3rd Party User Provisioning with Salesforce Identity

Looking for a way to use Salesforce to automatically manage users in other services and applications? With user provisioning for connected apps, you can connect your Salesforce users to users in other clouds and third-party applications. Once connected, you can create, update, and delete these users simply by managing your users in Salesforce.

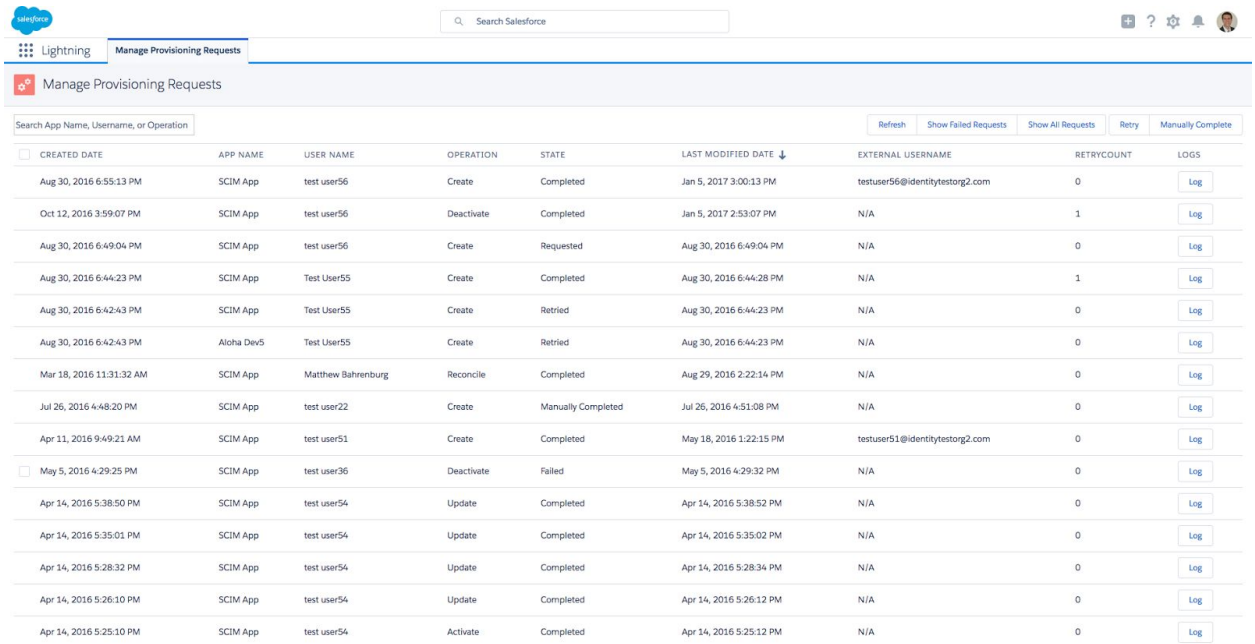
Salesforce provides integrations to several commonly used applications, as well as utilities to help you customize and build your own integrations. This document explains all the components that Salesforce provides, and walks your through the installation and configuration of each. By completing this document, you will be able to effectively link your Salesforce users to Box, Concur, Dropbox, Google, GoToMeeting, Office 365, ServiceNow, WebEx, Zendesk, and SCIM 1.1 compliant applications (like Salesforce).

If you're new to user provisioning on the Salesforce platform, check out this video to get you started.

Table of Contents

User Provisioning Connector Utils Package

Salesforce provides two packages for user provisioning. The first is the User Provisioning Connector Utils package. This package contains shared utility classes that can be used by all your user provisioning integrations; including those written by Salesforce, our partners, or your own development team. It also provides helpful reports, and, for our lightning trailblazers, an app for managing all your provisioning requests.



This app enables you to see all your provisioning requests and request logs. Therefore, if you need to check the status of a request, or troubleshoot an issue, you'll have all the information available in one place. You can even retry requests once you've addressed any issues. After you've installed the package, you can access this app by switching to Lightning Experience and opening the "Manage Provisioning Requests" item from the App Launcher.

User Provisioning Connectors Package

The second package that Salesforce provides for user provisioning is the connector package. This package contains integrations or "connectors" to manage users on the following applications:

- Box
- Concur
- Dropbox
- Google
- GoToMeeting
- Office365
- Salesforce (And any other provider that supports SCIM v1.1)
- ServiceNow
- Webex
- Zendesk

The package includes about 200 Apex classes, 10 flows, and 10 custom settings. That's a lot of code! Each connector follows a strict naming standard that starts with the application name. For example, `BoxCollectUsersPlugin.cls`, Google_Users_flow-1.flow, etc. If you only need to manage a single application, install the package on a DE org or sandbox, and migrate only the components matching the application name you need.

We realize that everyone has different requirements for how they manage their users, so the source code is provided. That's right, we give you the source code! Feel free to update the code or use it as a baseline to develop your own integrations.

Organization Setup

Now let's get started. First we'll setup your Salesforce org for user provisioning. If you don't have a Saleforce org, signup for a free Developer Edition orgs here: https://developer.salesforce.com/signup.

Set Up My Domain

For simplicity and easy management, you need to give your org a unique URL using “My Domain.” If you’re unfamiliar with My Domain, see Define Your Domain Name in Salesforce Help or the Setting Up a My Domain video.

For your org, do the following:

1. Navigate to **Setup > Domain Management > My Domain**.
2. Follow the steps to select and assign a unique name for your org.
3. Click **Deploy to Users**.

Install the User Provisioning Connector Utils Package

After you configure your domain, you’re ready to install the first package.

1. Go to it’s listing on Salesforce AppExchange.
2. Select **Get It Now** and follow the install flow.
3. If prompted to login, make sure you login with your admin user for your target org. If you're not prompted to login, verify from your My Domain URL that you're installing into the right org. (If not, stop the process, logout, and start again.) You can install for Admins Only if you wish.
4. For more information about the installation options, see Installing Packages.

