feature is being enumerated still.
GroupName	String	The name of the Team.
GroupDescription	String	The description of the Team.
MonitoredResourceType	Int	A value that indicates the Microsoft Teams feature that is being monitored with this configuration record. The list of possible values are documented above.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deleteteamsmonitoringconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //it worked
   string info = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
```

```
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    //this line of code assumes you created a class called
    //TeamsMonitoringConfiguration elsewhere that is defined as described above
    var data = jss.Deserialize<List<TeamsMonitoringConfiguration>>(results);
}
else
{
    //the operation failed
}
```

Remove Teams Monitoring Configuration

Use this method to remove Teams monitoring from the subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/deleteteamsmonitoringconfig

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The subscription ID for that you are checking.

Returns:

A 200 HTTP response if the call works; an error number and message if it fails.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";


//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
```

```
//make the post to get the list monitoring configuration
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"deleteteamsmonitoringconfig");

//look at the results
if (hrm.IsSuccessStatusCode)
{
   //it worked
}
else
{
   //the operation failed

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
   //the HTTP response code is here: (int)hrm.StatusCode

   //this DIDN'T work - take the appropriate action
}
```

Resellers

As mentioned in the beginning of this document, if you've registered with Office365Mon as a reseller then there are a few additional APIs for you to use. To create or delete subscriptions you **must** only use the Reseller APIs. They are also the only APIs that you can use to add or remove plans from that subscription, where a plan is some collection of features that the customer is charged money for.

All of the Reseller* APIs do the same basic validation check before attempting to process your request:

- Does the Dev Subscription have the Subscription Management API feature currently?
- Is the user that authenticated to the REST endpoint an Admin on the Dev Subscription?
- Is the Dev Subscription registered as a reseller?
- If you are using one of the Reseller* APIs to modify a subscription – such as deleting it, adding or deleting plans, or getting a list of plans for the subscription – did you create that subscription?

If any of the validation checks fail, then the request is not processed and the HTTP status code you will get in return is 401 – Unauthorized. The ErrorMessage provides more details as to which of the validation tests failed.

