Nintex Forms: API for SharePoint O365

Last updated Friday, March 4, 2016

Contents

Nintex Forms for Office 365 REST API

Welcome to the Nintex Forms for Office 365 REST API.

Use the left hand side navigation to browse the various help topics. You can also use the Search box in the top right hand corner to search for any key words.

- If you're eager to dive right into the REST API, the "Nintex Forms O365 Quick Start" on page 1 provides a short overview of what you need to get started.
- The "Guide" on page 3 provides practical information about using the REST API, with examples that you can use in your applications.
- The API Reference contains an overview of the technical information you'll need to use the REST API, as well as detailed reference information about the REST resources, data types, request and response bodies, and other elements included with the REST API.

If you have specific queries or questions, you may also want to try Nintex Connect, at http://connect.nintex.com, which is our vibrant community of Nintex customers and partners as well as a collection of blogs and additional material to help resolve problems and use the Nintex platform to its fullest.

If you require support, please contact support@nintex.com

Nintex Forms O365 Quick Start

You can quickly and easily export, import, migrate, save, and publish forms from Nintex Forms for Office 365 by using the NintexForms for Office 365 REST API. The quick start provided here demonstrates the REST API, by walking you through the steps needed to export a Nintex form from SharePoint Online.

To use the API, establish your credentials and the endpoint. Use a REST client to contact the endpoints. The following quick start walks you through the following steps:

1. Get the items you need before you work with the API.
2. Get your authentication cookie from SharePoint.
    A. Get your authentication cookie from SharePoint.
    B. Collect your header information.
    C. Construct a URL to reach the target endpoint.
        - Export Form
        - Import Form
        - Save Form
        - Publish Form
        - Delete Form

1. Get the items you need before you work with the API

Before you can begin you will need:

- An account on SharePoint O365 with site administrator privileges. You will need these credentials in creating a token to access the API.
- Nintex Forms for Office 365 and Workflow for Office 365 installed and activated on the O365 site where you plan on hosting the forms.
- An understanding of REST. The API uses RESTful endpoints. For more information about REST see "RESTful Web services: The basics."
- You will need the Nintex API Key and an API URL from the administrator of the SharePoint O365 tenancy. Note the API URL is different than the URL for the targeted SharePoint O365 tenancy.
- You may also need specific headers used in the REST call required by the tenancy; this will also be provided by the administrator of the SharePoint O365 tenancy.
- The ID of the list from where you plan on exporting or importing the Nintex Forms.

2. Working with the REST services

The following section will look at using the REST service using HTTP calls in order to orient you to use the API. You must first get your authentication cookie from the SharePoint 0365 tenancy that you plan on working with. You can retrieve the authentication cookie via HTTP, with a .Net REST client, or by using Windows PowerShell.

A. Get your authentication cookie from SharePoint

Send the following information to the SharePoint to get your authentication cookie:

- Username
- Password
- Site URL

The cookie will have the following format: `Cookie, Site URL, SPOIDCRL = cookiestring`

You can retrieve your authentication cookie via REST in a workflow using HTTP. For more information see "Getting Your SharePoint 0365 Credentials via REST" on page 41.

With a .NET REST Client

You can find detailed code samples for C# in the "Guide" on the next page section which include five samples using Microsoft's HTTPClient library. The following code snippet highlights how these clients retrieve the authorization cookie:

```
  var login = "Username";

  var password = "password";

  var siteUrl = "siteURL";

  var creds = new SharePointOnlineCredentials(login, password);

  var auth = creds.AuthenticateAsync(new Uri(siteUrl), true);

  var request = (HttpWebRequest)WebRequest.Create(siteUrl);

  request.CookieContainer = auth.Result.CookieContainer;

var result = (HttpWebResponse)request.GetResponse();
```

With Windows PowerShell

You can find a Windows PowerShell script that demonstrates how to use the **GetAuthenticationCookie** method to retrieve an authentication cookie for the SharePoint 0365 tenancy using the credentials. For more information see "Windows PowerShell" in "Authentication and authorization" on page 38.

B. Collect your header information

You will need the appropriate header information:

- Authorization

  Using the authorization cookie from Step A.
- API-Key

  Using the API Key from your site administrator.

C. Construct a URL to reach the target endpoint

The URL uses the following format:
domainofo365tenant/api/v1/forms/*listid*, i.e.,
https://crestan.com//api/v1/forms/6263627c-57a6-42d0-9c87-2232e4e1899d

Headers:

- Authorization header. The cookie will have the following format: `Cookie, Site URL, SPOIDCRL = cookiestring`.
- API-Key. The key will be 22 characters long.

The API supports the following methods:

- Export Form
  Call example:
  GET https://crestan.com/api/v1/forms/6263627c-57a6-42d0-9c87-2232e4e1899d with the headers. The call will save a binary file containing the Nintex Form data (NFForm.nfp). For details see the reference topic at Get Form.
- Import Form
  Call example:
  POST https://crestan.com/api/v1/forms/6263627c-57a6-42d0-9c87-2232e4e1899d with the headers. Include the payload of the Nintex Form. For details see the reference topic at Import Form.
- Save Form
  Call example:
  PUT https://crestan.com/api/v1/forms/6263627c-57a6-42d0-9c87-2232e4e1899d. You will receive a JSON payload containing a _link that holds a relative link to the publish end point (step 7). For details see the reference topic at Save Form.
- Publish Form
  Call example:
  POST https://crestan.com/api/v1/forms/6263627c-57a6-42d0-9c87-2232e4e1899d/publish with the headers. For details see the reference topic at Publish Form.
- Delete Form
  Call example:
  DELETE https://crestan.com/api/v1/forms/6263627c-57a6-42d0-9c87-2232e4e1899d to remove a form from a list. For details see the reference topic at Delete Form.

Next steps

The "Guide" below provides practical information and examples for each REST resource and operation included with the REST API. Experiment with the examples provided in the Guide to get a better idea about how to use the REST API.

The API Reference contains an overview of the technical information you'll need to use the REST API, as well as detailed reference information about the REST resources, data types, request and response bodies, and other elements included with the REST API. The API Reference is the best place to answer technical questions about implementing the REST API

Related information

"Guide" below

API Reference

Guide

This guide takes you through the REST resources and operations included with the Nintex Forms for Office 365 REST API, providing examples that you can use in Visual Studio 2013.

What is REST?

Representational State Transfer (REST) is a methodology for implementing scalable web services. REST emphasizes a stateless, cacheable implementation with a uniform interface, typically accessed using Hypertext Transfer Protocol (HTTP). The REST service decouples the client from the server, by providing access to REST resources, each of which is accessed by using a Universal Resource Identifier (URI) that identifies that particular REST resource.

A REST resource is typically accessed by using HTTP methods, such as GET and PUT, to send and receive self-descriptive messages between the client and the REST service. These messages are typically expressed using Hypertext Markup Language (HTML), Expressive Markup Language (XML), or JavaScript Object Notation (JSON). The messages are self-descriptive in that each message includes enough information to describe

how to process the message, and contains representations of information with which the client or the REST resource can interact. For example, the Nintex Forms for Office 365 REST API does not connect a client directly to Nintex Forms for Office 365, but instead serves as an intermediary between the two, interacting with representations of information, such as forms and export files, exchanged in such messages.

The combination of a REST resource and an HTTP method typically identifies a specific operation that can be performed for the information represented by that REST resource. Multiple HTTP methods are often used with a particular REST resource to represent different operations that can be performed for the information represented by that REST resource.

For example, to export an existing form from Nintex Forms for Office 365to a client for a fictional customer named Crestan, the following REST resource is invoked, using the HTTP GET method:

```
https://crestan.nintexo365.com/api/v1/forms/fd4f1cd2-7ea7-4b62-9751-0ff83ab609f7
```

However, invoking the very same REST resource and using the HTTP POST method allows the client to import a form file, overwriting the form inNintex Forms for Office 365 for the list. HTTP methods allow REST to avoid ambiguity when interacting with REST resources.

What can I do with the REST API?

The REST API includes the following REST resources:

- Exporting and importing forms

  This REST resource manages interaction with export files for Nintex Forms for Office 365. You'll use this resource to perform the following operations:

  - Exporting a form
    You can use this operation to export an existing form as an NFP file which can be used inNintex Forms for Office 365, in either the Forms designer or the REST API, to import or migrate forms to other SharePoint lists or sites.
  - Importing a form
    This operation creates a new form from an NFP file. If the export file contains a list form, you can optionally migrate the form to a different SharePoint list, allowing Nintex Forms for Office 365 to update the metadata from the form to match the new destination.

- Saving, publishing, and deleting forms

  This REST resource manages interaction with forms for Nintex Forms for Office 365. You'll use this resource to perform the following operations:

  - Saving a form
    You can use this operation to overwrite an existing list form from an export file.
  - Publishing a forms
    Use this operation to publish an unpublished form on Nintex Forms for Office 365.
  - Deleting a form
    Use this operation to delete a form in Nintex Forms for Office 365.