Install the User Provisioning Connectors Package

After you install the Connector Utilities package, you’re ready for the optional Connector package. If you’re writing your own integration, you only need the Connector Utility package. However, if you’d like to use our code as an example, or use the integrations we’ve created, you’ll need both packages. The Connector Utility package is also a dependency of the Connector package, so you’ll need to ensure that it’s installed first. To install the connector package:

1. Go to it’s listing on Salesforce AppExchange.
2. Select **Get It Now** and follow the install flow.
3. If prompted to login, make sure you login with your admin user for your target org. If you're not prompted to login, verify from your My Domain URL that you're installing into the right org. (If not, stop the process, logout, and start again.) You can install for Admins Only if you wish.
4. For more information about the installation options, see Installing Packages.

Setup Each Connector

As mentioned earlier, the connector package comes with several connectors. In the next section we'll step through how to setup each connector. If you only need to integrate with a specific application, skip ahead to that section.

Salesforce (Using SCIM v1.1)

In this example, we'll show you how to provision users from one Salesforce org to another Salesforce org using the SCIM v1.1 connector. To do so, you'll need two Salesforce orgs. We'll call the host organization where you installed the connector packages "Org 1", and the target system organization "Org 2."

If you don't have a second org, signup for a free Developer Edition org here: https://developer.salesforce.com/signup.

After you have your second org, setup My Domain on the org using the steps above before moving forward.

In Org 1:

Setup the Connected App

1. Click the **Setup** link in the upper-right corner.
2. Navigation to **Administer > Security Controls > Auth. Providers**.
3. Select the **SCIM** Auth Provider and copy the value for the **Callback URL** listed at the bottom of the page.
4. Navigate to **Build > Create > Apps**.
5. Create a **new** connected app with the following information:
   * Connected App Name: Org2 (or any vanity name you'd like to call this org)
   * API Name: Org2 (or any vanity name you'd like to call this org)
   * Contact Email: *your email address*
   * Enable OAuth Settings: True
   * Callback URL: *the URL you copied in step 2*
   * Scopes:
     * Access your basic information (id, profile, email, address, phone)
     * Access and manage your data (api)
     * Perform requests on your behalf at any time (refresh_token, offline_access)
6. **Save**.
7. Take note of the **Consumer Key** and **Consumer Secret** values. You'll need them for the next step.

Setup the Auth Provider

1. Click the **Setup** link in the upper-right corner.
2. Navigation to **Administer > Security Controls > Auth. Providers**.
3. Create a **new** Auth. Provider with the following information:
    * Provider Type: Salesforce (for Salesforce) or OpenID Connect (for OpenID Connect compliant applications)
    * Name: SCIM
    * URL Suffix: SCIM
    * Consumer Key: *Copied from the Org2 connected app created above*
    * Consumer Secret: *Copied from the Org2 connected app created above*
    * Authorize Endpoint URL:
        `https://`*MY DOMAIN OF ORG 2*`/services/oauth2/authorize`
    * Token Endpoint URL:
        `https://`*MY DOMAIN OF ORG 2*`/services/oauth2/token`
4. **Save** the Auth Provider, and make a copy of the Callback URL at the bottom of the page.
5. On Org 2, edit the OAuth settings for the Connect App you created for this integration, and set the callback URL as the callback URL you just copied from the new Auth Provider.

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **SCIM** Named Credential to reflect the following information:
    * Label: SCIM
    * Name: SCIM
    * URL: `https://`*MY DOMAIN OF ORG 2*`/services/scim/v1/Users`
    * Identity Type: Named Principal
    * Authentication Protocol: OAuth 2.0
    * Authentication Provider: SCIM (the Auth Provider you edited above)
    * Start Authentication Flow on Save: Checked
3. On saving, you will be redirected to login to Org2. Login with a user that has admin rights to manage Org2's users. When prompted, authorize Org1 for the scopes that are listed.

Setup the Custom Setting

A custom setting was provided with the installed package. This is where you can change configuration options for the connector. The option supported for SCIM is:

- Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object

To access this custom setting:

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select **Box**.
3. Select **Manage**.
4. **Edit** and **Save** the specific settings you wish to change.

Setup the Flow Variables

Use the Flow feature to link provisioning orchestration events to the corresponding Apex code to complete the changes. A sample flow has been provided. These steps help you complete the flow configuration to meet your specific user case.

1. Navigate to **Create > Workflows & Approvals > Flows**.
2. Click the **SCIM Users Flow** and **Open** next to the **Active** flow version.
3. Click the **Explorer** tab in the Flow Designer.
4. Scroll down until you reach the Formulas menu item.
5. Modify the following two formulas to meet your specific needs: UserNameFormula and EntitlementFormula.
    a. UsernameFormula returns the target system username that's assigned to new accounts upon creation. As an example, you can transform a user's current username by substituting Org1's suffix with the suffix used on Org2: substitute ({!User.Username}, '@Org1.com', '@Org2.com').
    b. EntitlementFormula returns the ProfileId for the Org2 profile that should be assigned to new users upon creation. For Example, if you want to assign the "Identity Only" profile to all new users created on Org2, run the following query on Org2: SELECT Id FROM Profile where Name='Identity Only'. Copy the resulting Id to this formula, replacing the current Id value.

You can also create more advanced formulas to meet your specific use cases. For more information, review the following documentation: https://help.salesforce.com/apex/HTViewHelpDoc?id=customize_formuladef.htm&language=en_US.

6. Click **Save As** and ensure the new flow is being saved as type **User Provisioning Flow**.
7. Click **Okay**, **Close**, then **Okay**.
8. Find the new version of the flow in the list view and click **Activate** to activate the new flow version.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **Org2** connected app created in the steps above.
3. Click **Edit**.

4. Place a checkmark next to **Enable User Provisioning.**
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as an Org2 user. **Note: As users are assigned access to this Org2 connected app for the first time, Org1 will create an Org2 user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Org1 will remove the user from Org2. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Org1 users, defer this change until after completing Step 8**.
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **SCIM Users Flow** (**Note**: If you do not see the Flow ensure it is activated).
        ii. Pick a named credential: **SCIM**.
        iii. **Save & Next**.
    b. On the *Approvals page*
        i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
        ii. Save & Next.
    c. On the *Automated Operations* page
        i. Choose all the operations you would like to support for Org2.
        ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to Org2 on when they change in Org1. **Note**: If you want to update Org2 when a users name changes, choose the *Name* attribute, not FirstName and LastName.
        iii. **Save & Next**.
    d. On the *Collect* page
        i. Click **Connect and Collect**. This will connect to Org2 and pull a list of all Org2 users into Salesforce.
        ii. Once the list is collected, you'll need to link the Org2 users to existing Salesforce users. That way Salesforce knows what Org2 account to update and disable if the corresponding user is updated or disabled in Org1. Click the **Save & Analyze** button to start this process.
    e. On the *Analyze* Page
        i. Pick the attribute from Org1 and Org2 you'd like to use to link the accounts. For example, Email. This will perform an exact string comparison and link any account that matches.
        ii. Click **Analyze.**
        iii. Click **Analyze Results**.
        iv. Click **Commit**.

