



Welcome to Microsoft Live@edu

Connected Campus using Exchange Web Services API

| | |
|--|-----------|
| Connected Campus using Exchange Web Services API | 1 |
| About This Document..... | 4 |
| Purpose | 4 |
| Executive Summary..... | 4 |
| Exchange Web Services Managed API | 5 |
| School Information Systems and EWS..... | 7 |
| EWS Managed API Scenarios..... | 8 |
| Scenario: Course Registration..... | 8 |
| Scenario: Publishing school events to student’s calendar | 15 |
| Scenario: Campus alerts e-mail..... | 17 |
| Scenario: Displaying mail and calendar data on student’s portal..... | 17 |
| Installation..... | 20 |
| Summary..... | 20 |
| Glossary..... | 21 |
| References..... | 21 |

The information contained in this document relates to pre-release software products and services that may be substantially modified before its first commercial release. Accordingly, the information may not accurately describe or reflect the software product when first commercially released. THIS GUIDE IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY AND MICROSOFT CORP. MAKES NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS GUIDE OR THE INFORMATION CONTAINED IN IT.

© 2010 Microsoft Corp. All rights reserved.

About This Document

Purpose

Exchange Web Services (EWS) provides the functionality to enable client applications to communicate with the Exchange Server. EWS provides access to much of the same data that is made available through Microsoft Office Outlook. EWS clients can integrate Outlook data into Line-of-Business (LOB) applications. This document does not include implementation and user guides for deployment, migration, or specific product features.

Executive Summary

The Live@edu offering from Microsoft provides a hosted e-mail solution and other online tools available for schools to create a “connected campus” environment. The advantages of this offering are:

- Microsoft Outlook Live provides the best-in-class features of Microsoft Exchange Server with the ease and convenience of online hosted e-mail.
- Seamless communication and collaboration between students, professors, and alumni, whether working from desktop, a web console or a mobile phone.
- Ability to have an on-premises messaging system and Outlook Live domain that share the same domain suffix, and thereby having the shared address book, contacts, and distribution lists.
- Outlook Live provides Exchange Web Services (EWS) API and PowerShell commands for seamless integration with different applications, Web portals, and other information systems on campus.
- Integration with the school’s Active Directory, thereby providing single-sign-on capabilities.

After Outlook Live is deployed in a university, the hosted e-mail service becomes part of the various other information systems existing in the school campus. This creates the need for integrating other information systems with Outlook Live. This can be easily achieved by leveraging EWS.

This document focuses on seamless integration of campus applications, information systems and portals to Outlook Live using EWS API and PowerShell. We have chosen a scenario of integrating a course registration system with Outlook Live using the EWS Managed API. The scenario explains the EWS Managed API with code snippets so that you can plan for integrating campus applications with Outlook Live.

Exchange Web Services Managed API

The Exchange Web Services Managed API provides access to Exchange resources like e-mail, calendar, contacts, distribution lists, tasks, and Exchange folders. The EWS Managed API is a fully object-oriented API built on the EWS XML protocol; it provides an easy-to-learn, easy-to-use, and easy-to-maintain .NET interface to Exchange Web Services. Figure 1 shows the Exchange resources accessed by EWS API.

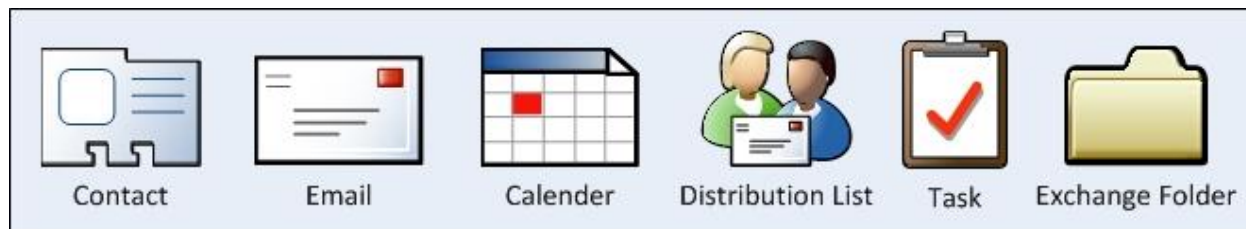


Figure 1: Exchange Resources

The EWS Managed API provides interfaces for managing the Exchange resources. Managing Exchange resources include features for creating, updating, retrieving, and deleting data in an object-oriented fashion. Each item type and folder type is exposed through a dedicated class. Figure 2 and Figure 3 illustrate the Item class hierarchy and Folder class hierarchy as exposed by the EWS Managed API.