Related information

"Nintex Forms O365 Quick Start" on page 1

API Reference

Exporting a form

You can use the Nintex Forms for Office 365 REST API to retrieve a form in a SharePoint list as a Nintex Forms for Office 365 export (.nfp) file.

The Nintex Forms Package (NFP) file is a ZIP file with the extension .NFP that contains the configuration settings, variables, lists, content types and columns for a Nintex form. The file is backwards compatible for with the legacy Form.XML files exported from previous version of Nintex Forms designer.

The export file can then be used for a variety of other operations in the REST API, such as migrating an exported list form to a different list on a different SharePoint site, or you can import the export file right into the Forms designer in Nintex Forms for Office 365.

For more information about the REST resource used to export list form, see Get Form.

Considerations

To avoid surprises while trying to export a form, take the following points into consideration:

- Ensure that the form exists, and that you're allowed to access it, before trying to export it.
- Ensure that your SharePoint authentication cookie hasn't expired.

  This example avoids that issue by getting an authentication cookie from SharePoint every time you run the example, but you can cache an authentication cookie and avoid a round trip to SharePoint as long as the authentication cookie hasn't expired.
- Don't modify the contents of the export file.

  The export file is a binary file that uses a format specific to Nintex Forms for Office 365. Manipulating the contents can cause unpredictable outcomes and invalidate the format of the export file, making it unusable.

Prerequisites

To use the example, ensure that you have the following prerequisites:

- Access to a SharePoint Online site, with Manage Web permissions for the site.
- A list form that you can export from the SharePoint Online site.
- Access to the Nintex Forms for Office 365 REST API, and an API key with which to authorize the REST API.
- Access to any version of Visual Studio 2013.

Code

You can download the code used in this example from the following location:

O365_RESTAPI_NF_Export.zip

Note: You need to configure the code for the example before you can build and run it. See step 4 in the following example for instructions about how to configure the code for the example.

Example

The following example describes how to create a Visual Studio 2013 project to export a list form from your SharePoint Online site, by using the REST API to get the list form definition from the site and then saving it to an export (.nfp) file.

1. Create a new Visual Studio 2013 project, using the Console Application template for Visual C#.

2. Add the following references to your new Visual Studio project.
    - Microsoft.SharePoint.Client
    - Microsoft.SharePoint.Client.Runtime
    - System.Net
    - System.Net.Http
3. In your new Visual Studio project, paste the following code into the file named Program.cs, overwriting the existing **using** statements at the beginning of the file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.SharePoint.Client;
using System.Security;
using System.Net.Http.Headers;
using System.Net.Http;
using System.Net;
using System.IO;
```

4. Paste the following code into Program.cs, overwriting the **Main()** static method already included in the **Program** static class, and then configure the code as described in the comments:

```
        // The API key and root URL for the REST API.
        // TODO: Replace with your API key and root URL.
        static private string apiKey = "";
        static private string apiRootUrl = "https://t-
stapigcsvwus01.nintexo365test.com";


        // The SharePoint site and credentials to use with the REST API.
        // TODO: Replace with your site URL, user name, and password.
        static private string spSiteUrl = "https://n-
intexinteraction01.sharepoint.com";
    static private string spUsername = "";
    static private string spPassword = "";
    // The list ID of the form to export, and the file path in which to
create
    // the export file.
    // TODO: Replace with your form identifier and the file path
    // in which to create your export file.
    static private string listId = "";
    static private string exportPath = "";


    /// <summary>
   /// The main entry point for the application.
   /// </summary>
   [STAThread]
   static void Main()
   {
        // Export the form to an export file.
        ExportFormToFile();
   }
```

All the configuration you need to do for this example happens here, and the code provided in subsequent steps uses these variables to get an authentication cookie from SharePoint and then export your list form from your SharePoint site.

5. Paste the following code into Program.cs, immediately after the **Main** static method in the **Program** static class:

```
static private string GetSPOCookie()
     {
           // If successful, this variable contains an authentication
cookie;
           // otherwise, an empty string.
           string result = String.Empty;
           try
           {
                // Construct a secure string from the provided password.
                // NOTE: For sample purposes only.
                var securePassword = new SecureString();
                foreach (char c in spPassword) { securePassword.AppendChar
(c); }
                // Instantiate a new SharePointOnlineCredentials object,
using the
                // specified username and password.
                var spoCredential = new SharePointOnlineCredentials(spUser-
name, securePassword);
                // If successful, try to authenticate the credentials for
the
                // specified site.
                if (spoCredential == null)
               {
                     // Credentials could not be created.
                     result = String.Empty;
               }
               else
               {
                     // Credentials exist, so attempt to get the authen-
tication cookie
                     // from the specified site.
                     result = spoCredential.GetAuthenticationCookie(new Uri
(spSiteUrl));
               }
          }
          catch (Exception ex)
          {
                // An exception occurred while either creating the cre-
dentials or
                // getting an authentication cookie from the specified site.
                Console.WriteLine(ex.ToString());
                result = String.Empty;
          }
```

```
            // Return the result.
            return result;
        }
```

The **GetSPOCookie** static method uses the SharePoint site and credentials that you configured in step 4 to get an authentication cookie from SharePoint.

6. Paste the following code into Program.cs, immediately after the code you pasted in the previous step:

```
static async private void ExportFormToFile()
     {
          // Create a new HTTP client and configure its base address.
          HttpClient client = new HttpClient();
          client.BaseAddress = new Uri(spSiteUrl);

          // Add common request headers for the REST API to the HTTP client
          client.DefaultRequestHeaders.Add("Api-Key", apiKey);
          // Get the SharePoint authorization cookie to be used by the HTTP
client
          // for the request, and use it for the Authorization request
header.
          string spoCookie = GetSPOCookie();
          if (spoCookie != String.Empty)
          {
               var authHeader = new AuthenticationHeaderValue(
                    "cookie",
                    String.Format("{0} {1}", spSiteUrl, spoCookie)
               );
               // Add the defined authentication header to the HTTP client's
               // default request headers.
               client.DefaultRequestHeaders.Authorization = authHeader;
          }
          else
          {
               throw new InvalidOperationException("Cannot define Authentic-
ation header for request.");
          }
          Console.Write("{0}/api/v1/forms/{1}\n",
               apiRootUrl.TrimEnd('/'),
               Uri.EscapeUriString(listId));

          // If we're at this point, we're ready to make our request.
          // Note that we're making this call synchronously - you can call
the REST API
          // asynchronously, as needed.
          var exportFormUri = String.Format("{0}/api/v1/forms/{1}",
               apiRootUrl.TrimEnd('/'),
               Uri.EscapeUriString(listId));
          HttpResponseMessage response = client.GetAsync(exportFormUri).Res-
ult;
          Console.Write("And: {0}", response);
```

```
            // If we're successful, write an export file from the body of the
response.
            if (response.IsSuccessStatusCode)
            {
                // Concatenate the export file name from the response head-
ers.
                string exportFilePath = String.Empty;
                if (response.Content.Headers.ContentDisposition.FileName !=
null)
                {
                    // Get the suggested file name from the Content-Dis-
position header.
                    exportFilePath = Path.Combine(exportPath,
                    response.Content.Headers.ContentDisposition.FileName.Tri-
m('"'));
                }
                else
                {
                    // Use a default file name if the suggested file name
couldn't be retrieved.
                    exportFilePath = Path.Combine(exportPath,
"DefaultForm.nfp");
                }