You've successfully setup the SCIM Connector. Now go to the "Testing your Connector" section of this document to test the connector.

Box

This section describes how to integrate Salesforce with Box.com. The scope of the document is specific to the Salesforce side of the integration. We assume that you have a Box.com account already. You'll also need to configure Box for OAuth authentication, and ensure that you've issued the "Create and manage app users" scope to the configuration. For more information, see the box.com OAuth documentation at `https://developers.box.com/oauth/`.

In Salesforce:

Setup the Auth Provider

4. Click the **Setup** link in the upper-right corner.
5. Navigation to **Administer > Security Controls > Auth. Providers**.
6. Create a **new** Auth. Provider with the following information:
   * Provider Type: OpenID Connect
   * Name: Box
   * URL Suffix: Box
   * Consumer Key: *Found at* `https://app.box.com/developers/services`
   * Consumer Secret: *Found at* `https://app.box.com/developers/services`
   * Authorize Endpoint URL: `https://app.box.com/api/oauth2/authorize`
   * Token Endpoint URL: `https://app.box.com/api/oauth2/token`
6. **Save** the Auth Provider, and make a copy of the Callback URL at the bottom of the page.
7. On Box.com, edit the OAuth settings for the App you created for this integration, and set the redirect URL in Box as the callback URL you just copied from the new Auth Provider.

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:
   * Label: Box
   * Name: Box
   * URL: `https://api.box.com/2.0/users`
   * Identity Type: Named Principal
   * Authentication Protocol: OAuth 2.0
   * Authentication Provider: Box (the Auth Provider you created above)
   * Start Authentication Flow on Save: Checked

3. On saving, you will be redirected to login to Box.com. Login with a user that has admin rights to manage Box users. When prompted, authorize Salesforce for the scopes that are listed.

Setup the Custom Setting

A custom setting was provided with the installed package. This is where you can change configuration options for the connector. The options supported for Box are:

- Force Delete Users (Optional): Delete users even if they have data / files associated with their Box.com account
- Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object

To access this custom setting:

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select **Box**.
3. Select **Manage**.
4. **Edit** and **Save** the specific settings you wish to change

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: Box
   * API Name: Box
   * Contact Email: *your email address*
3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **Box** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as a Box user. **Note: As users are assigned access to this Box connected app for the first time, Salesforce will create a Box**

user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from Box. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8.

8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **Box Users Flow** (**Note**: If you do not see the Flow ensure it is activated.)
        ii. Pick a named credential: **Box**.
        iii. **Save & Next**.
    b. On the *Approvals page*
        i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
        ii. Save & Next.
    c. On the *Automated Operations* page
        i. Choose all the operations you would like to support for Box
        ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to Box on when they change in Salesforce. **Note**: If you want to update Box when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
        iii. **Save** & **Next**.
    d. On the *Collect* page
        i. Click **Connect and Collect**. This will connect to Box and pull a list of all Box users into Salesforce.
        ii. Once the list is collected, you'll need to link the Box users to existing Salesforce users. That way Salesforce knows what Box account to update and disable if the corresponding user is updated or disabled in Salesforce. Click the **Save & Analyze** button to start this process.
    e. On the *Analyze* Page
        i. Pick the attribute from Salesforce and Box you'd like to use to link the accounts. For example, Email. This will perform an exact string comparison and link any account that matches.
        ii. Click **Analyze**.
        iii. Click **Analyze Results**.
        iv. Click **Commit**.

You've successfully setup the Box Connector. Now go to the "Testing your Connector" section of this document to test the connector.

Concur

This section describes how to integrate Salesforce with Concur. The scope of the document is specific to the Salesforce side of the integration. We assume that you have a Concur account already. You'll also need to configure Concur for OAuth authentication. For more information, see the Concur OAuth documentation at https://developer.concur.com/oauth-20/oauth-20-overview and https://developer.concur.com/overview/partner-applications.

Note that this connector uses both [v1.0](https://developer.concur.com/docs-and-resources/documentation) and [v3.0](https://www.concursolutions.com/api/docs/index.html) APIs from Concur. The v1.0 API is limited to XML and does not have documented pagination support; maximum results are limited to 500. Concur's v3.0 APIs feature JSON responses and pagination but offer limited support for manipulating data. So we must fall back on the v1.0 API (for now) to create and update users.

In Salesforce:

Setup the Named Credential

The Concur API currently uses a non-standard OAuth 2.0 implementation, requiring a non-standard setup within Salesforce. As such, the below Named Credential is used only for authenticating with the Concur API. Access to User resources is controlled by the settings defined in the **Setup the Custom Settings** section.

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:
   * Label: Concur
   * Name: Concur
   * URL: https://www.concursolutions.com/net2/oauth2/accesstoken.ashx
   * Identity Type: Named Principal
   * Authentication Protocol: Password Authentication
   * Username: *your Concur username*
   * Password: *your Concur password*
3. **Save**.

Setup the Custom Setting

A custom setting was provided with the installed package. This is where you can change configuration options for the connector.

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select **Concur**.
3. Select **Manage**.
4. Set values for the following required field(s):
   - User Endpoint v1 (Required): URL for the User endpoint in v1.0 of the Concur API - e.g. https://www.concursolutions.com/api/user/v1.0/
   - User Endpoint v3 (Required): URL for the User endpoint in v3.0 of the Concur API - e.g. `https://www.concursolutions.com/api/v3.0/common/users`
   - Partner Application Key (Required): The key (not secret) for the Partner Application defined in the Concur Web Services settings *Found at* https://www.concursolutions.com/companyadmin/partnerapp/registration.asp
   - Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object.
5. **Save**.

Setup the Flow Variables

Creating a user via the Concur API requires providing a password for the newly created user. The default password for these accounts must be set in the Flow provided for User Provisioning.