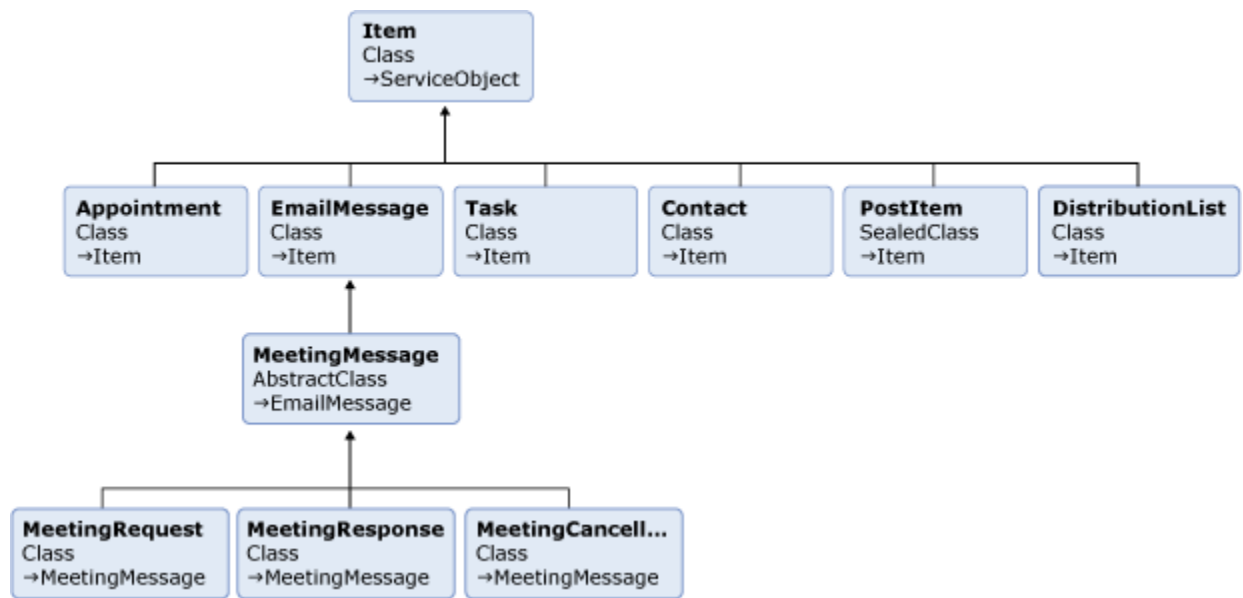


Figure 2 Item Class Hierarchies

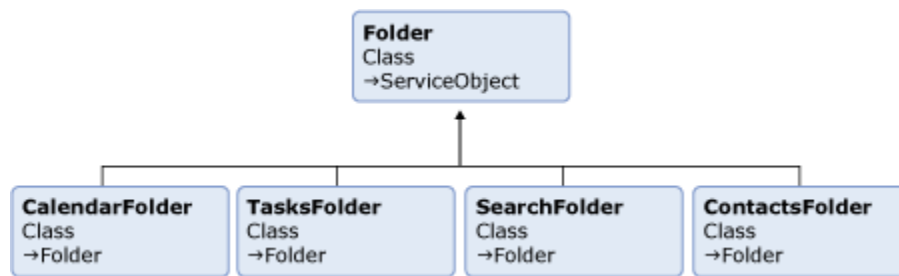


Figure 3 Folder class hierarchies

School Information Systems and EWS

School Information Systems comprise different out-of-the-box applications and custom-built solutions to facilitate various school functions. Typical school information systems include an enrollment management system, a course registration system, a financial aid system, learning management system, billing system, HR management system, and an ERP system.