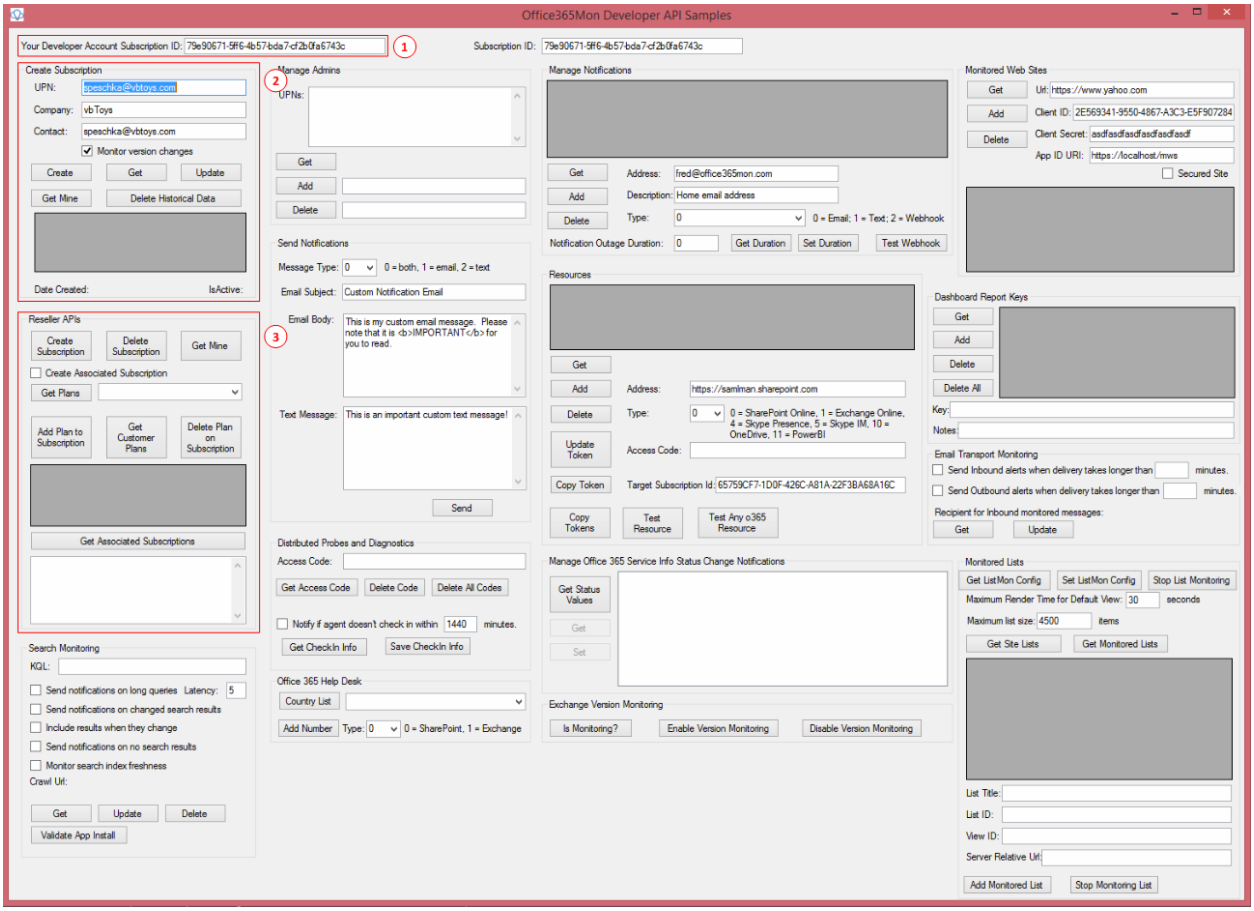
The Reseller* APIs also include a specific set of error numbers to provide you with as much information as possible when using them. These error numbers are used across all of the Reseller* APIs so are documented here at the beginning of the section. You can look at the status code of an HTTP request to determine the error number and message. Here's some generic sample code for parsing the return code using C# and the HttpResponseMessageClass (in the code below "hrm" is an HttpResponseMessage that was used to make a REST call to the Subscription Management API):

```
if (hrm.IsSuccessStatusCode)
{
   //this worked - take the appropriate action
}
else
{
   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
   //the HTTP response code is here: (int)hrm.StatusCode

   //this DIDN'T work - take the appropriate action
}
```

Here are the error codes used by the Reseller* APIs:

Error Number	Description
100 – Unexpected error	A general error; details are included in the ErrorMessage.
110 – Not a valid subscription	The Subscription Management API is not currently an active feature in the Dev Subscription or the authenticated is not an Admin in the Dev Subscription.
120 – Reseller not found	The Dev Subscription ID is not registered as a reseller. Please contact support@office365mon.com to discuss how you can become a reseller.
130 – Missing required parameters	One or more required parameters are missing. See the Error Message for a list of the required parameters.
140 – Plan already exists for this subscription	You are trying to add a plan to a subscription that already has that plan.
150 – Add plan general error	A general error adding a plan; details are included in the ErrorMessage.
160 – Plan not configured	Your Dev Subscription ID is not configured to sell this plan. Please contact support@office365mon.com for assistance.
170 – Add subscription general error	A general error adding a subscription; details are included in the ErrorMessage.
180 – Delete plan general error	A general error deleting a plan; details are included in the ErrorMessage
190 – Invalid plan	The plan you are trying to add does not exist.
200 – Delete subscription general error	A general error deleting a subscription; details are included in the ErrorMessage
210 – Subscription is not a premium feature	You tried to add a subscription as a plan to a subscription. A subscription is not a plan though and cannot be added as one.
220 – Dev subscription did not create the subscription being modified	The subscription you are trying to modify by deleting it, adding or deleting a plan to it, or getting the list of plans for it, is not one that you created with your DevSubscriptionId.