1. Navigate to **Create > Workflows & Approvals > Flows**.
2. Click the **Concur Users Flow** and **Open** next to the **Active** flow version.
3. Click the **Explorer** tab in the Flow Designer.
4. Double-click the **DefaultPassword** entry under the **VARIABLES** heading.
5. Assign a value for **Default Value** and click **OK**.
6. Click **Save As** and ensure the new flow is being saved as type **User Provisioning Flow**.
7. Click **Okay**, **Close, Okay**.
8. Find the new version of the flow in the list view and click **Activate** to activate the new flow version.

*Note that this is a basic way to assign the same default password to all new Concur accounts. However, this default password is accessible to any Salesforce user with **Edit Flow** permissions. For added security, you should write custom Apex to generate a secure, random password that is sent to the user upon successful creation of their account.*

Setup the Remote Site Settings

The Concur API currently uses a non-standard OAuth 2.0 implementation, requiring a non-standard setup within Salesforce. As such, Named Credentials are not used for accessing User resources, requiring the below Remote Site definition.

1. Navigate to **Administer > Security Controls > Remote Site Settings**.
2. Click the **New Remote Site** button and set the following values:
   * Remote Site Name: ConcurSolutions
   * Remote Site URL: `https://www.concursolutions.com/`
3. **Save** the Remote Site settings.

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: Concur
   * API Name: Concur
   * Contact Email: *your email address*
3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **Concur** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as a Concur user. **Note: s users are assigned access to this Concur connected app for the first time, Salesforce will create a Concur user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from Concur. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8**.
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **Concur Users Flow** (**Note**: If you do not see the Flow ensure it is activated.)

ii. Pick a named credential: **Concur**.
iii. **Save & Next**.

b. On the *Approvals page*
i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
ii. Save & Next.

c. On the *Automated Operations* page
i. Choose all the operations you would like to support for Concur.
ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to Concur on when they change in Salesforce. **Note**: If you want to update Concur when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
iii. **Save & Next**.

d. On the *Collect* page
i. Click the **Connect and Collect** button. This will connect to Concur and pull a list of all Concur users into Salesforce. If successful, you should see "Total User accounts found: …"
ii. You'll now need to link the Concur users to existing Salesforce users. That way Salesforce knows what Concur account to update and disable if the corresponding user is updated or disabled in Salesforce. Click the **Save & Next** button to start this process.

e. On the *Analyze* Page
i. Pick the attribute from Salesforce and Concur you'd like to use to link the accounts. For example, Email. This will perform an exact string comparison and link any account that matches.
ii. Click **Analyze Collected Information**.
iii. Once complete, this provides you with a count of the users linked from the target system.
iv. Click **Commit, Next, Finish.**

You've successfully setup the Concur Connector. Now go to the "Testing your Connector" section of this document to test the connector.

Dropbox

This section describes how to integrate Salesforce with Dropbox. The scope of the document is specific to the Salesforce side of the integration. We assume that you have a Dropbox account already. You'll also need to configure Dropbox for OAuth authentication by creating a **Business App** at `https://www.dropbox.com/developers/apps/create` and ensuring that you've issued the **Team member management** scope to the configuration. For more information, see the Dropbox OAuth documentation at `https://www.dropbox.com/developers/reference/oauthguide` and `https://www.dropbox.com/developers/business`.

In Salesforce:

Setup the Auth Provider

1. Click the **Setup** link in the upper-right corner.
2. Navigation to **Administer > Security Controls > Auth. Providers**.
3. Create a **new** Auth. Provider with the following information:
   * Provider Type: OpenID Connect
   * Name: Dropbox
   * URL Suffix: Dropbox
   * Consumer Key: *Found at* `https://www.dropbox.com/developers/apps`
   * Consumer Secret: *Found at* `https://www.dropbox.com/developers/apps`
   * Authorize Endpoint URL: `https://www.dropbox.com/1/oauth2/authorize`
   * Token Endpoint URL: `https://api.dropboxapi.com/1/oauth2/token`
4. **Save** the Auth Provider, and make a copy of the Callback URL at the bottom of the page.
5. On Dropbox, edit the OAuth settings for the Business App you created for this integration, and set the **Redirect URIs** in Dropbox as the callback URL you just copied from the new Auth Provider.

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:
   * Label: Dropbox
   * Name: Dropbox
   * URL: `https://api.dropbox.com/1/team/members/`
   * Identity Type: Named Principal
   * Authentication Protocol: OAuth 2.0
   * Authentication Provider: Dropbox (the Auth Provider you created above)
   * Start Authentication Flow on Save: Checked
4. On saving, you will be redirected to login to Dropbox. Login with a user that has admin rights to manage Dropbox users. When prompted, authorize Salesforce for the scopes that are listed.

Setup the Custom Settings

A custom setting was provided with the installed package. This is where you can change configuration options for the connector. The options supported for Dropbox are:

- Transfer Destination Member ID (Optional): Files from a deleted member account will be transferred to this member

- Transfer Admin Member ID (Optional): Errors during the transfer process will be sent via email to this member.
- Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object

To access this custom setting:

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select **Dropbox**.
3. Select **Manage**.
4. **Edit** and **Save** the specific settings you wish to change.

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: Dropbox
   * API Name: Dropbox
   * Contact Email: *your email address*
3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **Dropbox** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as a Dropbox user. **Note: As users are assigned access to this Dropbox connected app for the first time, Salesforce will create a Dropbox user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from Dropbox. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8**.
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **Dropbox Users Flow** (**Note**: If you do not see the Flow ensure it is activated.)

ii. Pick a named credential: **Dropbox**.
iii. **Save & Next**.

b. On the *Approvals page*
   i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
   ii. **Save & Next**.

c. On the *Automated Operations* page
   i. Choose all the operations you would like to support for Dropbox.
   ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to Dropbox on when they change in Salesforce. **Note**: If you want to update Dropbox when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
   iii. **Save & Next**.

d. On the *Collect* page
   i. Click the **Connect and Collect** button. This will connect to Dropbox and pull a list of all Dropbox users into Salesforce. If successful, you should see "Total User accounts found: …"
   ii. You'll now need to link the Dropbox users to existing Salesforce users. That way Salesforce knows what Dropbox account to update and disable if the corresponding user is updated or disabled in Salesforce. Click the **Save & Next** button to start this process.

e. On the *Analyze* Page
   i. Pick the attribute from Salesforce and Dropbox you'd like to use to link the accounts. For example, Email. This will perform an exact string comparison and link any account that matches.
   ii. Click **Analyze Collected Information**.
   iii. Once complete, this provides you with a count of the users linked from the target system.
   iv. Click **Commit, Next, Finish**.