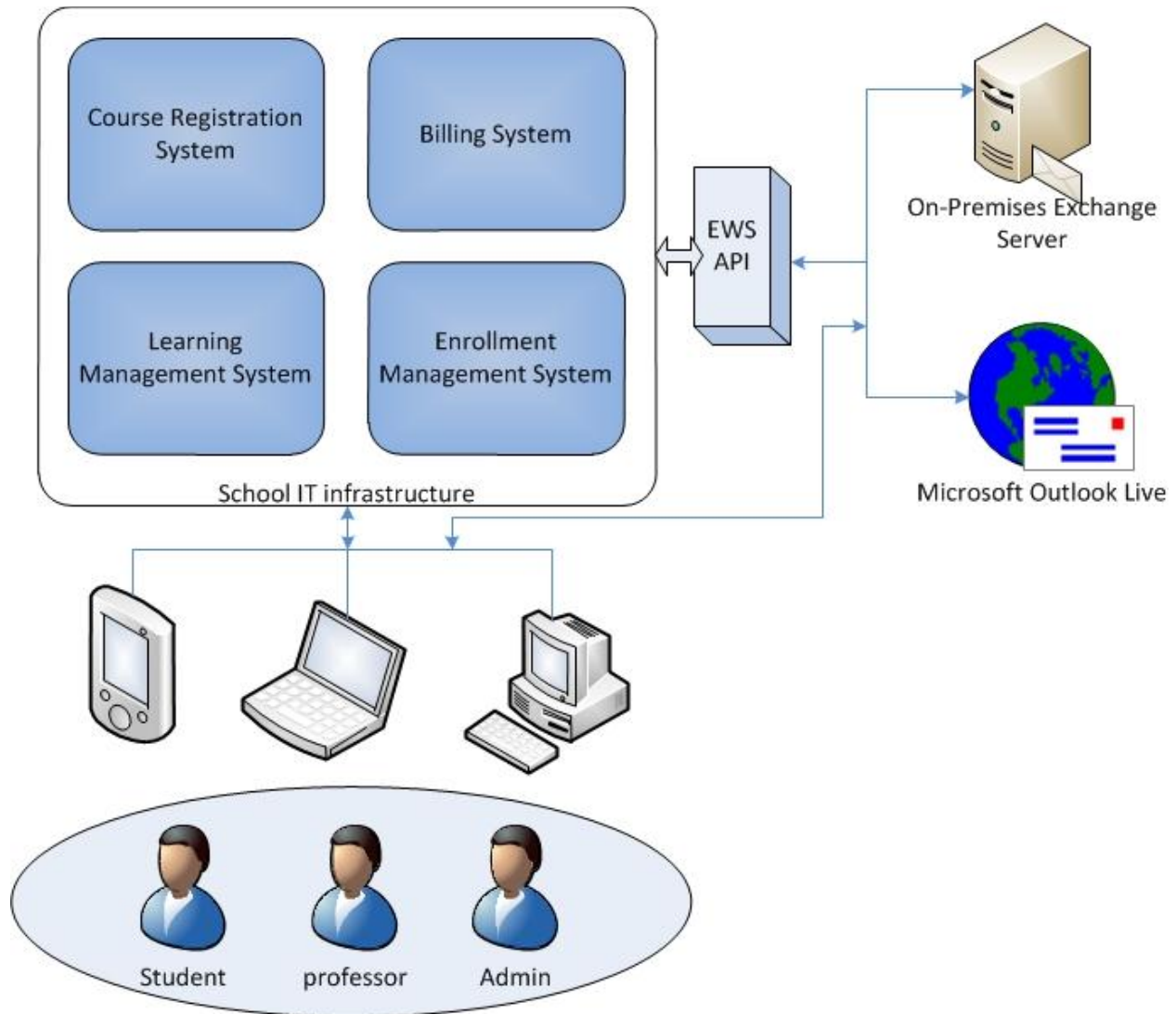


Figure 4 Connected campus using EWS API

In a connected campus environment, the on-premises Exchange Server and Outlook Live can be accessed by users and applications in a seamless fashion. EWS API can be leveraged by school applications to access Exchange resources that exist on the on-premises server as well as Outlook Live. The figure above shows a pictorial representation of how various School Information Systems and users can communicate and access Outlook Live and the on-premises Exchange server.

EWS Managed API Scenarios

In this section we will illustrate different real-world school scenarios and how EWS managed API features can be used. The scenarios are picked to help demonstrate the EWS API's. For installation prerequisites for using EWS Managed API, refer to the Installation section of this document.

Scenario: Course Registration

A very common scenario in most schools is to register students for different courses. A high school administrator may register on behalf of all the students, but in a university, the students register themselves for courses that fit their schedule. When a student enrolls for a particular course, the following EWS API related steps take place:

- A student gets course appointments on to the calendar for the course schedule from professor e-mail. The calendar invite also contains course content and suggested course material.
- The student's e-mail id is added to the course distribution list.
- The student's calendar is checked for events that conflict with the course schedule that the student is registering for and notified accordingly.