If you are a reseller and have a small number of customers, you may not wish to create your own web pages to go through the process of creating subscriptions. In that case, we recommend that you use the sample application for this SDK, which you can download from https://www.office365mon.com/DevApiSample.zip. It is a simple Windows application, but is fully functional using the Office365Mon REST API. If you wish to use it to create subscriptions, use the elements of the application that are highlighted below:



To create a subscription:

1. Enter your reseller Subscription ID in the text box.
2. Fill out the information in the Create Subscription section.
3. Click the "Create Subscription" button in the Reseller APIs section. DO NOT click the "Create" button in the "Create Subscription" section, because that will not activate the reseller features.

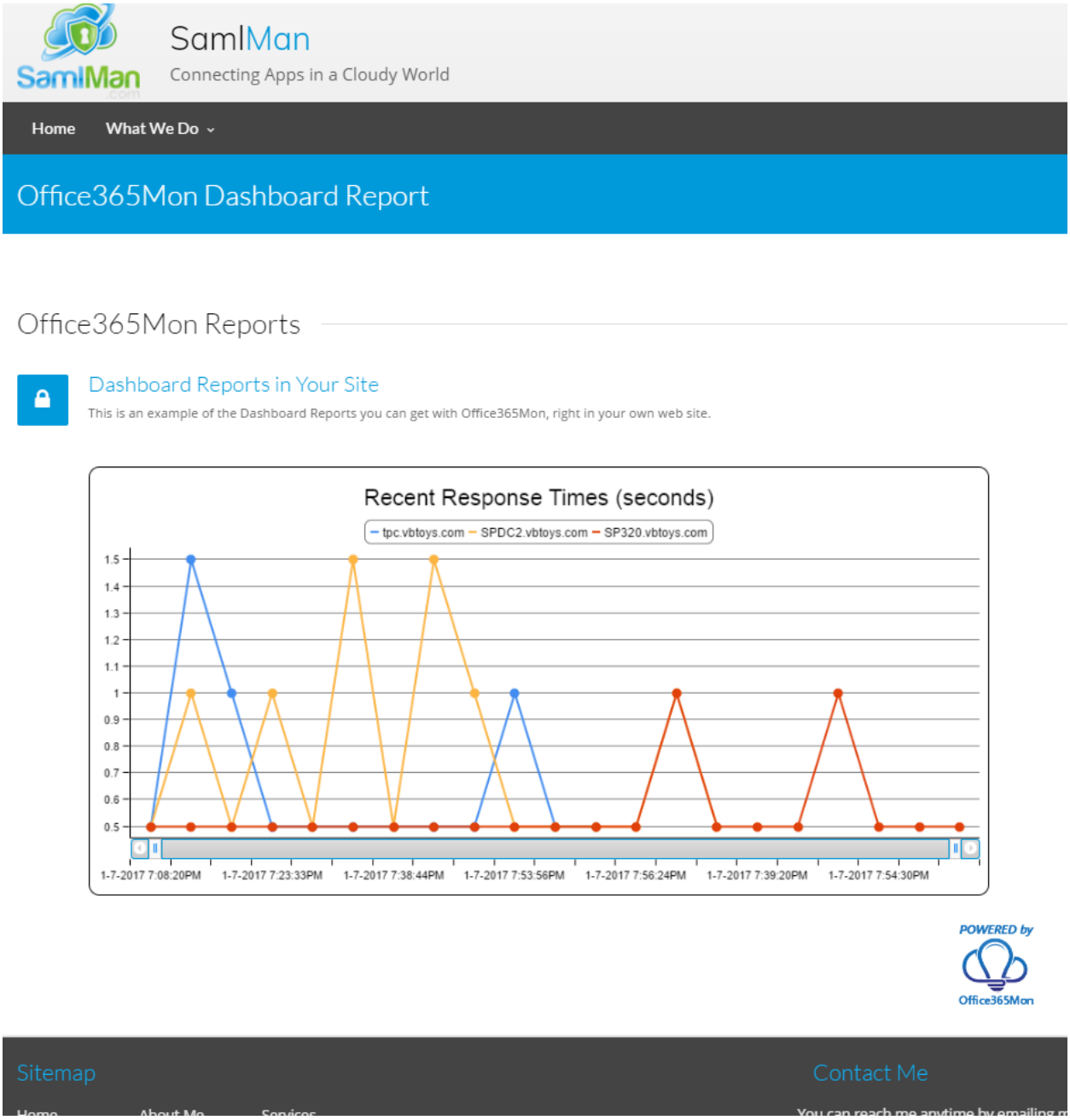
After you've created the subscription, you can manage the other features for it using the sample application, or you can add your UPN as a subscription admin and then manage it in the browser on office365mon.com.