                // The response body contains a Base64-encoded binary string,
which we'll
                // asynchronously retrieve and then write to a new export
file.
                byte[] exportFileContent = await response.Con-
tent.ReadAsByteArrayAsync();
                System.IO.File.WriteAllBytes(exportFilePath, exportFileCon-
tent);
            }
        }
```

The **ExportFormToFile** static method uses an HTTP client to invoke the REST resource provided by the REST API to export your list form from your SharePoint site. The client's default request headers are configured, the **GetAsync** method is used to invoke the REST resource, and, if successful, the response is written to an export file.

7. Build and run the Visual Studio project.

   If you've configured the variables provided in step 4 appropriately, running the project produces an export file in the specified file path for the list you specified from your SharePoint site. The export file's name is typically determined by the **Content-Disposition** content header included in the response.

Related Information

Get Form

"Guide" on page 3

Importing a form

You can use the Nintex Forms for Office 365 REST API to import the contents of a Nintex Forms for Office 365 export (.nfp) file into a SharePoint list. This process will import, save, and publish your form where it will be available to users of the SharePoint site.

For more information about the REST resource used to export list form, see Import Form.

Considerations

To avoid surprises while trying to import into a new form, take the following points into consideration:

- Ensure that your SharePoint authentication cookie hasn't expired.

  This example avoids that issue by getting an authentication cookie from SharePoint every time you run the example, but you can cache an authentication cookie and avoid a round trip to SharePoint as long as the authentication cookie hasn't expired.
- Don't modify the contents of the export file.

  The export file is a binary file that uses a format specific to Nintex Forms for Office 365. Manipulating the contents can cause unpredictable outcomes and invalidate the format of the export file, making it unusable.

Prerequisites

To use the example, ensure that you have the following prerequisites:

- Ensure that your SharePoint authentication cookie hasn't expired.

  This example avoids that issue by getting an authentication cookie from SharePoint every time you run the example, but you can cache an authentication cookie and avoid a round trip to SharePoint as long as the authentication cookie hasn't expired.
- Don't modify the contents of the export file.

  The export file is a binary file that uses a format specific to Nintex Forms for Office 365. Manipulating the contents can cause unpredictable outcomes and invalidate the format of the export file, making it unusable.

Code

You can download the code used in this example from the following location: O365_RESTAPI_NF_Import.zip

Note: You need to configure the code for the example before you can build and run it. See step 4 in the following example for instructions about how to configure the code for the example.

Example

The following example describes how to create a Visual Studio 2013 project to import a list form to your SharePoint Online site. In this sample you will use the REST API to post the form definition into a SharePoint list on your site.

1. Create a new Visual Studio 2013 project, using the Console Application template for Visual C#.

2. Add the following references to your new Visual Studio project.
    - Microsoft.SharePoint.Client
    - Microsoft.SharePoint.Client.Runtime
    - System.Net
    - System.Net.Http
3. In your new Visual Studio project, paste the following code into the file named Program.cs, overwriting the existing **using** statements at the beginning of the file:

```
using System;

using System.Text;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.SharePoint.Client;

using System.Security;

using System.Net.Http.Headers;

using System.Net.Http;

using System.Net;

using System.IO;
```

4. Paste the following code into Program.cs, overwriting the **Main()** static method already included in the **Program** static class, and then configure the code as described in the comments:

```
        // The API key and root URL for the REST API.
        // TODO: Replace with your API key and root URL.
        static private string apiKey = "";
        static private string apiRootUrl = "";

        // The SharePoint site and credentials to use with the REST API.
        // TODO: Replace with your site URL, user name, and password.
        static private string spSiteUrl = "";
        static private string spUsername = "";
        static private string spPassword = "";

        // The list form to export, and the name of the destination list for
which
        // the new form is to be imported.
        // TODO: Replace with the path to your form package (NFP or XML)
        // and target list title.
        static private string importFileName = "";
        static private string listId = "";

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // Copy a list form from a local form file to a destination
list.
            CopyFormToList();
            Console.Write("Press a key to close the window.");
            Console.ReadKey();
        }
```

All the configuration you need to do for this example happens here, and the code provided in subsequent steps uses these variables to get an authentication cookie from SharePoint, and then import into a new list form for a SharePoint list on your site.

5. Paste the following code into **Program.cs**, immediately after the **Main** static method in the **Program** static class:

```
        static private string GetSPOCookie()
        {
            // If successful, this variable contains an authentication
cookie;
            // otherwise, an empty string.
            string result = String.Empty;
            try
            {
                // Construct a secure string from the provided password.
                // NOTE: For sample purposes only.
                var securePassword = new SecureString();
                foreach (char c in spPassword) { securePassword.AppendChar
(c); }

                // Instantiate a new SharePointOnlineCredentials object,
using the
                // specified username and password.
                var spoCredential = new SharePointOnlineCredentials(spUser-
name, securePassword);
                // If successful, try to authenticate the credentials for
the
                // specified site.
                if (spoCredential == null)
                {
                    // Credentials could not be created.
                    result = String.Empty;
                }
                else
                {
                    // Credentials exist, so attempt to get the authen-
tication cookie
                    // from the specified site.
                    result = spoCredential.GetAuthenticationCookie(new Uri
(spSiteUrl));
                }
            }
            catch (Exception ex)
            {
                // An exception occurred while either creating the cre-
dentials or
                // getting an authentication cookie from the specified site.
                Console.WriteLine(ex.ToString());
                result = String.Empty;
            }
```

```
            // Return the result.
            return result;
        }
```

6. The **GetSPOCookie** static method uses the SharePoint site and credentials that you configured in step 4 to get an authentication cookie from SharePoint.

7. Paste the following code into Program.cs, immediately after the code you pasted in the previous step:

```
        static private void CopyFormToList()
        {
            // Create a new HTTP client and configure its base address.
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(spSiteUrl);
            // Add common request headers for the REST API to the HTTP cli-
ent.
            client.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/json"));
            client.DefaultRequestHeaders.Add("Api-Key", apiKey);
            // Get the SharePoint authentication cookie to be used by the
HTTP client
            // for the request, and use it for the Authorization request
header.
            string spoCookie = GetSPOCookie();
            if (spoCookie != String.Empty)
            {
                var authHeader = new AuthenticationHeaderValue(
                    "cookie",
                    String.Format("{0} {1}", spSiteUrl, spoCookie)
                );
                // Add the defined Authorization header to the HTTP client's
                // default request headers.
                client.DefaultRequestHeaders.Authorization = authHeader;
            }
            else
            {
                throw new InvalidOperationException("Cannot define Author-
ization header for request.");
            }

            // Read the contents of our form into a byte array, so that we
can send the
            // contents as a ByteArrayContent object with the request.
            if (System.IO.File.Exists(importFileName))
            {
                // Read the file.
                byte[] exportFileContents = System.IO.File.ReadAllBytes
(importFileName);
                ByteArrayContent saveContent = new ByteArrayContent
(exportFileContents);

                // If we're at this point, we're ready to make our request.
```

```
                // Note that we're making this call synchronously - you can
call the REST API
                // asynchronously, as needed.
                var importFormUri = String.Format("{0}/api/v1/forms/{1}",
                    apiRootUrl.TrimEnd('/'),
                    Uri.EscapeUriString(listId));
                HttpResponseMessage saveResponse = client.PutAsync
(importFormUri, saveContent).Result;

                if (saveResponse.IsSuccessStatusCode)
                {
                    Console.WriteLine("Successfully imported form.");
                }
                else
                {
                    Console.WriteLine("Failed to import form.");
                }
            }
        }
```

The **ImportForm** static method uses an HTTP client to invoke the REST resource provided by the REST API to first import your list form from a local file.

Then, the same HTTP client is used to import the contents of the file into a new list form for the specified SharePoint list. A **ByteArrayContent** object is used to encapsulate the binary contents of the export file, and the **PutAsync** method is used to invoke the REST resource.

8. Build and run the Visual Studio project.

   If you've configured the variables provided in step 4 appropriately, running the project produces a copy of the specified list form associated with the specified SharePoint list on your SharePoint site.

> **Note:** The form name does not change. If you already have a list form for the specified SharePoint list with a name that matches the name of the imported form, an error occurs.

Related Information

Get Form

"Guide" on page 3

Saving a form

You can use the Nintex Forms for Office 365 REST API to save the contents of a Nintex Forms for Office 365 export (.nfp) file into an existing SharePoint list. This sample will allow you to post your form as a draft, and return a JSON payload with a publish relative URI. A published form can be accessed by users of the SharePoint site.

For more information about the REST resource used to save into a form, see Save Form.

Considerations

To avoid surprises while trying to save a form, take the following points into consideration:

- Ensure that your SharePoint authentication cookie hasn't expired.

  This example avoids that issue by getting an authentication cookie from SharePoint every time you run the example, but you can cache an authentication cookie and avoid a round trip to SharePoint as long as the authentication cookie hasn't expired.
- Don't modify the contents of the export file.

  The export file is a binary file that uses a format specific to Nintex Forms for Office 365. Manipulating the contents can cause unpredictable outcomes and invalidate the format of the export file, making it unusable.

Prerequisites

To use the example, ensure that you have the following prerequisites:

- Access to a SharePoint Online site, with Manage Web permissions for the site.
- A list form that you can export from the SharePoint Online site.
- Access to the Nintex Forms for Office 365 REST API, and an API key with which to authorize the REST API.
- Access to any version of Visual Studio 2013.

Code

You can download the code used in this example from the following location: O365_RESTAPI_NF_Save.zip

Note: You need to configure the code for the example before you can build and run it. See step 4 in the following example for instructions about how to configure the code for the example.

Example

The following example describes how to create a Visual Studio 2013 project to update an existing form from an export file previously exported from that Nintex Forms.

1. Create a new Visual Studio 2013 project, using the Console Application template for Visual C#.
2. Add the following references to your new Visual Studio project.
   - Microsoft.SharePoint.Client
   - Microsoft.SharePoint.Client.Runtime
   - System.Net
   - System.Net.Http

3. In your new Visual Studio project, paste the following code into the file named Program.cs, overwriting the existing **using** statements at the beginning of the file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.SharePoint.Client;
using System.Security;
using System.Net.Http.Headers;
using System.Net.Http;
using System.Net;
using System.IO;
```

4. Paste the following code into Program.cs, overwriting the **Main()** static method already included in the **Program** static class, and then configure the code as described in the comments:

```
        // The API key and root URL for the REST API.
        // TODO: Replace with your API key and root URL.
        static private string apiKey = "";
        static private string apiRootUrl = "";

        // The SharePoint site and credentials to use with the REST API.
        // TODO: Replace with your site URL, user name, and password.
        static private string spSiteUrl = "";
        static private string spUsername = "";
        static private string spPassword = "";

        // The list form to export, and the name of the destination list for
which
        // the new form is to be imported.
        // TODO: Replace with your form identifier and list title.
        static private string listId = "";
        static private string importFileName = "";

        /// <Summary>
        /// The main entry point for the application.
        /// </Summary>
        [STAThread]
        static void Main()
        {
            // Save a lists.
            SaveForm();
            Console.Write("Press any key to close window.");
            Console.ReadKey();
        }
```

All the configuration you need to do for this example happens here, and the code provided in subsequent steps uses these variables to get an authentication cookie from SharePoint, read the specified export file, and then save the contents of the export file to the specified list.