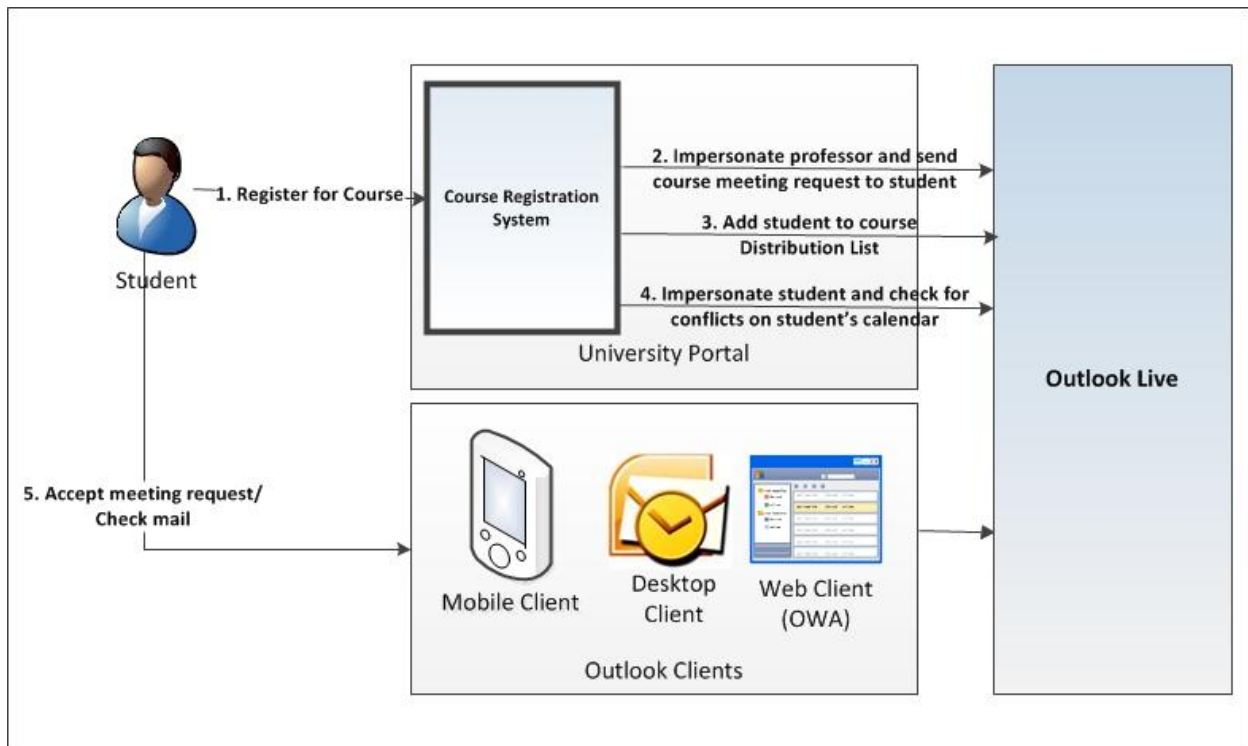


Figure 5 Student Course Registration scenario using EWS API

Send a recurring meeting request to student on behalf of professor

The first step in our scenario helps demonstrate how to automatically send a recurring meeting request. The following steps must be done to automatically send a recurring meeting request to the student for the registered course (on behalf of the professor).

- Create EWS Binding
- Autodiscover EWS URL
- Impersonate professor
- Create and send a meeting request to student on behalf of professor
- Accept the meeting request on behalf of student

The steps are described below in brief with code snippets.

Create EWS Binding

To use EWS Managed API the first step is to create an instance of the *ExchangeService* class as shown in the following code snippet.

```
ExchangeService service = new ExchangeService(ExchangeVersion.Exchange2010);
service.Credentials = new WebCredentials(
    "admin@contosouniversity.com", "password");
service.Url = new Uri("https://sn1prd0202.outlook.com/EWS/Exchange.asmx");
```

The URL is the end point of the EWS service of your domain on Outlook Live. This URL is hard coded in the above code snippet. This service URL can be auto discovered by giving the credentials of a particular e-mail account and Outlook domain name as shown in the step below.

Autodiscover EWS URL

EWS Managed API provides a built-in *Autodiscover* client. The *Autodiscover* service is a key part of the Exchange 2007 and Exchange 2010 architecture. With *Autodiscover*, an application can determine the settings that it should use to communicate with Exchange, such as the URL of Exchange Web Services.

```
ExchangeService service = new ExchangeService(ExchangeVersion.Exchange2010);
service.Credentials = new WebCredentials(
    "admin@contosouniversity.com", "password");
service.AutodiscoverUrl(
    "professor@contosouniversity.com", delegate(string url) { return true;});
```

Impersonate professor

Sending the meeting request on behalf of the professor for the course may not be required, but this presents a very significant opportunity for schools to put the course calendar in the hands of the person running the class. The professor, owning the meeting request, would be able to send any updates to the meeting request if the class is cancelled, or the location and timing changes. This would keep the student updated with any changes to the course class appointment.