Reseller Branding Requirements

If you are creating your own web pages to manage any aspect of Office365Mon subscriptions, please remember that you must include “Powered By Office365Mon” content on each page. We have provided multiple images that you can use for this purpose:

Image	Url
	https://www.office365mon.com/images/o365mon_logo_96x96.png
	https://www.office365mon.com/images/o365mon_logo_transparent_96x96.png
POWERED by Office365Mon	https://www.office365mon.com/images/o365mon_powered_by_logo.png
POWERED by Office365Mon	https://www.office365mon.com/images/o365mon_powered_by_transparent_logo.png

Here’s an example of it in third-party site:



Associated Subscriptions

Associated subscriptions are part of a feature that allows you to create multiple Office365Mon reseller subscriptions for a single customer of yours. When you do this, you are only charged for the original, or "Parent" Office365Mon subscription. This allows your customers to have multiple Office365Mon subscriptions, the same as do customers that purchase from Office365Mon directly.

Please note that the same licensing rules apply though – all Associated Subscriptions **can only be used** to monitor the same Office 365 tenant as the "Parent" Office365Mon subscription. Trying to use it across multiple Office 365 tenants is a licensing violation and if you do so all subscriptions will be cancelled.

Working with Associated Subscriptions is easy and straightforward. If you want to create an AssociatedSubscription, you need to include one additional parameter called ParentSubscriptionId when calling the API to create a new subscription. To delete an AssociatedSubscription, you don't need to do anything different – just use the reseller delete subscription API. Additionally, you can also get a list of the Id's for all of the AssociatedSubscriptions for any parent subscription using our Get Associated Subscriptions API. The specifics for all of these are covered in the documentation and examples below.

Create a Subscription

Use this method to create a new subscription.

REST Endpoint:

https://www.office365mon.com/api/developers/resellercreatesub

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
UPN	String	The UPN for the initial administrator of the Office365Mon subscription.
CompanyName	String	The company whose name the subscription is for.
ContactEmail	String	An email address that can be used for pricing changes, feature changes, etc. for the subscription
SubscriptionId	Int	**OPTIONAL.** If this parameter is not included, or if it is 0, an Office365Mon subscription is created. If it is 1, an AzureServiceMon subscription is created.
ParentSubscriptionId	Guid	**OPTIONAL.** Use this property **only** if you want the new subscription to be an AssociatedSubscription of this one.

Returns:

Name	Type	Description
SubscriptionId	Guid	The subscription ID for the new Office365Mon subscription.

If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("UPN", "someupn@foo.com"));
vals.Add(new KeyValuePair<string, string>("CompanyName", "My Company"));
vals.Add(new KeyValuePair<string, string>("ContactEmail", "admin@mycompany.com"));

//we can leave this out to make an Office365Mon subscription, but in this
//case we're going to make it an AzureServiceMon subscription
vals.Add(new KeyValuePair<string, string>("SubscriptionType", "1"));

//if this is an AssociatedSubscription you would add this:
//vals.Add(new KeyValuePair<string, string>("ParentSubscriptionId", "12345678-1234-1234-
1234-123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellercreatesub");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	string subscriptionId = data["SubscriptionId"].ToString();
}
else
{
	//this DIDN'T work - take the appropriate action

	//when there is an error you will get info about it in the results
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();
```

```
        //set the maximum length value because some of the data,
        //especially health pings, can be quite long
        jss.MaxJsonLength = int.MaxValue;

        var data = jss.Deserialize<Dictionary<string, object>>(results);