You've successfully setup the Dropbox Connector. Now go to the "Testing your Connector" section of this document to test the connector.

Google

This section describes how to integrate Salesforce with Google. The scope of the document is specific to the Salesforce side of the integration. We assume that you have a Google account already. You'll also need to configure Google for OAuth authentication using the "Web Application" application type, and ensuring you've enabled the "Admin SDK" API. For more information, see the Google OAuth documentation at https://developers.google.com/admin-sdk/directory/v1/guides/authorizing and https://developers.google.com/identity/protocols/OAuth2.

In Salesforce:

Setup the Auth Provider

1. Click the **Setup** link in the upper-right corner.
2. Navigation to **Administer > Security Controls > Auth. Providers**.
3. Create a **new** Auth. Provider with the following information:
   * Provider Type: OpenID Connect
   * Name: Google
   * URL Suffix: Google
   * Consumer Key: *Found at* `https://console.developers.google.com/project/`
   * Consumer Secret: *Found at* `https://console.developers.google.com/project/`
   * Authorize Endpoint URL: `https://accounts.google.com/o/oauth2/auth?access_type=offline&approval_prompt=force`
   * Token Endpoint URL: `https://www.googleapis.com/oauth2/v3/token`
   * Default Scopes: `https://www.googleapis.com/auth/admin.directory.user`
4. **Save** the Auth Provider, and make a copy of the Callback URL at the bottom of the page.
5. On Google, edit the OAuth settings for the App you created for this integration, and set the Authorized Redirect URIs in Google as the callback URL you just copied from the new Auth Provider.

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:
   * Label: Google
   * Name: Google
   * URL: `https://www.googleapis.com/admin/directory/v1/users/`
   * Identity Type: Named Principal
   * Authentication Protocol: OAuth 2.0
   * Authentication Provider: Google (the Auth Provider you created above)
   * Start Authentication Flow on Save: Checked
3. On saving, you will be redirected to login to Google. Login with a user that has admin rights to manage Google users. When prompted, authorize Salesforce for the scopes that are listed.

Setup the Custom Setting

A custom setting was provided with the installed package. This is where you can change configuration options for the connector.

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select Google.
3. Select **Manage**.
4. Set values for the required field(s):
   - Company Domain (Required): The company domain within Google Apps to be managed
   - Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object
5. **Save**.

Setup the Flow Variables

Creating a user via the Google API requires providing a password for the newly created user. The default password for these accounts must be set in the Flow provided for User Provisioning.

1. Navigate to **Create > Workflows & Approvals > Flows**.
2. Click the **Google Users Flow** and **Open** next to the **Active** flow version.
3. Click the **Explorer** tab in the Flow Designer.
4. Double-click the **DefaultPassword** entry under the **VARIABLES** heading.
5. Assign a value for **Default Value** and click **OK**.
6. Click **Save As** and ensure the new flow is being saved as type **User Provisioning Flow**.
7. Click **Okay**, **Close, Okay**.
8. Find the new version of the flow in the list view and click **Activate** to activate the new flow version.

*Note that this is a basic way to assign the same default password to all new Google accounts. However, this default password is accessible to any Salesforce user with **Edit Flow** permissions. For added security, you should write custom Apex to generate a secure, random password that is sent to the user upon successful creation of their account.*

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: Google (can also be a specific Google app: Gmail, etc)
   * API Name: Google (can also be a specific Google app: Gmail, etc)
   * Contact Email: *your email address*

3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **Google** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as a Google user. **Note: As users are assigned access to this Google connected app for the first time, Salesforce will create a Google user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from Google. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8**.
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **Google Users Flow** (**Note**: If you don't see the Flow ensure it is activated.)
        ii. Pick a named credential: **Google**.
        iii. **Save & Next**.
    b. On the *Approvals page*
        i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
        ii. **Save & Next**.
    c. On the *Automated Operations* page
        i. Choose all the operations you would like to support for Google.
        ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to Google on when they change in Salesforce. **Note**: If you want to update Google when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
        iii. **Save & Next**.
    d. On the *Collect* page
        i. Click **Connect and Collect** button. This will connect to Google and pull a list of all Google users into Salesforce. If successful, you should see "Total User accounts found: …"

ii. You'll now need to link the Google users to existing Salesforce users. That way Salesforce knows what Google account to update and disable if the corresponding user is updated or disabled in Salesforce. Click the **Save & Next** button to start this process.

e. On the *Analyze* Page
   i. Pick the attribute from Salesforce and Google you'd like to use to link the accounts. For example, Email. This will perform an exact string comparison and link any account that matches.
   ii. Click **Analyze Collected Information**.
   iii. Once complete, this provides you with a count of the users linked from the target system.
   iv. Click **Commit, Next, Finish**.

You've successfully setup the Google Connector. Now go to the "Testing your Connector" section of this document to test the connector.

GoToMeeting

This section describes how to integrate Salesforce with GoToMeeting. The scope of the document is specific to the Salesforce side of the integration. We assume that you have a GoToMeeting account already. You'll also need to configure GoToMeeting for OAuth authentication. For more information, see the GoToMeeting OAuth documentation at `https://developer.citrixonline.com/oauth`.

In Salesforce:

Setup the Auth Provider

1. Click the **Setup** link in the upper-right corner.
2. Navigation to **Administer > Security Controls > Auth. Providers**.
3. Create a **new** Auth. Provider with the following information:
   * Provider Type: OpenID Connect
   * Name: GoToMeeting
   * URL Suffix: GoToMeeting
   * Consumer Key: *Found at* `https://developer.citrixonline.com`
   * Consumer Secret: *Found at* `https://developer.citrixonline.com`
   * Authorize Endpoint URL: `https://api.citrixonline.com/oauth/authorize`
   * Token Endpoint URL:
`https://api.citrixonline.com/oauth/access_token`
4. **Save** the Auth Provider, and make a copy of the Callback URL at the bottom of the page.

5. On GoToMeeting, edit the OAuth settings for the App you created for this integration, and set the Application URL in GoToMeeting as the callback URL you just copied from the new Auth Provider.