To send a meeting request for a course on behalf of a professor, one needs to impersonate the professor. Below is the code snippet to impersonate a professor.

```
ExchangeService service = new ExchangeService(ExchangeVersion.Exchange2010);
service.Credentials = new WebCredentials(
    "admin@contosouniversity.com", "password");
service.AutodiscoverUrl(
    " professor@contosouniversity.com", delegate(string url) { return true;});
service.ImpersonatedUserId = new ImpersonatedUserId(
    ConnectingIdType.SmtpAddress,
    "professor@contosouniversity.com");
```

Impersonating another user is to simply set the property *ImpersonatedUserId* during creation of *ExchangeService* object.

Note: For a user to impersonate another user, “*Application Impersonation*” role should be assigned to the user on the Outlook Live domain. In the above case, admin@contosouniversity.com is given impersonation rights. Hence the application can impersonate the user professor@contosouniversity.com.

Create meeting request

EWS API provides *Appointment* class for creating and sending a meeting request. A meeting request is an appointment with attendees, start time, end time, and location. The following code snippet shows how to create an *Appointment* object and set properties of the appointment like attendees, subject start and end time, and location.

```
Appointment appointment = new Appointment(service);
appointment.Subject = "BI0101: Introduction to Biology";
appointment.Body = new MessageBody();
appointment.Body.Text = "Course Content: Origins of Biology, Cells,
Photosynthesis, Mitosis, Cancer and Genetics \n
Course Material: Introduction to Biology by David.L.Wilson";
appointment.Location = "Aderhold Building- 400-Classroom South";
appointment.RequiredAttendees.Add("student@consotouniversity.com");
```

The recurrence can be daily, weekly, monthly or yearly. A course generally is a weekly recurrence. The following code snippet sets weekly recurrence to every Monday and Wednesday from 10:00 AM to 11:30 AM.

```
appointment.Recurrence = new Recurrence.WeeklyPattern(  
    new DateTime(2010, 1, 15),  
    1,  
    new DayOfWeek[] { DayOfWeek.Monday, DayOfWeek.Wednesday }  
);  
appointment.Recurrence.EndDate = new DateTime(2010, 5, 30);
```

Send the meeting request to the student and save a copy in the professor's Sent Items folder. The following code snippet shows on how to send the meeting request to the student.

```
appointment.save(SendInvitationsMode.SendToAllAndSaveCopy);
```

For a particular course schedule, a new meeting request should be created only once when the first student registers. For subsequent student registrations for the same course, the existing meeting request in professor's calendar should be updated with new student as another attendee. This can be achieved in two different ways.

One way to do it is to bind to an existing meeting request by using its unique identifier. To achieve this, store the unique *AppointmentId* along with the course when the meeting request is created for the first time. Below is the code snippet to bind to an existing meeting request.

```
Appointment appointment = Appointment.Bind(service,  
    new ItemId(appointmentId));  
appointment.RequiredAttendees.Add("newstudent@contosouniversity.com");  
appointment.Update(ConflictResolutionMode.OverWrite,  
    SendInvitationsOrCancellationsMode.SendOnlyToChanged);
```

Another way to do this is to load all appointments in professor's calendar and find the appointment of interest to update it. Below is the code snippet for this approach.

```

// Bind to the calendar folder
CalendarFolder calendarFolder = CalendarFolder.Bind(service,
    WellKnownFolderName.Calendar);
int offset = 0;
FindItemsResults<Item> calendarAppointments;
List<Appointment> appointments = new List<Appointment>();
do
{
    ItemView itemView = new ItemView(100, offset);
    calendarAppointments = calendarFolder.FindItems(itemView);

    // Load the calendar data for the following properties
    PropertySet appointmentProperty = new PropertySet(
        new PropertyDefinitionBase[] {AppointmentSchema.Subject,
            AppointmentSchema.Location}
        );

    // Get the appointments and convert to CalendarData information
    foreach (Appointment appointment in calendarAppointments)
    {
        appointment.Load(appointmentProperty);
        appointments.Add(appointment);
    }

    offset += 100;
} while (calendarAppointments.MoreAvailable);

// Find the appointment of interest
Appointment appointment = appointments.Find((item) =>
    (item.Location == "Aderhold Building- 400-Classroom South" &&
        item.Subject == "BI0101: Introduction to Biology"));
appointment.RequiredAttendees.Add("newstudent@contosouniversity.com");
appointment.Update(ConflictResolutionMode.OverWrite,
    SendInvitationsOrCancellationsMode.SendOnlyToChanged);

```

The method of storing the *AppointmentId* can be used in scenarios where there is need for performance, and lookup of the Outlook calendar need not be done. The latter method can be used in scenarios where storing of additional *AppointmentId* to a file and database is not possible, and the application can afford the lookup time from Outlook Live.