        //get the error number and message from the API
        int errNum = int.Parse(data["ErrorNumber"].ToString());
        string errMsg = data["ErrorMessage"].ToString();
}
```

Delete a Subscription

Use this method to delete a subscription; this only works for subscriptions created with the Reseller API.

REST Endpoint:

https://www.office365mon.com/api/developers/resellerdeletesub

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription being deleted. It can be any subscription – a standard and/or parent subscription, or an Associated Subscription. **IMPORTANT:** If it's a parent subscription, then **all** AssociatedSubscriptions are also deleted.
UPN	String	The UPN for the administrator of the Office365Mon subscription.

Returns:

	Type	Description
"true"	string	Returns "true" if the call succeeded. If it fails then the HTTP status code upon return will be something other than 200 – OK.

If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("UPN", "someupn@foo.com"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellerdeletesub");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//technically this should always be true; if it's not then
	//it should raise an error which you would catch below
	bool worked = bool.Parse(results);
}
else
{
	//this DIDN'T work - take the appropriate action

	//when there is an error you will get info about it in the results
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	//get the error number and message from the API
	int errNum = int.Parse(data["ErrorNumber"].ToString());
	string errMsg = data["ErrorMessage"].ToString();
}
```

Get Associated Subscriptions

Use this method to get all of the subscriptions that are associated with another subscription. This only works for subscriptions created with the Reseller API.

REST Endpoint:

https://www.office365mon.com/api/developers/resellermysubs

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The "Parent" subscription for any Associated Subscriptions.

Returns:

A list of GUIDs. Each GUID is the Subscription ID of an AssociatedSubscription. You can use that to get details about it, add a Plan to it, delete it, etc. If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to get the list of associated subscriptions
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellergetassociatedsubs");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string subData = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<List<Guid>>(results);
}
else
{
```

```
        //this DIDN'T work - take the appropriate action

        //when there is an error you will get info about it in the results
        string results = await hrm.Content.ReadAsStringAsync();

        //this is used to convert our JSON data into a dictionary of values
        JavaScriptSerializer jss = new JavaScriptSerializer();

        //set the maximum length value because some of the data,
        //especially health pings, can be quite long
        jss.MaxJsonLength = int.MaxValue;

        var data = jss.Deserialize<Dictionary<string, object>>(results);

        //get the error number and message from the API
        int errNum = int.Parse(data["ErrorNumber"].ToString());
        string errMsg = data["ErrorMessage"].ToString();
}
```

Get My Reseller Subscriptions

Use this method to get all of the subscriptions that are owned by this reseller. This only works for subscriptions created with the Reseller API.

REST Endpoint:

https://www.office365mon.com/api/developers/resellermysubs

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.

Returns:

An array of information about each subscription:

Name	Type	Description
CompanyName	String	The name of the company for the subscription.
ContactEmail	String	The email address that will be used to contact the subscription owner in the case of pricing changes, feature changes, licensing changes, general announcements, etc.
DateCreated	DateTime	The date the subscription was created.

IsActive	Int	An integer representing the active state of the subscription.	
		Active State	**Code**
		Inactive	0
		Active	1
MonitorVersions	Int	An integer describing whether version changes for Office 365 resources should be monitored	
		Monitoring State	**Code**
		Don't monitor for and notify me of changes in the version of my Office 365 resources.	0
		Do monitor for and notify me of changes in the version of my Office 365 resources.	1
SubscriptionType	Int	The type of subscription: 0 = Office365Mon; 1 = AzureServiceMon.	

If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL + "resellermysubs");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string subData = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	Array data = jss.DeserializeObject(subData) as Array;

	foreach (var sub in data)
	{
```

```
                Dictionary<string, object> items = sub as Dictionary<string, object>;

                string CompanyName = items["CompanyName"].ToString();
                string ContactEmail = items ["ContactEmail"].ToString();
                DateTime DateCreated DateTime.Parse(items ["DateCreated"]);
                string IsActive = items ["IsActive"].ToString();
                string MonitorVersions = items ["MonitorVersions"].ToString();
        }
}
else
{
        //this DIDN'T work - take the appropriate action

        //when there is an error you will get info about it in the results
        string results = await hrm.Content.ReadAsStringAsync();

        //this is used to convert our JSON data into a dictionary of values
        JavaScriptSerializer jss = new JavaScriptSerializer();

        //set the maximum length value because some of the data,
        //especially health pings, can be quite long
        jss.MaxJsonLength = int.MaxValue;

        var data = jss.Deserialize<Dictionary<string, object>>(results);

        //get the error number and message from the API
        int errNum = int.Parse(data["ErrorNumber"].ToString());
        string errMsg = data["ErrorMessage"].ToString();
}
```

Get Plans

Use this method to get the list of Plans that can be added and deleted from a subscription created with the Reseller API.