5. Paste the following code into Program.cs, immediately after the **Main** static method in the **Program** static class:

```
        static private string GetSPOCookie()
        {
            // If successful, this variable contains an authentication
cookie;
            // otherwise, an empty string.
            string result = String.Empty;
            try
            {
                // Construct a secure string from the provided password.
                // NOTE: For sample purposes only.
                var securePassword = new SecureString();
                foreach (char c in spPassword) { securePassword.AppendChar
(c); }
                // Instantiate a new SharePointOnlineCredentials object,
using the
                // specified username and password.
                var spoCredential = new SharePointOnlineCredentials(spUser-
name, securePassword);
                // If successful, try to authenticate the credentials for
the
                // specified site.
                if (spoCredential == null)
                {
                    // Credentials could not be created.
                    result = String.Empty;
                }
                else
                {
                    // Credentials exist, so attempt to get the authen-
tication cookie
                    // from the specified site.
                    result = spoCredential.GetAuthenticationCookie(new Uri
(spSiteUrl));
                }
            }
            catch (Exception ex)
            {
                // An exception occurred while either creating the cre-
dentials or
                // getting an authentication cookie from the specified site.
                Console.WriteLine(ex.ToString());
                result = String.Empty;
            }
```

```
            // Return the result.
            return result;
        }
```

The **GetSPOCookie** static method uses the SharePoint site and credentials that you configured in step 4 to get an authentication cookie from SharePoint.

6. Paste the following code into Program.cs, immediately after the code you pasted in the previous step:

```
      static private void SaveForm()
        {
            // Create a new HTTP client and configure its base address.
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(spSiteUrl);


            // Add common request headers for the REST API to the HTTP cli-
ent.
            new MediaTypeWithQualityHeaderValue("application/json");
            client.DefaultRequestHeaders.Add("Api-Key", apiKey);


            // Get the SharePoint authentication cookie to be used by the
HTTP client
            // for the request, and use it for the Authorization request
header.
            string spoCookie = GetSPOCookie();
            if (spoCookie != String.Empty)
            {
                var authHeader = new AuthenticationHeaderValue(
                    "cookie",
                    String.Format("{0} {1}", spSiteUrl, spoCookie)
                );
                // Add the defined Authorization header to the HTTP client's
                // default request headers.
                client.DefaultRequestHeaders.Add("Test-Environment", "01");
//remove this for my sample
                client.DefaultRequestHeaders.Authorization = authHeader;
            }
            else
            {
                throw new InvalidOperationException("Cannot define Author-
ization header for request.");
            }


            // Read the contents of our form into a byte array, so that we
can send the
            // contents as a ByteArrayContent object with the request.
            if (System.IO.File.Exists(importFileName))
            {
                // Read the file.
                byte[] exportFileContents = System.IO.File.ReadAllBytes
(importFileName);
```

```
                ByteArrayContent saveContent = new ByteArrayContent
(exportFileContents);


                Console.Write("Loading file...");
                Console.Write(saveContent);

                // If we're at this point, we're ready to make our request.
                // Note that we're making this call synchronously - you can
call the REST API
                // asynchronously, as needed.
                var importFormUri = String.Format("{0}/api/v1/forms/{1}",
                    apiRootUrl.TrimEnd('/'),
                    Uri.EscapeUriString(listId));
                HttpResponseMessage saveResponse = client.PutAsync
(importFormUri, saveContent).Result;


                if (saveResponse.IsSuccessStatusCode)
                {
                    Console.WriteLine("Successfully saved form.");
                }
                else
                {
                    Console.WriteLine("Failed to saved form.");
                }
            }
        }
```

The **SaveForm** static method first reads the specified export file, and then uses an HTTP client to invoke the REST resource provided by the REST API to save the contents of the export file to the specified list on your SharePoint site. The client's default request headers are configured, and the **PutAsync** method is used to invoke the REST resource.

7. Build and run the Visual Studio project.

   If you've configured the variables provided in step 4 appropriately, running the project updates the specified list with the contents of the specified export file.

Related Information

Import Form

"Guide" on page 3

Publishing a form

You can use the Nintex Forms for Office 365 REST API to publish an existing SharePoint form. This operation makes the latest draft version of the form available for use.

For more information about the REST resource used to save into a list, see Publish Form.

Considerations

To avoid surprises while trying to publish a form, take the following points into consideration:

- Ensure that the form exists, and that you're allowed to access it, before trying to publish it.
- Ensure that your SharePoint authentication cookie hasn't expired.

  This example avoids that issue by getting an authentication cookie from SharePoint every time you run the example, but you can cache an authentication cookie and avoid a round trip to SharePoint as long as the authentication cookie hasn't expired.
- Don't modify the contents of the export file.

  The export file is a binary file that uses a format specific to Nintex Forms for Office 365. Manipulating the contents can cause unpredictable outcomes and invalidate the format of the export file, making it unusable.

Prerequisites

To use the example, ensure that you have the following prerequisites:

- Access to a SharePoint Online site, with Manage Web permissions for the site.
- A list form that you can export from the SharePoint Online site.
- Access to the Nintex Forms for Office 365 REST API, and an API key with which to authorize the REST API.
- Access to any version of Visual Studio 2013.

Code

You can download the code used in this example from the following location: O365_RESTAPI_NF_Publish.zip

Note: You need to configure the code for the example before you can build and run it. See step 4 in the following example for instructions about how to configure the code for the example.

Example

The following example describes how to create a Visual Studio 2013 project to publish an existing form on a SharePoint site.

1. Create a new Visual Studio 2013 project, using the Console Application template for Visual C#.
2. Add the following references to your new Visual Studio project.
   - Microsoft.SharePoint.Client
   - Microsoft.SharePoint.Client.Runtime
   - System.Net
   - System.Net.Http

3. In your new Visual Studio project, paste the following code into the file named Program.cs, overwriting the existing **using** statements at the beginning of the file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.SharePoint.Client;
using System.Security;
using System.Net.Http.Headers;
using System.Net.Http;
using System.Net;
using System.IO;
```

4. Paste the following code into Program.cs, overwriting the **Main()** static method already included in the **Program** static class, and then configure the code as described in the comments:

```
        // The API key and root URL for the REST API.
        // TODO: Replace with your API key and root URL.
        static private string apiKey = "";
        static private string apiRootUrl = "";

        // The SharePoint site and credentials to use with the REST API.
        // TODO: Replace with your site URL, user name, and password.
        static private string spSiteUrl = "";
        static private string spUsername = "";
        static private string spPassword = "";
        // The list form to export, and the name of the destination list for
which
        // the new form is to be imported.
        // TODO: Replace with your form identifier and list title.
        static private string listId = "";

        /// <Summary>
        /// The main entry point for the application.
        /// </Summary>
        [STAThread]
        static void Main()
        {
            // Copy a list form from a source list to a destination list.
            PublishForm();
            Console.Write("Press any key to close window.");
            Console.ReadKey();
        }
```

All the configuration you need to do for this example happens here, and the code provided in subsequent steps uses these variables to get an authentication cookie from SharePoint, read the specified export file, and then save the contents of the export file to the specified list.

5. Paste the following code into Program.cs, immediately after the **Main** static method in the **Program** static class:

```
        static private string GetSPOCookie()
        {
            // If successful, this variable contains an authentication
cookie;
            // otherwise, an empty string.
            string result = String.Empty;
            try
            {
                // Construct a secure string from the provided password.
                // NOTE: For sample purposes only.
                var securePassword = new SecureString();
                foreach (char c in spPassword) { securePassword.AppendChar
(c); }