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:
   * Label: GoToMeeting
   * Name: GoToMeeting
   * URL: `https://api.citrixonline.com/G2M/rest/organizers`
   * Identity Type: Named Principal
   * Authentication Protocol: OAuth 2.0
   * Authentication Provider: GoToMeeting (the Auth Provider you created above)
   * Start Authentication Flow on Save: Checked
3. On saving, you will be redirected to login to GoToMeeting. Login with a user that has admin rights to manage GoToMeeting users. When prompted, authorize Salesforce for the scopes that are listed.

Setup the Custom Settings

A custom setting was provided with the installed package. This is where you can change configuration options for the connector. The options supported for GoToMeeting are:

- Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object

To access this custom setting:

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select **Box**.
3. Select **Manage**.
4. **Edit** and **Save** the specific settings you wish to change.

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: GoToMeeting
   * API Name: GoToMeeting
   * Contact Email: *your email address*
3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **GoToMeeting** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as a GoToMeeting user. **Note: As users are assigned access to this GoToMeeting connected app for the first time, Salesforce will create a GoToMeeting user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from GoToMeeting. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8.**
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **GoToMeeting Users Flow**. (**Note**: If you do not see the Flow ensure it is activated.)
        ii. Pick a named credential: **GoToMeeting**.
        iii. **Save & Next**.
    b. On the *Approvals page*
        i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
        ii. **Save & Next**.
    c. On the *Automated Operations* page
        i. Choose all the operations you would like to support for GoToMeeting.
        ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to GoToMeeting on when they change in Salesforce. **Note**: If you want to update GoToMeeting when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
        iii. **Save & Next**.
    d. On the *Collect* page
        i. Click **Connect and Collect**. This will connect to GoToMeeting and pull a list of all GoToMeeting users into Salesforce. If successful, you should see "Total User accounts found: …"
        ii. You'll now need to link the GoToMeeting users to existing Salesforce users. That way Salesforce knows what GoToMeeting account to update

and disable if the corresponding user is updated or disabled in Salesforce. Click the **Save & Next** button to start this process.

e. On the *Analyze* Page
   i. Pick the attribute from Salesforce and GoToMeeting you'd like to use to link the accounts. For example, Email. This will perform an exact string comparison and link any account that matches.
   ii. Click **Analyze Collected Information**.
   iii. Once complete, this provides you with a count of the users linked from the target system.
   iv. Click **Commit, Next, Finish**.

You've successfully setup the GoToMeeting Connector. Now go to the "Testing your Connector" section of this document to test the connector.

Office 365

This section describes how to integrate Salesforce with Office 365. The scope of the document is specific to the Salesforce side of the integration. We assume that you have an Office 365 / Azure account already. You'll also need to configure Azure for OAuth authentication, and ensure that you've issued the "Read and write directory data" delegated permission to the configuration. For more information, see the Azure OAuth documentation at https://msdn.microsoft.com/en-us/library/azure/dn645542.aspx and https://azure.microsoft.com/en-gb/documentation/articles/active-directory-integrating-applications/.

In Salesforce:

Setup the Auth Provider

1. Click the **Setup** link in the upper-right corner.
2. Navigation to **Administer > Security Controls > Auth. Providers**.
3. Create a **new** Auth. Provider with the following information:
   * Provider Type: OpenID Connect
   * Name: Office 365
   * URL Suffix: Office_365
   * Consumer Key: *Found at* `https://manage.windowsazure.com/`
   * Consumer Secret: *Found at* `https://manage.windowsazure.com/`
   * Authorize Endpoint URL: `https://login.microsoftonline.com/`*tenant id*`/oauth2/authorize?resource=https://graph.windows.net`
   * Token Endpoint URL: `https://login.microsoftonline.com/<tenant id>/oauth2/token`

4. **Save** the Auth Provider, and make a copy of the Callback URL at the bottom of the page.
5. On Azure, edit the OAuth settings for the App you created for this integration, and set the Reply URL in Azure as the callback URL you just copied from the new Auth Provider.

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:
   * Label: Office 365
   * Name: Office_365
   * URL: `https://graph.windows.net/<tenant domain>/users`
   * Identity Type: Named Principal
   * Authentication Protocol: OAuth 2.0
   * Authentication Provider: Office 365 (the Auth Provider you created above)
   * Start Authentication Flow on Save: Checked
3. On saving, you will be redirected to login to Azure. Login with a user that has admin rights to manage Azure users. When prompted, authorize Salesforce for the scopes that are listed.

A custom setting was provided with the installed package. This is where you can change configuration options for the connector.

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select Office 365.
3. Select **Manage**.
4. Set values for the required field(s):
   - Company Domain (Required): The company domain within Office 365 to be managed
   - Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object.
5. **Save**.

Setup the Flow Variables

Creating a user via the Office 365 API requires providing a password for the newly created user. The default password for these accounts must be set in the Flow provided for User Provisioning.

1. Navigate to **Create > Workflows & Approvals > Flows**.
2. Click the **Office365 Users Flow** and **Open** next to the **Active** flow version.
3. Click the **Explorer** tab in the Flow Designer.
4. Double-click the **DefaultPassword** entry under the **VARIABLES** heading.
5. Assign a value for **Default Value** and click **OK**.

6. Click **Save As** and ensure the new flow is being saved as type **User Provisioning Flow**.
7. Click **Okay**, **Close, Okay**.
8. Find the new version of the flow in the list view and click **Activate** to activate the new flow version.

*Note that this is a basic way to assign the same default password to all new Office365 accounts. However, this default password is accessible to any Salesforce user with **Edit Flow** permissions. For added security, you should write custom Apex to generate a secure, random password that is sent to the user upon successful creation of their account.*

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: Office 365
   * API Name: Office_365
   * Contact Email: *your email address*
3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **Office 365** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as an Office 365 user. **Note: As users are assigned access to this Office 365 connected app for the first time, Salesforce will create an Office 365 user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from Office 365. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8**.
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **Office365 Users Flow** (**Note**: If you do not see the Flow ensure it is activated.)
        ii. Pick a named credential: **Office 365**.
        iii. **Save & Next**.

b. On the *Approvals page*
    i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
    ii. **Save & Next**.
c. On the *Automated Operations* page
    i. Choose all the operations you would like to support for Office 365
    ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to Office 365 on when they change in Salesforce. **Note**: If you want to update Office 365 when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
    iii. **Save & Next**.
d. On the *Collect* page
    i. Click **Connect and Collect**. This will connect to Office 365 and pull a list of all Office 365 users into Salesforce. If successful, you should see "Total User accounts found: …"
    ii. You'll now need to link the Office 365 users to existing Salesforce users. That way Salesforce knows what Office 365 account to update and disable if the corresponding user is updated or disabled in Salesforce. Click **Save & Next** to start this process.
e. On the *Analyze* Page
    i. Pick the attribute from Salesforce and Office 365 you'd like to use to link the accounts. For example, Email. This will perform an exact string. comparison and link any account that matches.
    ii. Click **Analyze Collected Information**.
    iii. Once complete, this provides you with a count of the users linked from the target system.
    iv. Click **Commit, Next, Finish**.