REST Endpoint:

https://www.office365mon.com/api/developers/resellergetfeatureplans

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.

Returns:

Name	Type	Description
	List<ResellerFeaturePlan>	A list of ResellerFeaturePlan items that contains information about each plan that can be added or deleted from a subscription.
SubscriptionType	Int	**OPTIONAL.** If this parameter is not included, or if it is 0, Office365Mon plans are returned. If it is 1, AzureServiceMon plans are returned.

If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

The ResellerFeaturePlan class is defined like this:

```
public class ResellerFeaturePlan
{
	public string PlanId { get; set; }
	public string PlanName { get; set; }

	public List<PlanFeatures> Features { get; set; }

	public ResellerFeaturePlan()
	{
		this.Features = new List<PlanFeatures>();
	}

	public ResellerFeaturePlan(string PlanId, string PlanName)
	{
		this.PlanId = PlanId;
		this.PlanName = PlanName;
		this.Features = new List<PlanFeatures>();
	}

	public class PlanFeatures
	{
		public Guid FeatureId { get; set; }
		public string FeatureName { get; set; }

		public PlanFeatures() { }

		public PlanFeatures(Guid FeatureId, string FeatureName)
		{
			this.FeatureId = FeatureId;
			this.FeatureName = FeatureName;
		}
	}
}
```

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";
```

```
//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//we can leave this out to get Office365Mon plans, but in this
//case we're going to get AzureServiceMon plans
vals.Add(new KeyValuePair<string, string>("SubscriptionType", "1"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellergetfeatureplans");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<List<ResellerFeaturePlan>>(results);
 }
else
{
	//this DIDN'T work - take the appropriate action
}
```

Add a Plan to a Subscription

Use this method to add a plan to a subscription; this only works for subscriptions created with the Reseller API.

REST Endpoint:

https://www.office365mon.com/api/developers/reselleraddfeatureplan

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription having a plan added.
PlanId	String	The ID of the plan being added to the subscription.

UPN	String	The UPN for the administrator of the Office365Mon subscription.

Returns:

	Type	Description
"true"	string	Returns "true" if the call succeeded. If it fails then the HTTP status code upon return will be something other than 200 – OK.

If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("PlanId", "ENTERPRISE_FEATURES"));
vals.Add(new KeyValuePair<string, string>("UPN", "someupn@foo.com"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"reselleraddfeatureplan");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//technically this should always be true; if it's not then
	//it should raise an error which you would catch below
	bool worked = bool.Parse(results);
}
else
{
	//this DIDN'T work - take the appropriate action

	//when there is an error you will get info about it in the results
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();
```

```
        //set the maximum length value because some of the data,
        //especially health pings, can be quite long
        jss.MaxJsonLength = int.MaxValue;

        var data = jss.Deserialize<Dictionary<string, object>>(results);

        //get the error number and message from the API
        int errNum = int.Parse(data["ErrorNumber"].ToString());
        string errMsg = data["ErrorMessage"].ToString();
}
```

Delete a Plan from a Subscription

Use this method to delete a plan from a subscription; this only works for subscriptions created with the Reseller API.

REST Endpoint:

https://www.office365mon.com/api/developers/resellerdeletefeatureplan

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription having a plan deleted.
PlanId	String	The ID of the plan being deleted from the subscription.
UPN	String	The UPN for the administrator of the Office365Mon subscription.

Returns:

	Type	Description
“true”	string	Returns “true” if the call succeeded. If it fails then the HTTP status code upon return will be something other than 200 – OK.