Find conflicts in the students calendar

Finding conflicts with student's calendar is not mandatory step in the course registration scenario. However this section of code is presented as informational and might be useful in any other scenario where it's needed to find conflicts of an appointment in the Calendar.

Once the student is registered, a student can be notified if there are any appointments in students calendar that conflict with the course schedule. Below is the code snippet to achieve this.

```
// Bind to the calendar folder
CalendarFolder calendarFolder = CalendarFolder.Bind(service,
    WellKnownFolderName.Calendar);
int offset = 0;
FindItemsResults<Item> calendarAppointments;
List<Appointment> appointments = new List<Appointment>();
do
{
    ItemView itemView = new ItemView(100, offset);
    calendarAppointments = calendarFolder.FindItems(itemView);
    foreach (Appointment appointment in calendarAppointments)
    {
        appointments.Add(appointment);
    }
    offset += 100;
} while (calendarAppointments.MoreAvailable);

// Find the appointment of interest
Appointment appointment = appointments.Find((item) =>
    (item.Location == "Aderhold Building- 400-Classroom South" &&
    item.Subject == "BI0101: Introduction to Biology"));
PropertySet appointmentProperty = new PropertySet(
    new PropertyDefinitionBase[]
    {AppointmentSchema.ConflictingMeetingCount});
appointment.Load(appointmentProperty);
if (appointment.ConflictingMeetingCount > 0)
{
    return true;
}
else return false;
```

Any conflicting meetings can be checked with the count in *ConflictingMeetingCount* property in *Appointment* object.

Add student to course distribution group

The last step in our scenario is to add the student to the course Distribution Group. A distribution group is useful in organizing all students and teachers in a single group to help promote communication and collaboration with the class. Creation of a distribution group is a prerequisite for this step. It is described in the next section, "Create Distribution Group".

Adding a student to distribution group is an administrative task. PowerShell is leveraged to accomplish this scenario. PowerShell is a task-based command-line shell and scripting language designed especially for system administration. Built on the .NET Framework, Windows PowerShell

helps IT professionals and power users control and automate the administration of the Windows operating system and applications that run on Windows.

A PowerShell session to Outlook Live domain has to be created to run any command. The following code snippet creates and imports a PowerShell session.

```
// Create runspace.
Runspace runspace = System.Management.Automation.RunspaceFactory.CreateRunspace();
runspace.Open();

// Create PowerShell session and import session into runspace.
PowerShell powershell = PowerShell.Create();
powershell.Runspace = runspace;

// Create credential.
string password = "password";
SecureString securePassword = new SecureString();
for (int i = 0; i < password.Length; i++)
    securePassword.AppendChar(password[i]);
PSCredential cred = new PSCredential("admin@contosouniversity.com",
    securePassword);

// Set as variable.
PSCommand command = new PSCommand();
command.AddCommand("Set-Variable");
command.AddParameter("Name", "Cred");
command.AddParameter("Value", cred);
powershell.Commands = command;
powershell.Invoke();
command.Clear();

// Create PowerShell session to Outlook Live.
string newSessionScript = "$Session = New-PSSession"
    + " -ConfigurationName:Microsoft.Exchange -Authentication:Basic"
    + " -ConnectionUri: https://ps.outlook.com/powershell"
    + " -Credential $Cred -AllowRedirection";
command.AddScript(newSessionScript);
powershell.Commands = command;
powershell.Invoke();
command.Clear();

// Import session.
string importSessionScript = "Import-PSSession $Session";
command.AddScript(importSessionScript);
powershell.Commands = command;
powershell.Invoke();
command.Clear();
```

After creating and importing a session to the Outlook Live domain, you can add the student to the course Distribution List as shown in the below code snippet.

```
string addToDLScript = "Add-DistributionGroupMember"
    + " -Identity 'BI0101-IntroductiontoBiology'"
    + " -Member 'student@contosouniversity.com'";
command.AddScript(addToDLScript);
powershell.Commands = command;
powershell.Invoke();
```

Create a distribution group

At the start of the quarter or semester, a School or University admin would want to create distribution groups for each course. This can be achieved with the following code snippet after creating and importing PowerShell session to Outlook Live domain as explained in section “Adding student to course Distribution Group”.

```
string createDLScript = New-DistributionGroup
    + " -Name 'BI0101-IntroductiontoBiology'"
    + " -DisplayName 'CSC8711-DatabasesAndWeb'";
command.AddScript(createDLScript);
powershell.Commands = command;
powershell.Invoke();
```

It would be easy to develop a service that will loop through all the courses and create distribution groups for each course as shown in the above code snippet.