                // Instantiate a new SharePointOnlineCredentials object,
using the
                // specified username and password.
                var spoCredential = new SharePointOnlineCredentials(spUser-
name, securePassword);
                // If successful, try to authenticate the credentials for
the
                // specified site.
                if (spoCredential == null)
                {
                    // Credentials could not be created.
                    result = String.Empty;
                }
                else
                {
                    // Credentials exist, so attempt to get the authen-
tication cookie
                    // from the specified site.
                    result = spoCredential.GetAuthenticationCookie(new Uri
(spSiteUrl));
                }
            }
            catch (Exception ex)
            {
                // An exception occurred while either creating the cre-
dentials or
                // getting an authentication cookie from the specified site.
                Console.WriteLine(ex.ToString());
                result = String.Empty;
            }
```

```
            // Return the result.
            return result;
        }
```

The **GetSPOCookie** static method uses the SharePoint site and credentials that you configured in step 4 to get an authentication cookie from SharePoint.

6. Paste the following code into Program.cs, immediately after the code you pasted in the previous step:

```
        static private void PublishForm()
        {
            // Create a new HTTP client and configure its base address.
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(spSiteUrl);

            // Add common request headers for the REST API to the HTTP cli-
ent.
            client.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/json"));
            client.DefaultRequestHeaders.Add("Api-Key", apiKey);

            // Get the SharePoint authorization cookie to be used by the
HTTP client
            // for the request, and use it for the Authorization request
header.
            string spoCookie = GetSPOCookie();
            if (spoCookie != String.Empty)
            {
                var authHeader = new AuthenticationHeaderValue(
                    "cookie",
                    String.Format("{0} {1}", spSiteUrl, spoCookie)
                );
                // Add the defined authentication header to the HTTP cli-
ent's
                // default request headers.
                client.DefaultRequestHeaders.Authorization = authHeader;
            }
            else
            {
                throw new InvalidOperationException("Cannot define Author-
ization header for request.");
            }
            // Console test block
            var testURI = String.Format("{0}/api/v1/forms/{1}/publish",
                apiRootUrl.TrimEnd('/'),
                Uri.EscapeUriString(listId));
            Console.Write(testURI,"\n");

            // If we're at this point, we're ready to make our request.
            // Note that we're making this call synchronously - you can call
the REST API
            // asynchronously, as needed.
```

```
            var publishFormUri = String.Format("{0}/api/v1/forms/{1}/pub-
lish",
                apiRootUrl.TrimEnd('/'),
                Uri.EscapeUriString(listId));
            HttpResponseMessage publishResponse = client.PostAsync(pub-
lishFormUri, new StringContent("")).Result;


            if (publishResponse.IsSuccessStatusCode)
            {
                Console.WriteLine("Successfully published form.");
            }
            else
            {
                Console.WriteLine("Failed to publish form.");
            }
        }
```

The **PublishForm** static method uses an HTTP client to invoke the REST resource provided by the REST API to publish the specified form on your SharePoint site. The client's default request headers are configured, and the **PostAsync** method is used to invoke the REST resource, with an empty string as content.

7. Build and run the Visual Studio project.

   If you've configured the variables provided in step 4 appropriately, running the project updates the specified list with the contents of the specified export file.

Related Information

Get Form

"Guide" on page 3

Delete a form

You can use the Nintex Forms for Office 365 REST API to publish an existing SharePoint form. This operation makes the latest draft version of the form available for use.

For more information about the REST resource used to save into a list, see Publish Form.

Considerations

To avoid surprises while trying to publish a form, take the following points into consideration:

- Ensure that the form exists, and that you're allowed to access it, before trying to publish it.
- Ensure that your SharePoint authentication cookie hasn't expired.

  This example avoids that issue by getting an authentication cookie from SharePoint every time you run the example, but you can cache an authentication cookie and avoid a round trip to SharePoint as long as the authentication cookie hasn't expired.
- Don't modify the contents of the export file.

  The export file is a binary file that uses a format specific to Nintex Forms for Office 365. Manipulating the contents can cause unpredictable outcomes and invalidate the format of the export file, making it unusable.

Prerequisites

To use the example, ensure that you have the following prerequisites:

- Access to a SharePoint Online site, with Manage Web permissions for the site.
- A list form that you can export from the SharePoint Online site.
- Access to the Nintex Forms for Office 365 REST API, and an API key with which to authorize the REST API.
- Access to any version of Visual Studio 2013.

Code

You can download the code used in this example from the following location:

O365_RESTAPI_NF_Delete.zip

Note: You need to configure the code for the example before you can build and run it. See step 4 in the following example for instructions about how to configure the code for the example.

Example

The following example describes how to create a Visual Studio 2013 project to publish an existing form on a SharePoint site.

1. Create a new Visual Studio 2013 project, using the Console Application template for Visual C#.
2. Add the following references to your new Visual Studio project.
    - Microsoft.SharePoint.Client
    - Microsoft.SharePoint.Client.Runtime
    - System.Net
    - System.Net.Http
3. In your new Visual Studio project, paste the following code into the file named Program.cs, overwriting the existing **using** statements at the beginning of the file:

```
using System;

using System.Text;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.SharePoint.Client;

using System.Security;

using System.Net.Http.Headers;

using System.Net.Http;

using System.Net;

using System.IO;
```

4. Paste the following code into Program.cs, overwriting the **Main()** static method already included in the **Program** static class, and then configure the code as described in the comments:

```
 // The API key and root URL for the REST API.
        // TODO: Replace with your API key and root URL.
        static private string apiKey = "";
        static private string apiRootUrl = "";

        // The SharePoint site and credentials to use with the REST API.
        // TODO: Replace with your site URL, user name, and password.
        static private string spSiteUrl = "";
        static private string spUsername = "";
        static private string spPassword = "";

        // The list form to export, and the name of the destination list for
which
        // the new form is to be imported.
        // TODO: Replace with the path to your form package (NFP or XML)
        // and target list title.
        static private string importFileName = "";
        static private string listId = "";
        /// <Summary>
        /// The main entry point for the application.
        /// </Summary>
        [STAThread]
        static void Main()
        {
            // Copy a list form from a local form file to a destination
list.
            DeleteForm();
            Console.Write("Press a key to close the window.");
            Console.ReadKey();
        }
```

All the configuration you need to do for this example happens here, and the code provided in subsequent steps uses these variables to get an authentication cookie from SharePoint, read the specified export file, and then save the contents of the export file to the specified list.

5. Paste the following code into Program.cs, immediately after the **Main** static method in the **Program** static class:

```
 static private string GetSPOCookie()
        {
            // If successful, this variable contains an authentication
cookie;
            // otherwise, an empty string.
            string result = String.Empty;
            try
            {
                // Construct a secure string from the provided password.
                // NOTE: For sample purposes only.
                var securePassword = new SecureString();
                foreach (char c in spPassword) { securePassword.AppendChar
(c); }
                // Instantiate a new SharePointOnlineCredentials object,
using the
                // specified username and password.
                var spoCredential = new SharePointOnlineCredentials(spUser-
name, securePassword);
                // If successful, try to authenticate the credentials for
the
                // specified site.
                if (spoCredential == null)
                {
                    // Credentials could not be created.
                    result = String.Empty;
                }
                else
                {
                    // Credentials exist, so attempt to get the authen-
tication cookie
                    // from the specified site.
                    result = spoCredential.GetAuthenticationCookie(new Uri
(spSiteUrl));
                }
            }
            catch (Exception ex)
            {
                // An exception occurred while either creating the cre-
dentials or
                // getting an authentication cookie from the specified site.
                Console.WriteLine(ex.ToString());
                result = String.Empty;
            }
```

```
            // Return the result.
            return result;
        }
```

The **GetSPOCookie** static method uses the SharePoint site and credentials that you configured in step 4 to get an authentication cookie from SharePoint.

6. Paste the following code into Program.cs, immediately after the code you pasted in the previous step:

```
 static private void DeleteForm()
        {
            // Create a new HTTP client and configure its base address.
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(spSiteUrl);


            // Add common request headers for the REST API to the HTTP cli-
ent.
            client.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/json"));
            client.DefaultRequestHeaders.Add("Api-Key", apiKey);


            // Get the SharePoint authentication cookie to be used by the
HTTP client
            // for the request, and use it for the Authorization request
header.
            string spoCookie = GetSPOCookie();
            if (spoCookie != String.Empty)
            {
                var authHeader = new AuthenticationHeaderValue(
                    "cookie",
                    String.Format("{0} {1}", spSiteUrl, spoCookie)
                );
                // Add the defined Authorization header to the HTTP client's
                // default request headers.
                client.DefaultRequestHeaders.Authorization = authHeader;
            }
            else
            {
                throw new InvalidOperationException("Cannot define Author-
ization header for request.");
            }


            // Read the contents of our form into a byte array, so that we
can send the
            // contents as a ByteArrayContent object with the request.
            if (System.IO.File.Exists(importFileName))
            {