If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

Example:

```
//all REST endpoints start here
```

```
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("PlanId", "ENTERPRISE_FEATURES"));
vals.Add(new KeyValuePair<string, string>("UPN", "someupn@foo.com"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellerdeletefeatureplan");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//technically this should always be true; if it's not then
	//it should raise an error which you would catch below
	bool worked = bool.Parse(results);
}
else
{
	//this DIDN'T work - take the appropriate action

	//when there is an error you will get info about it in the results
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	//get the error number and message from the API
	int errNum = int.Parse(data["ErrorNumber"].ToString());
	string errMsg = data["ErrorMessage"].ToString();
}
```

Get Customer Plans

Use this method to get the list of Plans that a customer with a subscription created by the Reseller API has.

REST Endpoint:

https://www.office365mon.com/api/developers/resellergetcustomerfeatureplans

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled.
SubscriptionId	Guid	The ID of the subscription for which you are getting the list of plans.

Returns:

Name	Type	Description
	List<CustomerPlan>	A list of CustomerPlan items that contains information about each plan associated with the subscription.

If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

The CustomerPlan class is defined like this:

```
public class CustomerPlan
{
	public Guid SubscriptionId { get; set; }
	public string PlanId { get; set; }
	public string DisplayName { get; set; }
	public DateTime DateCreated { get; set; }
	public int Price { get; set; }
	public string CompanyName { get; set; }
	public string ContactEmail { get; set; }
}
```

Note that the Price property is for internal user and does not necessarily reflect the price that a reseller is charged for the plan.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));
```

```
//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellergetcustomerfeatureplans");

//look at the results
if (hrm.IsSuccessStatusCode)
{
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<List<CustomerPlan>>(results);
 }
else
{
	//this DIDN'T work - take the appropriate action

	//when there is an error you will get info about it in the results
	string results = await hrm.Content.ReadAsStringAsync();

	//this is used to convert our JSON data into a dictionary of values
	JavaScriptSerializer jss = new JavaScriptSerializer();

	//set the maximum length value because some of the data,
	//especially health pings, can be quite long
	jss.MaxJsonLength = int.MaxValue;

	var data = jss.Deserialize<Dictionary<string, object>>(results);

	//get the error number and message from the API
	int errNum = int.Parse(data["ErrorNumber"].ToString());
	string errMsg = data["ErrorMessage"].ToString();
}
```

Convert a Trial Subscription to a Paid Subscription

Use this method to convert a customer subscription that was created as a Trial subscription with one DevSubscriptionId to a paid subscription using another DevSubscriptionId. This is for partners that have two Office365Mon subscriptions, and with the SubscriptionId of one they create trial subscriptions for their customers, and with the other they create paid subscriptions for their customers.

REST Endpoint:

https://www.office365mon.com/api/developers/resellerconverttrialtopaid

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled that is used for your customer paid subscriptions.
TrialDevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled that is used for your customer trial subscriptions. It should be the DevSubscriptionId that you used when originally creating the customer subscription.
CustomerSubscriptionId	Guid	The ID of the customer's Office365Mon subscription that is being converted to a paid subscription.

Returns:

Name	Type	Description
	List< ProbeConfigurationInfo>	A list of ProbeConfigurationInfo items that contains information about each Distributed Probe resource configuration associated with the subscription.

If the call succeeds the HttpStatusCode of the result will be 200 – OK. If the call fails, the status code will be one of the following:

- 401 – if the DevSubscriptionId and/or TrialDevSubscriptionId either does not have the Subscription Management API feature enabled, or the call to the API is made with an account that is not an administrator on either subscription. Also you will get this error if the TrialDevSubscriptionId was not the DevSubscriptionId used to create the customer subscription.
- 409 – if the subscription associated with the TrialDevSubscription was not configured to be used for trial subscriptions. Also, there is an issue in the actual internal process to convert the customer subscription to the new DevSubscriptionId, this result is also returned.

If the call fails, the status code will be 400 – Bad Request. Examine the ErrorNumber and ErrorMessage for more details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
```

```
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("TrialDevSubscriptionId", "12345678-1234-1234-
1234-123456789012"));
vals.Add(new KeyValuePair<string, string>("CustomerSubscriptionId", "12345678-1234-1234-
1234-123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellerconverttrialtopaid");

//look at the results
if (hrm.IsSuccessStatusCode)
{
       //it worked
}
else
{
   //this DIDN'T work - take the appropriate action

   //when there is an error you will get info about it in the results
   string results = await hrm.Content.ReadAsStringAsync();

   //this is used to convert our JSON data into a dictionary of values
   JavaScriptSerializer jss = new JavaScriptSerializer();

   //set the maximum length value because some of the data,
   //especially health pings, can be quite long
   jss.MaxJsonLength = int.MaxValue;

   var data = jss.Deserialize<Dictionary<string, object>>(results);

   //get the error number and message from the API
   int errNum = int.Parse(data["ErrorNumber"].ToString());
   string errMsg = data["ErrorMessage"].ToString();
}
```