You've successfully setup the Office 365 Connector. Now go to the "Testing your Connector" section of this document to test the connector.

ServiceNow

This section describeshow to integrate Salesforce with ServiceNow. The scope of the document is specific to the Salesforce side of the integration. We assume that you have a ServiceNow account already. This connector also requires the Fuji ServiceNow release or newer.

In Salesforce:

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:
   * Label: ServiceNow
   * Name: ServiceNow
   * URL:

https://<instance-name>.service-now.com/api/now/v1/table/sys_user
   * Identity Type: Named Principal
   * Authentication Protocol: Password Authentication
   * Username: `your ServiceNow username`
   * Password: `your ServiceNow password`
3. **Save**.

Note: ServiceNow account you use must have the **security_admin** role assigned.

Setup the Custom Settings

A custom setting was provided with the installed package. This is where you can change configuration options for the connector. The option supported by ServiceNow is:

- Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object

To access this custom setting:

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select **ServiceNow**.
3. Select **Manage**.
4. **Edit** and **Save** the specific settings you wish to change.

Setup the Flow Variables

Creating a user via the ServiceNow API requires providing a password for the newly created user. The default password for these accounts must be set in the Flow provided for User Provisioning.

1. Navigate to **Create > Workflows & Approvals > Flows**.
2. Click the **ServiceNow Users Flow** and **open** next to the **Active** version.
3. Click the **Explorer** tab in the Flow Designer.
4. Double-click the **DefaultPassword** entry under the **VARIABLES** heading.
5. Assign a value for **Default Value** and click **OK**.
6. Click **Save As** and ensure the new flow is being saved as type **User Provisioning Flow**.

7. Click **Okay**, **Close, Okay**.
8. Find the new version of the flow in the list view and click **Activate** to activate the new flow version.

*Note that this is a basic way to assign the same default password to all new Office365 accounts. However, this default password is accessible to any Salesforce user with **Edit Flow** permissions. For added security, you should write custom Apex to generate a secure, random password that is sent to the user upon successful creation of their account.*

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: ServiceNow
   * API Name: ServiceNow
   * Contact Email: *your email address*
3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **ServiceNow** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as a ServiceNow user. **Note: As users are assigned access to this ServiceNow connected app for the first time, Salesforce will create a ServiceNow user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from ServiceNow. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8**.
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **ServiceNow Users Flow** (**Note**: If you do not see the Flow ensure it is activated.)
        ii. Pick a named credential: **ServiceNow**.
        iii. **Save & Next**.
    b. On the *Approvals page*

i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
ii. **Save & Next**.

c. On the *Automated Operations* page
i. Choose all the operations you would like to support for ServiceNow.
ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to ServiceNow on when they change in Salesforce. Note: If you want to update ServiceNow when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
iii. **Save & Next**.

d. On the *Collect* page
i. Click **Connect and Collect**. This will connect to ServiceNow and pull a list of all ServiceNow users into Salesforce. If successful, you should see "Total User accounts found: …"
ii. You'll now need to link the ServiceNow users to existing Salesforce users. That way Salesforce knows what ServiceNow account to update and disable if the corresponding user is updated or disabled in Salesforce. Click **Save & Next** to start this process.

e. On the *Analyze* Page
i. Pick the attribute from Salesforce and ServiceNow you'd like to use to link the accounts. For example, Email. This will perform an exact string. comparison and link any account that matches.
ii. Click **Analyze Collected Information**.
iii. Once complete, this provides you with a count of the users linked from the target system.
iv. Click **Commit, Next, Finish**.

You've successfully setup the ServiceNow Connector. Now go to the "Testing your Connector" section of this document to test the connector.

WebEx

This section describes how to integrate Salesforce with WebEx. The scope of the document is specific to the Salesforce side of the integration. We assume that you have a WebEx account already.

In Salesforce:

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:

* Label: WebEx
* Name: WebEx
* URL: `https://<site name>.webex.com/WBXService/XMLService`
* Identity Type: Named Principal
* Authentication Protocol: Authentication
* Username: *your WebEx username*
* Password: *your WebEx password*
* Allow Merge Fields in HTTP Body: Checked

3. **Save**.

Setup the Custom Setting

A custom setting was provided with the installed package. This is where you can change configuration options for the connector.

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select WebEx.
3. Select **Manage**.
4. Set values for the required field(s):
   - Site Name (Required): The name of the WebEx site being managed - e.g. *acme* if your WebEx URL is acme.webex.com
   - Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object
5. **Save**.

Setup the Flow Variables

Creating a user via the WebEx API requires providing a password for the newly created user. The default password for these accounts must be set in the Flow provided for User Provisioning.

1. Navigate to **Create > Workflows & Approvals > Flows**.
2. Click the **Webex Users Flow** and **Open** next to the **Active** flow version.
3. Click the **Explorer** tab in the Flow Designer.
4. Double-click the **DefaultPassword** entry under the **VARIABLES** heading.
5. Assign a value for **Default Value** and click **OK**. **Note**: Webex requires a mixed-case password.
6. Click **Save As** and ensure the new flow is being saved as type **User Provisioning Flow**.
7. Click **Okay**, **Close, Okay**.
8. Find the new version of the flow in the list view and click **Activate** to activate the new flow version.

*Note that this is a basic way to assign the same default password to all new WebEx accounts. However, this default password is accessible to any Salesforce user with **Edit Flow***

permissions. For added security, you should write custom Apex to generate a secure, random password that is sent to the user upon successful creation of their account.