                // If we're at this point, we're ready to make our request.
                // Note that we're making this call synchronously - you can
call the REST API
                // asynchronously, as needed.
```

```
                var importFormUri = String.Format("{0}/api/v1/forms/{1}",
                    apiRootUrl.TrimEnd('/'),
                    Uri.EscapeUriString(listId));
                HttpResponseMessage saveResponse = client.DeleteAsync
(importFormUri).Result;

                if (saveResponse.IsSuccessStatusCode)
                {
                    Console.WriteLine(Successfully deleted form.");
                }
                else
                {
                    Console.WriteLine("Failed to delete form.");
                }
            }
        }
```

The **DeleteForm** static method uses an HTTP client to invoke the REST resource provided by the REST API to publish the specified form on your SharePoint site. The client's default request headers are configured, and the **PostAsync** method is used to invoke the REST resource, with an empty string as content.

7. Build and run the Visual Studio project.

   If you've configured the variables provided in step 4 appropriately, running the project updates the specified list with the contents of the specified export file.

Related Information

Delete Form

"Guide" on page 3

Base URL

All of the operations provided by the Nintex Forms for Office 365 REST API use the following root path as the base URL, replacing *{customer}* with the customer-specific portion of the URL and *{version}* with the appropriate version of the REST API:

```
{customer}.nintexo365.com/api/{version}
```

For example, the following URL represents a request to export a list form, using version 1 of the REST API implemented for a fictional customer named Crestan:

```
https://crestan.nintexo365.com/api/v1/forms/fd4f1cd2-7ea7-4b62-9751-0ff83ab609f7
```

The REST API is served over the HTTPS protocol. To ensure data privacy, unencrypted HTTP is not supported.

Related Information

REST API Resources

Authentication and authorization

The Nintex Forms for Office 365 REST API requires that you have both a subscription to the REST API and authorization to use the REST API on a specified SharePoint site.

- "Authenticating operations for SharePoint Online" on the next page
- "Authorizing operations for the Nintex Forms for Office 365 REST API" on page 41

Authenticating operations for SharePoint Online

The Nintex Forms for Office 365 REST API takes advantage of Office 365 passive authentication capabilities, using SharePoint Online credentials and Windows Azure Active Directory to authorize an operation on a specified SharePoint site. An authorization cookie, retrieved from SharePoint Online, that represents a valid credential for the specified site must be provided to authorize the invocation of operations included with the REST API.

You must include the **Authorization** request header with every operation. The request header must contain a cookie that uses the following format, replacing <*site*> with the SharePoint site URL and <*authcookie*> with a valid SharePoint SPOIDCRL or FedAuth authentication cookie for the specified site, as needed, to authenticate the request with SharePoint.

```
cookie <site> <authcookie>
```

Required SharePoint permissions

Before retrieving an authentication cookie, ensure that the credential to be used for the specified SharePoint site has the ability to perform all administration tasks for the Web site and manage website content. In other words, the credential must belong to a role in SharePoint that has the Manage Web permission for the specified SharePoint site; otherwise, an error occurs when the authentication cookie is used to invoke a REST API operation.

Obtaining an authentication cookie

You can use the **GetAuthenticationCookie** operation of the **SharePointOnlineCredentials** object, in the **Microsoft.SharePoint.Client** namespace, to obtain a SPOIDCRL authentication cookie for use with the Nintex Forms for Office 365 REST API.

The **Microsoft.SharePoint.Client** namespace is included with the SharePoint Online Client Side Object Model (CSOM). You can add a reference to the SharePoint Online CSOM to your Visual Studio 2013 project as a NuGet package, provided by the Office Developer Platform Team on NuGet at https://www.nuget.org/packages/Microsoft.SharePointOnline.CSOM/16.1.3912.1204.

Visual C#

The following Visual C# method demonstrates how to use the **GetAuthenticationCookie** method to get an authentication cookie for a specified SharePoint site, using a specified username and password.

```
/// <summary>
/// Gets a SharePoint Online authentication cookie from the specified site, using
/// the specified username and password.
/// </summary>
/// <param name="siteUrl">The site with which to authenticate.</param>
/// <param name="username">The username of the credentials to authenticate.</param>
/// <param name="password">The password of the credentials to authenticate.</param>
/// <returns>If successful, a SharePoint Online authentication cookie;
/// otherwise, an empty string.</returns>
static public string GetSPOCookie(string siteUrl, string username, string password)
{
    // If successful, this variable contains an authentication cookie;
    // otherwise, an empty string.
    string result = String.Empty;
    try
    {
        // Construct a secure string from the provided password.
        // NOTE: For sample purposes only.
        var securePassword = new SecureString();
        foreach (char c in password) { securePassword.AppendChar(c); }

        // Instantiate a new SharePointOnlineCredentials object, using the
        // specified username and password.
        var spoCredential = new SharePointOnlineCredentials(username, securePassword);
```

```
        // If successful, try to authenticate the credentials for the
        // specified site.
        if (spoCredential == null)
        {
            // Credentials could not be created.
            result = String.Empty;
        }
        else
        {
            // Credentials exist, so attempt to get the authentication cookie
            // from the specified site.
            result = spoCredential.GetAuthenticationCookie(new Uri(siteUrl));
        }
    }
    catch (Exception ex)
    {
        // An exception occurred while either creating the credentials or
        // getting an authentication cookie from the specified site.
        Console.WriteLine(ex.ToString());
        result = String.Empty;
    }

    // Return the result.
    return result;
}
```

Windows PowerShell

The following Windows PowerShell script example demonstrates how to use the **GetAuthenticationCookie** method to get an authentication cookie for a specified SharePoint site, using a specified credential. The **PSCredential** object can provides an interactive interface, if needed, in which a credential can be supplied by the user, or you can pipe a valid **PSCredential** object directly to the Windows PowerShell script at invocation.

```
<#
.SYNOPSIS
Retrieves a SharePoint Online authentication cookie for the specified SharePoint URI and cre-
dential.
.DESCRIPTION
.PARAMETER SiteURI
Required. A System.Uri object that represents the SharePoint site for the authentication cookie.
.PARAMETER Credential
Required. A PSCredential object that represents the credential for the authentication cookie.
.EXAMPLE
Get-SPOCookie -SiteURI "https://crestan.sharepoint.com" -Credential "spowner@crestan.com"
#>
[CmdletBinding()]
param (
    [Parameter(Mandatory=$true)]
    [System.Uri]$SiteURI,
    [Parameter(Mandatory=$true)]
    [PSCredential]$Credential
)

# Load the SharePoint client assemblies into the PowerShell session.
# If needed, change the paths to match the location of the SharePoint 2013 client assemblies. The
paths provided
# below are the default locations for the SharePoint 2013 client assemblies.
Add-Type -Path "C:\Program Files\Common Files\Microsoft Shared\Web Server Exten-
sions\15\ISAPI\Microsoft.SharePoint.Client.dll"
Add-Type -Path "C:\Program Files\Common Files\Microsoft Shared\Web Server
Extensions\15\ISAPI\Microsoft.SharePoint.Client.Runtime.dll"

# Create a new SharePointOnlineCredentials object, using the specified credential.
```

```
$SPOCred = New-Object -TypeName Microsoft.SharePoint.Client.SharePointOnlineCredentials -Argu-
mentList $Credential.UserName, $Credential.Password