Scenario: Publishing school events to student’s calendar

In this scenario, important school events are published to all students’ calendars. An event sitting on a student’s personal calendar would have more chance of reminding the student of the event. As the academic quarter or semester progresses and the student gets busy with all the academic work, it is very likely that students may not visit the Web site where the school might publish all events. Some examples of the events are holidays, semester breaks, and semester start dates.

This scenario is enabled by sending a calendar invite to all the students. The calendar invite can be sent from an admin user using Impersonation.

Event information has attributes like name of the event, description, location of the event, and schedule of the event. In our scenario the events are read from an xml file. The below figure shows a sample xml file with event information

```
- <CalendarEvents xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://microsoft/liveatdudemo/2009/">
- <calendarData>
- <Body>
  <BodyType i:nil="true" />
  <Text i:nil="true" />
</Body>
<End>2010-01-01T17:00:00</End>
<Location i:nil="true" />
<OptionalAttendees xmlns:d3p1="http://schemas.microsoft.com/2003/10/Serialization/Arrays" />
<RecurrenceAppointment i:nil="true" />
<RequiredAttendees xmlns:d3p1="http://schemas.microsoft.com/2003/10/Serialization/Arrays" />
<Start>2010-01-01T08:00:00</Start>
<Subject>New Year Eve</Subject>
</calendarData>
+ <calendarData>
</CalendarEvents>
```

Figure 6 Events xml file content

An event object used in the code snippet below is a data structure which contains the information read from above xml file. This data can be read from a database or any other data source in real world apps. The following code snippet shows how to impersonate each user and create an event in their calendar.

```
ExchangeService service = new ExchangeService(ExchangeVersion.Exchange2010);
service.Credentials = new WebCredentials(
    "admin@contosouniversity.com", "password");
service.AutodiscoverUrl(
    "professor@contosouniversity.com", delegate(string url) { return true;});

foreach (User user in users)
{
    service.ImpersonatedUserId = new ImpersonatedUserId(
        ConnectingIdType.SmtpAddress,
        user.WindowsLiveId);
    foreach (Event event in events)
    {
        Appointment appointment = new Appointment(service);
        appointment.Subject = event.Subject;
        appointment.Body = new MessageBody();
        appointment.Body.Text = event.Description;
        appointment.Start = event.Start;
        appointment.End = event.End;
        appointment.Save();
    }
}
```

User and *Event* are custom objects and not part of EWS managed API. Also note that the appointment with no attendees is just an appointment or event.

Scenario: Campus alerts e-mail

A common scenario with academic institutions is the need for the school to send out campus alerts. These alerts could come in the form of a campus closure due to weather, or a welcome message to new students. To send an e-mail, instantiate E-mailMessage object. The following code snippet shows how to send an e-mail.

```
E-mailMessage message = new E-mailMessage(service);

//Add recipients
E-mailAddress e-mailAddress = new E-mailAddress();
e-mailAddress.Name = "Student";
e-mailAddress.Address = "student@contosouniversity.com";

message.ToRecipients.Add(e-mailAddress);
message.Subject = "Welcome to Contoso University";
message.Body = new MessageBody("A warm welcome to Contoso University.
    All the best for your course!");
message.SendAndSaveCopy();
```

In the above code snippet e-mail message *Subject* and *Body* are hard coded. This information can be driven from an xml file, database or any other data source.

Scenario: Displaying mail and calendar data on student's portal

Some schools want to provide students with a central Web portal where they can access campus information, and have it tailored to the student's personal information. In this scenario, examples are provided showing how student e-mails and calendar appointments can be displayed on a student's personal home page in the school Web portal. This scenario displays the last five e-mails in student's inbox and events for the day from student's calendar.

Note: In the below code snippets the user credentials refer to Outlook Live credentials. In case of university portal it is possible that the student's credentials are different. It is assumed that the credentials are mapped by using a DB or lookup to a service which has the mapping stored. It is expected to send the Outlook credentials to EWS API.

Below are the code snippets to fetch mail and calendar items and access necessary information to display in the portal.