Get Distributed Probe Configuration for One Subscription

Use this method to retrieve the Distributed Probe configuration information for a single Office365Mon subscription for which you are the reseller.

REST Endpoint:

https://www.office365mon.com/api/developers/resellermyprobeinfo

Input Parameters:

Name	Type	Description

DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled that is used for your customer paid subscriptions.
SubscriptionId	Guid	The ID of the Office365Mon subscription for which you wish to retrieve the Distributed Probe configuration.

Returns:

If the call succeeds it will return a List<ProbeConfigurationInfo>. The ProbeConfigurationInfo class is defined like this:

```
public class ProbeConfigurationInfo
{
    public string AccessToken { get; set; }
    public string RefreshToken { get; set; }
    public string EndpointUri { get; set; }
    public string ResourceId { get; set; }
    public string ClientId { get; set; }
    public string ClientPassword { get; set; }
    public int ResourceType { get; set; }
    public string UPN { get; set; }
    public string SiteAddress { get; set; }
    public int OutageDurationToEmail { get; set; }
}
```

If the call fails you can examine the ErrorNumber and ErrorMessage in the return data for more details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));
vals.Add(new KeyValuePair<string, string>("SubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellermyprobeinfo");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();
```

```
    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<List<ProbeConfigurationInfo>>(results);

    foreach(ProbeConfigurationInfo info in data)
    {
        //do something
    }
}
else
{
    //this DIDN'T work - take the appropriate action

    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```

Get Distributed Probe Configuration for All Subscriptions

Use this method to retrieve the Distributed Probe configuration information for all of the Office365Mon subscriptions for which you are the reseller.

REST Endpoint:

https://www.office365mon.com/api/developers/resellermyprobeinfos

Input Parameters:

Name	Type	Description
DevSubscriptionId	Guid	The ID of your Office365Mon subscription that has the Subscription Management API feature enabled that is used for your customer paid subscriptions.

Returns:

If the call succeeds it will return a List<ProbeConfigurationInfo>. The ProbeConfigurationInfo class is defined in the previous section here.

If the call fails you can examine the ErrorNumber and ErrorMessage in the return data for more details.

Example:

```
//all REST endpoints start here
const string BASE_REST_URL = "https://www.office365mon.com/api/developers/";

//create the collection of values to send to the POST
List<KeyValuePair<string, string>> vals = new List<KeyValuePair<string, string>>();

//add them
vals.Add(new KeyValuePair<string, string>("DevSubscriptionId", "12345678-1234-1234-1234-
123456789012"));

//make the post to create the subscription
HttpResponseMessage hrm = await MakePostRequest(vals, BASE_REST_URL +
"resellermyprobeinfos");

//look at the results
if (hrm.IsSuccessStatusCode)
{
    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;

    var data = jss.Deserialize<List<ProbeConfigurationInfo>>(results);

    foreach(ProbeConfigurationInfo info in data)
    {
        //do something
    }
}
else
{
    //this DIDN'T work - take the appropriate action

    //when there is an error you will get info about it in the results
    string results = await hrm.Content.ReadAsStringAsync();

    //this is used to convert our JSON data into a dictionary of values
    JavaScriptSerializer jss = new JavaScriptSerializer();

    //set the maximum length value because some of the data,
    //especially health pings, can be quite long
    jss.MaxJsonLength = int.MaxValue;
```

```
    var data = jss.Deserialize<Dictionary<string, object>>(results);

    //get the error number and message from the API
    int errNum = int.Parse(data["ErrorNumber"].ToString());
    string errMsg = data["ErrorMessage"].ToString();
}
```