# Return the authentication cookie from the SharePointOnlineCredentials object,
# using the specified SharePoint site.
$SPOCred.GetAuthenticationCookie($SiteURI)
```

Authorizing operations for the Nintex Forms for Office 365 REST API

The Nintex Forms for Office 365 REST API requires an API key, issued by Nintex, to authorize the invocation of operations included in the REST API. You must include the API key in the **Api-Key** request header included with every operation.

Obtaining an API key

You can obtain an API key by contacting your Nintex representative.

Related Information

REST API Resources

Getting Your SharePoint 0365 Credentials via REST

This topic describes how to get your authentication cookie via REST in a workflow using the Nintex Forms O365 API only using HTTP.

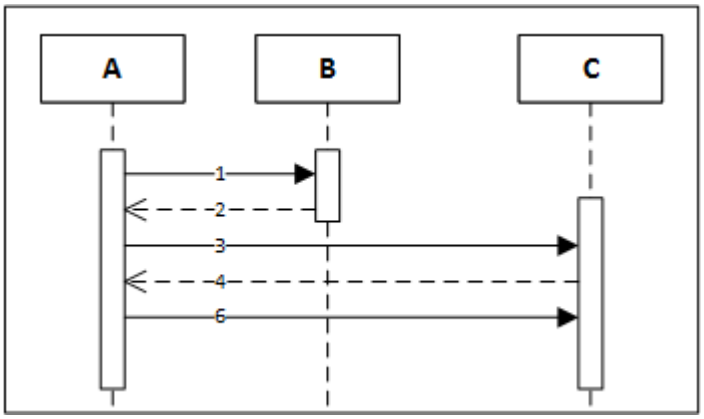
Using HTTP to establish your credentials with the SharePoint O365 tenant may be useful if you are developing applications in a context outside of the SharePoint and .NET. If you are using SharePoint it probably is more expedient to follow the instructions at "Authentication and authorization" on page 38, or refer to the code samples in the "Guide" on page 3 which retrieves an authorization cookie during execution.

The authentication process involves five steps, which you can see in the sequence diagram below.

1. Send Security Assertion Markup Language (SAML) request, including username and password to Microsoft Online.
2. Receive SAML return payload which includes the security token.
3. Post the security token to your SharePoint O365 tenancy.
4. Receive authentication cookies.
5. Send request including authentication cookies.

The actors include:

A. Your application
B. Microsoft Online
C. SharePoint O365 tenant



Sequence diagram for retrieving your authentication cookies

Get Your Authentication Cookies

You will need authentication cookies to work with the Forms for Office 365 API. You can retrieve the cookies by connecting with Microsoft Online, and then using your token to get an access token form your tenancy, and then use the access token to get the authorization cookies.

Send SAML request, including username and password, to Microsoft Online

Provide the username and password and the URL at which we want access to the SharePoint Online Security Token Service along with an envelope in the request body.

Send to the endpoint: https://login.microsoftonline.com/extSTS.srf

In the following template replace the username, password, and SharePoint address with your values:

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"

     xmlns:a="http://www.w3.org/2005/08/addressing"

     xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">

  <s:Header>

  <a:Action s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue</a:Action>

    <a:ReplyTo>

      <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>

    </a:ReplyTo>

    <a:To s:mustUnderstand="1">https://login.microsoftonline.com/extSTS.srf</a:To>

      <o:Security s:mustUnderstand="1"

      xmlns:o="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd">

      <o:UsernameToken>

      <o:Username>Username@yourdomain.sharepoint.com</o:Username>

      <o:Password>password</o:Password>

      </o:UsernameToken>

      </o:Security>

    </s:Header>

    <s:Body>

    <t:RequestSecurityToken xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">

     <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">

     <a:EndpointReference>

     <a:Address>https://yourdomain.sharepoint.com</a:Address>

    </a:EndpointReference>

  </wsp:AppliesTo>

     <t:KeyType>http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey</t:KeyType>

       <t:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Issue</t:RequestType>

       <t:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</t:TokenType>

     </t:RequestSecurityToken>

  </s:Body>
```

```
</s:Envelope>
```

Receive SAML return payload which includes the security token

The return to your SAML request contains the security token that you will need to use to retrieve the access token from your SharePoint site.

You can find your security token in the **BinarySecurityToken** element. In the following example of a SAML payload you can find the token between the `<wsse:BinarySecurityToken Id="Compact0">`*security token*`</wsse:BinarySecurityToken>`.

Example:

<wsse:BinarySecurityToken Id="Compact0">

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsa-a="http://www.w3.org/2005/08/addressing">

  <S:Header>

    <wsa:Action xmlns:S="http://www.w3.org/2003/05/soap-envelope" xmlns:wsa-a="http://www.w3.org/2005/08/addressing" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="Action" S:mustUnder-stand="1">http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue</wsa:Action>

    <wsa:To xmlns:S="http://www.w3.org/2003/05/soap-envelope" xmlns:wsa-a="http://www.w3.org/2005/08/addressing" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="To" S:mustUnder-stand="1">http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>

    <wsse:Security S:mustUnderstand="1">

      <wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse-curity-utility-1.0.xsd" wsu:Id="TS">

      <wsu:Created>2016-02-19T21:05:13Z</wsu:Created>

      <wsu:Expires>2016-02-19T21:10:13Z</wsu:Expires>

      </wsu:Timestamp>

    </wsse:Security>

  </S:Header>

<S:Body>

  <wst:RequestSecurityTokenResponse xmlns:S="http://www.w3.org/2003/05/soap-envelope" xmlns:wst-t="http://schemas.xmlsoap.org/ws/2005/02/trust" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:wsp-p="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:psf="http://schemas.microsoft.com/Passport/SoapServices/SOAPFault">

    <wst:TokenType>urn:passport:compact</wst:TokenType>

    <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing">

    <wsa:EndpointReference>

    <wsa:Address>https://nintexinteraction01.sharepoint.com/FormsOnline/api</wsa:Address>

  </wsa:EndpointReference>

</wsp:AppliesTo>

<wst:Lifetime>

  <wsu:Created>2016-02-19T21:05:13Z</wsu:Created>

    <wsu:Expires>2016-02-20T21:05:13Z</wsu:Expires>
```

```
  </wst:Lifetime>

<wst:RequestedSecurityToken>

  <wsse:BinarySecurityToken Id="Co-
mpact0">t-
=EwA4A06hBwAUNfDkMme61kIdXqvj9tWnUbHtXWEAAQbiCLW4yppnBO220/abLI1J6M4oq50n8KOFhGdsUnqeUGYbhWVp6dmhY5DY3nSEgyKuVFpKt9