Fetch mail items

This scenario demonstrates accessing the Outlook Live folders. By now it's obvious that the first steps are to initialize *ExchangeService*, *Autodiscover* and then *Impersonate* the student. We have already shown this in previous scenarios in this document. Below is the code snippet for reference.

```
ExchangeService service = new ExchangeService(ExchangeVersion.Exchange2010);
service.Credentials = new WebCredentials(
    "admin@contosouniversity.com", "password");
service.AutodiscoverUrl(
    "student@contosouniversity.com", delegate(string url) { return true;});
service.ImpersonatedUserId = new ImpersonatedUserId(
    ConnectingIdType.SmtpAddress,
    "student@contosouniversity.net");
```

Folder object is used to retrieve mail items from the Inbox. Bind this object to Inbox folder. Enum *WellKnownFolderName* has list of all well-known folders. Inbox is a well-known folder. *ItemView* object is used to specify number of items to retrieve. Input value to this object is 5 since the first five e-mails have to be fetched.

```
Folder inbox = Folder.Bind(service, WellKnownFolderName.Inbox);
// Fetch only first five messages from Inbox.
FindItemsResults<Item> messages = inbox.FindItems(new ItemView(5));
```

Now we have access to first five e-mail items. Iterate through these mail items to get information (like subject of mail, date sent,) to render in the UI. These mail items are of type *E-mailMessage*. Following is the code snippet to iterate through mail items and fetch information.

```
foreach (E-mailMessage msg in messages)
{
    DateTime dateTimeSent = msg.DateTimeSent;
    String e-mailFrom = msg.From.Name;
    String e-mailSubject = msg.Subject;
    String openItemUri = "https://" + service.Url.Host + "/owa/"
        + msg.WebClientReadFormQueryString;
}
```

For a sample SharePoint 2007 mail and calendar Web parts, refer to the link <http://code.msdn.microsoft.com/SharePointWebPartEWS>.

Fetch Calendar items

CalendarFolder object is used to fetch calendar items. *CalendarView* object is used to specify the date range between which events are to be fetched. Following code snippet fetched all events for the current day.

```
CalendarFolder calendar = CalendarFolder.Bind(service,
    WellKnownFolderName.Calendar);
FindItemsResults<Appointment> appointments = calendar.FindAppointments(
    new CalendarView(DateTime.Today, DateTime.Today.AddDays(1)));
```

Now we have access to all the events for the current day. Iterate through these appointments to get information (like subject of the event, location, and start time) to render in the UI. Calendar items are of type *Appointment*.

```
foreach (Appointment appointment in appointments)
{
    String subject = appointment.Subject;
    String location = appointment.Location;
    DateTime startTime = appointment.Start;
    DateTime endTime = appointment.End;
    Bool isAllDayEvent = appointment.IsAllDayEvent;
    String openUrl = "https://" + service.Url.Host + "/owa/" +
        appointment.WebClientReadFormQueryString;
}
```

Installation

This section describes the necessary software to be installed on the machine in order to use EWS API and PowerShell to access Outlook Live resources. The following software need to be installed:

- EWS Managed API 1.0 – Download and install EWS Managed API from Microsoft Download center. Link to the download is present in the References section.
- Windows PowerShell – Before you can use Windows PowerShell with Outlook Live, make sure you have the correct versions of Windows PowerShell and Windows Remote Management (WinRM) installed and configured on your computer. Outlook Live requires the Windows Management Framework, which contains the correct versions of Windows PowerShell v2 and WinRM 2.0. Link to the download is present in the References section.

Note: If your computer is running Windows 7 or Windows Server 2008 R2, you don't have to install Windows Power Shell since it is already installed.

Summary

Using the Exchange Web Services Managed API code snippets discussed in this document, schools can work to better integrate Outlook Live Exchange resources into their school information systems, thereby enabling a connected campus.

Glossary of Acronyms

EWS – Exchange Web Services
API – Application Programming Interface
HRM – Human Resource Management
ERP – Enterprise Resource Planning

References

- Exchange Web Services overview
 - <http://msdn.microsoft.com/en-us/library/dd877045.aspx>
- EWS Managed API overview
 - [http://msdn.microsoft.com/en-us/library/dd633709\(EXCHG.80\).aspx](http://msdn.microsoft.com/en-us/library/dd633709(EXCHG.80).aspx)
- EWS Managed API download
 - <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=c3342fb3-fbcc-4127-becf-872c746840e1>
- Install and Configure Windows PowerShell
 - <http://help.outlook.com/en-us/140/cc952756.aspx?ref=search>
- SharePoint Web Parts
 - <http://code.msdn.microsoft.com/SharePointWebPartEWS>