</wst:RequestedSecurityToken>

  <wst:RequestedAttachedReference>

     <wsse:SecurityTokenReference>

     <wsse:Reference URI="bzi+G/bkxv72sxupv7IGHGhG+Ww="></wsse:Reference>

  </wsse:SecurityTokenReference>

</wst:RequestedAttachedReference>

  <wst:RequestedUnattachedReference>

    <wsse:SecurityTokenReference>

        <wsse:Reference URI="bzi+G/bkxv72sxupv7IGHGhG+Ww="></wsse:Reference>

    </wsse:SecurityTokenReference>

  </wst:RequestedUnattachedReference>

</wst:RequestSecurityTokenResponse>

</S:Body>

</S:Envelope>
```

Post the security token to your SharePoint Online tenancy

Once the security token has been retrieved, post the token to SharePoint 0365 to get your authorization cookies. Post the token as the body to the following endpoint:

https://*yourdomain.sharepoint.com*/_forms/default.aspx?wa=wsignin1.0

> **Note:** You will want to make sure you change the HTML entity (&) at the end of the token to a literal ampersand (&).

Receive authentication cookies

The response from the request will include the request digest in the XML response and two cookies that you can use to send a request to the API endpoint. Retrieve the value of both cookies. The following table contains examples of the authentication cookies.

Key	Value
FedAuth	77u/PD94bWwgdmVyc2lv bj0iMS4wIiBlbmNvZGlu Zz0idXRmLTgiPz48U1A+ RmFsc2UsMGguZnxtZW1i ZXJzaGlwfDEwMDNiZmZk ODkzMzFlMGZAbGl2ZS5j b20sMCMuZnxtZW1iZXJz aGlwfHNwc2l0ZWFkbWlu QG5pbnRleGludGVyYWN0 aW9uMDEub25taWNyb3Nv ZnQuY29tLDEzMTAxMTU4 MDM0MDYyMTYzNjsxMzA0 ODIxOTkyMDAwMDAwMDAs RmFsc2UsVVNTUTdPNEV6 bzQ3azBtak5Rb2RnWHlm ZTZHOEZjZ3FLVmQxeWlN aG1xb0VIeVdhdXgxMW9N KzlhZ2w2ZE1hZFIzZW9p b2EzN3RDSmFFTjVUSGtl UHNHS1lCYzMzd2EzdDJI V0J0R043MStlTmkybmNC cFE1QWg1Yjg5L0ozWW80 N01xd0VvczBWcFMzV2tE TWYxSTRvZktBL0kzNFZO M0RGbUxvdDVQNVFmUDNL S3lMYWlUakkxdmp1azZG enp0MHRHdDZCcUE4cFF0 V0JweHNmeVRiNDZEVmZ5 Kzg1WkFONVBldFIxWFVv SUlHKytrREhLV1JyVnYz SzVaQkJ6VHFvZzNrTTBO V0dYaWhOVGhseTV0bk1Y NVpXZmJrUEt1M2hHUTBh dE4yUU1JRDBmSmVZZ1Q0 bjRsdG5rWnozeWw2UmRj WXgzUCt3WklOMHNMSThi QkdRPT0saHR0cHM6Ly9u aW50ZXhpbnRlcmFjdGlv bjAxLnNoYXJlcG9pbnQu Y29tLzwvU1A+
rtFa	1h7uD64IbzIMJbaMdvkS ESxgr1Pvmip7im344/ea Y4wD4NRaDm7vKG5o15Ea LbLSRfxo6RyL7R5Y0V// 0SJu8pMAwbGICBRmT4GJ DtWfulY3oQ/R7OtSuXHv Z5d5K5NW+K4L7UCJOCGG ARFklIewKoGcUNNX57D1 nMbVWjO8FacVD3YQ5PQT 7nCE1E7qwN+YMRBhermV b9QlpT7wfDk5q9yMSw1v NT7/IjO5CLaMZtr3waq4 nAWB4i9tzV5PPXhMdUDv dPK2pk952kIFC/cnrvVQ WXQaR0rI0ooYz7KbiYyO /NivVWofEkgA8Y180g1e ju0BpBT35/0F7mcq3peb FzC2GaKUYQdczZTYLw58 jeobnJoGGgbixwBbmeWg ZYruIAAAAA==

Send request including authentication cookies

For each request to the API, you must include the FedAuth and rtFA cookie as headers in your request with the following format:

- Cookie: FedAuth=*77u...*
- Cookie: rtFA=*1h7...*

SharePoint Online will be able to accept your authorization and will process your request.

Related Information

REST API Resources

Common headers

The Nintex Forms for Office 365 REST API uses both common request and response headers, supported by all operations included in the REST API, and specific request and response headers, used only in certain operations. This section describes the common request and response headers used in the REST API. For more information about specific request and response headers used for a specific operation in the REST API, see the documentation for that operation.

Request headers

The following request headers are supported by all operations included with the Nintex Forms for Office 365 REST API.

Header	Description
Accept	Set to the following value: `application/json`

Header	Description
Authorization	A cookie that contains the SharePoint site used by the REST operation and the SharePoint authentication cookie used to authenticate the REST operation for SharePoint. The cookie uses the following format, replacing <*site*> with the SharePoint site URL and <*authcookie*> with a valid SharePoint SPOIDCRL or FedAuth authentication cookie for the specified site, as needed, to authenticate the request with SharePoint. `cookie <site> <authcookie>` For more information, see "Authentication and authorization" on page 38.
Api-Key	The API key for NintexForms for Office 365 REST API. For more information, see "Authentication and authorization" on page 38.

Response headers

The following response headers are supported by all operations included with the Nintex Forms for Office 365 REST API. Depending on SharePoint and ASP.NET configuration, other headers may also be returned.

Response Header	Description
Content-Length	The length, in octlets, of the response body.
Content-Type	The MIME type of the content in the response body. Unless specified by the operation, set to the following value: `application/json; charset=utf-8`
Correlation-Identifier	The correlation ID of the operation.
Date	The date and time at which the response was sent by the server.

Related Information

REST API Resources

Common status and error codes

All responses from operations in the Nintex Forms for Office 365 REST API include an HTTP status code, indicating whether the operation was successful. If an error was encountered by an operation, additional information about the error can be provided in the response body, depending on the error.

HTTP status codes

HTTP status codes are used to indicate the status of an operation in the Nintex Forms for Office 365 REST API. The following common HTTP status codes are supported by the REST API. For more information about the **Message** and **Error** objects in responses, see "Response bodies" on the next page.

Status code	Description
200	OK The operation was successful. The operation can return a **Message** object in the response body, depending on the operation.

Status code	Description
400	Bad Request An invalid operation has occurred. The operation can return an **Error** object in the response body, further describing the error, depending on the operation.
401	Unauthorized The operation is not authorized for the specified user. The operation returns an **Error** object in the response body, further describing the error.
403	Forbidden. The operation is forbidden. The operation returns an **Error** object in the response body, further describing the error.
404	Not Found The specified resource could not be found. The operation returns an **Error** object in the response body, further describing the error, depending on the operation.
405	Method Not Allowed The operation is not allowed. The operation returns an **Error** object in the response body, further describing the error.
500	Internal Server Error The application has encountered an unknown error. The operation can return an **Error** object in the response body, further describing the error, depending on the operation.

Nintex Forms for Office 365 error codes

If an HTTP status code other than a success code is returned by an operation, an **Error** object is returned in the response body of the operation, in which additional details of the error are provided. The following common error codes are supported by the Nintex Forms for Office 365 REST API.

Error code	Message
NF-1001	The Nintex Forms app was not found for the specified site.
NF-1002	The user does not have permission.
NF-1003	Failed to publish Nintex Form.
NF-1004	The specified package is not valid.
NF-1005	The specified list could not be found in the site.
NF-1006	Nintex form not found for the specified list.
NF-1009	Failed to delete Nintex Form.

Related Information

REST API Resources

Response bodies

The responses provided by operations in the Nintex Forms for Office 365 REST API typically use one of two objects to represent the response body of the response. If an operation was successful, the response body contains a "Message" on the next page object that represents the information returned by the operation. Otherwise, the response body contains an "Error" on the next page object, which provides information about the error response.

Message

The **Message** object represents the response body of a successful response returned by an operation. The object can provide information about the identifier, data, and metadata about the successful response, as well as links to other operations relevant to the successful response.

Properties

Name	Type	Nullable	Description
data	varies	true	If specified, the data returned by the operation.
metadata	varies	true	If specified, the metadata returned by the operation.
id	string	true	If specified, the identifier returned by the operation.
_links	array	true	If specified, an array of **Link** objects, representing relative links to other REST operations relevant to the operation. For more information about the **Link** object, see "Data types" on the next page.

Remarks

When an operation returns a successful response, the content of the response varies, depending on the operation itself. For more information about the contents of a response body for a successful response from an operation, see the documentation for that operation.

Example

The following example describes the response body of a successful response received while attempting to import a form to a SharePoint list. For more information about importing a new form, see "Importing a form" on page 12. The **id** property provides the identifier of the new form, and the **_links** property provides links to relevant operations, relative to the new form.

```
{
  "id": "fd4f1cd2-7ea7-4b62-9751-0ff83ab609f7",
  "_links": [
    {
      "rel": "self",
      "href": "{?migrate}",
      "isTemplated": true
    },
    {
      "rel": "export",
      "isTemplated": false
    },
    {
      "rel": "publish",
      "isTemplated": false
    }
  ]
}
```

Error

The **Error** object represents the response body of an error response returned by an operation. The object can provide information about the SharePoint correlation ID for the error, internal error code, and additional information about the error response, as well as links to other operations relevant to the error response.

Properties

Name	Type	Nullable	Description
code	string	true	If specified, the internal error code of the error. For more information about error subcodes, see "Common status and error codes" on page 46.
correlationId	string	false	The correlation ID of the error.
message	string	true	If specified, the description of the error.
moreInfo	string	true	If specified, a string that represents a URI which can provide additional information about the error.
_links	array	true	If specified, an array of **Link** objects, representing relative links to other REST operations relevant to the error. For more information about the **Link** data type, see "Data types" below.

Remarks

When an error is encountered by an operation in the Nintex Forms for Office 365 REST API, this type can return an internal error code in **code**. The internal error code represents either a common error code for the Nintex Forms for Office 365 API, or a specific error code for that operation. For more information about error codes specific to an operation, see the documentation for that operation. For more information about error codes common to all operations in Nintex Forms for Office 365, see "Common status and error codes" on page 46.

Example

The following example describes the response body of a successful response received after importing a form. For more information about importing a form, see Import Form. The **rel** property contains the relative URL of the default view for the specified SharePoint list, relative to the SharePoint site, the **href** property contains a relative URL of the endpoint to publish the form., and the **isTemplated** property indicates if the form is based on a template.

```
{
"_links": [
     {
     "rel": "forms.publishdefault",
     "href": "/api/v1/forms/6263627c-57a6-42d0-9c87-2232e4e1899d/publish",
     "isTemplated": false
    }
  ]
}
```

Related Information

REST API Resources

Data types

The following section describes the data types, other than response bodies, used by the Nintex Forms for Office 365 REST API. For more information about response bodies, see Response bodies.

The following data types are provided by the REST API:

- "Link" on the next page

Link

The **Link** object represents a relative link to another resource, such as an operation in the REST API.

Properties

Name	Type	Nullable	Description
rel	string	false	The relation, or name, of the link.
href	string	true	The relative URL or URL template of the link. If **isTemplated** is set to true, this property contains a relative URL template; otherwise, this property contains a relative URL. In either case, the value is relative to the base URL of the REST API. For more information about the base URL, see "Base URL" on page 38.
isTemplated	boolean	false	If set to true, the value of the **href** property represents a relative URL template; otherwise, the value represents a relative URL.

Remarks

The **Link** objects provided by the REST API are compatible with the Hypertext Application Language (HAL) specification. For more information about the HAL specification, see HAL - Hypertext Application Language, at http://stateless.co/hal_specification.html.

Example

The following example illustrates a collection of **Link** objects, returned in the **_links** collection of a **Message** response body. All three **Link** objects represent relative URL templates, covering four operations included in the REST API.

```
{
  "_links": [
    {
       "rel": "forms.publishdefault",
       "href": "/api/v1/forms/b7fea5eb-db3c-45f5-b590-ddcd0b12bd4e/publish",
       "isTemplated": false
    }
  ]
}
```

Related Information

"Response bodies" on page 47

REST API Resources