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: WebEx
   * API Name: WebEx
   * Contact Email: *your email address*
3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **WebEx** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as a WebEx user. **Note: As users are assigned access to this WebEx connected app for the first time, Salesforce will create a WebEx user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from WebEx. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8**.
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **WebEx Users Flow** (**Note**: If you do not see the Flow ensure it is activated.)
        ii. Pick a named credential: **WebEx**.
        iii. **Save & Next**.
    b. On the *Approvals page*
        i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
        ii. **Save & Next**.
    c. On the *Automated Operations* page
        i. Choose all the operations you would like to support for WebEx

ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to WebEx on when they change in Salesforce. **Note**: If you want to update WebEx when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
iii. **Save & Next**.

d. On the *Collect* page
i. Click **Connect and Collect**. This will connect to WebEx and pull a list of all WebEx users into Salesforce. If successful, you should see "Total User accounts found: …"
ii. You'll now need to link the WebEx users to existing Salesforce users. That way Salesforce knows what WebEx account to update and disable if the corresponding user is updated or disabled in Salesforce. Click the **Save & Next** button to start this process.

e. On the *Analyze* Page
i. Pick the attribute from Salesforce and WebEx you'd like to use to link the accounts. For example, Email. This will perform an exact string comparison and link any account that matches.
ii. Click **Analyze Collected Information**.
iii. Once complete, this provides you with a count of the users linked from the target system
iv. Click **Commit, Next, Finish**

You've successfully setup the WebEx Connector. Now go to the "Testing your Connector" section of this document to test the connector.

ZenDesk

This section describes how to integrate Salesforce with Zendesk. The scope of the document is specific to the Salesforce side of the integration. We assume that you have a Zendesk account already. You'll also need to configure Zendesk for OAuth authentication, and ensure that you make note of the Client Secret in the process as it will only be shown once. For more information, see the Zendesk OAuth documentation at https://support.zendesk.com/hc/en-us/articles/203663836-Using-OAuth-authentication-with-your-application.

In Salesforce:

Setup the Auth Provider

1. Click the **Setup** link in the upper-right corner.
2. Navigation to **Administer > Security Controls > Auth. Providers**.
3. Create a **new** Auth. Provider with the following information:

* Provider Type: OpenID Connect
* Name: Zendesk
* URL Suffix: Zendesk
* Consumer Key: *Found at* `https://<subdomain>.zendesk.com/agent/admin/api`
* Consumer Secret: *noted during the integration setup in Zendesk*
* Authorize Endpoint URL: `https://<subdomain>.zendesk.com/oauth/authorizations/new`
* Token Endpoint URL: `https://<subdomain>.zendesk.com/oauth/tokens`

4. **Save** the Auth Provider, and make a copy of the Callback URL at the bottom of the page.
5. On Zendesk, edit the OAuth settings for the App you created for this integration, and set the Redirect URL in Zendesk as the callback URL you just copied from the new Auth Provider.

Setup the Named Credential

1. Navigate to **Setup > Administer > Security Controls > Named Credentials**.
2. Create a **new** Named Credential with the following information:
   * Label: Zendesk
   * Name: Zendesk
   * URL: `https://<subdomain>.zendesk.com/api/v2/`
   * Identity Type: Named Principal
   * Authentication Protocol: OAuth 2.0
   * Authentication Provider: Zendesk (the Auth Provider you created above)
   * Start Authentication Flow on Save: Checked
3. On saving, you will be redirected to login to Box.com. Login with a user that has admin rights to manage Zendesk users. When prompted, authorize Salesforce for the scopes that are listed.

Setup the Custom Settings

A custom setting was provided with the installed package. This is where you can change configuration options for the connector. The option supported for Zendesk is:

- Enable Debug Logging (Optional): The connector will log all debug activity to the UserProvisioningLog standard object

To access this custom setting:

1. Navigate to **Setup > Develop > Custom Settings**.
2. Select **Zendesk**.

3. Select **Manage**.
4. **Edit** and **Save** the specific settings you wish to change.

Setup the Connected App

1. Navigate to **Build > Create > Apps**.
2. Create a **new** connected app with the following information:
   * Connected App Name: Zendesk
   * API Name: Zendesk
   * Contact Email: *your email address*
3. **Save**.

Setup User Provisioning

1. Navigate to **Setup > Administer > Manage Apps > Connected Apps**.
2. Click on the **Zendesk** connected app created in the steps above.
3. Click **Edit**.
4. Place a checkmark next to **Enable User Provisioning**.
5. Change the **Permitted Users** OAuth policy to **Admin approved users are preauthorized**.
6. Click **Save**.
7. Click **Manage Permission Sets** *or* **Manage Profiles** and assign the profiles and permission sets you wish to authorize as a Zendesk user. **Note: As users are assigned access to this Zendesk connected app for the first time, Salesforce will create a Zendesk user for them. If they are removed from all permission sets and profiles that provide access to this connected app, Salesforce will remove the user from Zendesk. By completing this step, it will take effect for all future changes. If you wish for this change to affect all existing Salesforce users, defer this change until after completing Step 8.**
8. Click the **Launch User Provisioning Wizard** button. This will launch a flow to configure provisioning:
    a. On the *Connecting* page
        i. Select **Use an existing flow**: **Zendesk Users Flow** (**Note**: If you do not see the Flow ensure it is activated.)
        ii. Pick a named credential: **Zendesk**.
        iii. **Save & Next**.
    b. On the *Approvals page*
        i. Leave require approvals unchecked for now. Later if you'd like to require approvals, you can come back and enable this field.
        ii. **Save & Next**.
    c. On the *Automated Operations* page
        i. Choose all the operations you would like to support for Zendesk.

ii. If you selected the Update operation, choose which User object attributes you would like trigger an update to Zendesk on when they change in Salesforce. **Note**: If you want to update Zendesk when a user's name changes, choose the *Name* attribute, not FirstName and LastName.
iii. **Save & Next**.

d. On the *Collect* page
i. Click **Connect and Collect**. This will connect to Zendesk and pull a list of all Zendesk users into Salesforce. If successful, you should see "Total User accounts found: …"
ii. You'll now need to link the Zendesk users to existing Salesforce users. That way Salesforce knows what Zendesk account to update and disable if the corresponding user is updated or disabled in Salesforce. Click the **Save & Next** button to start this process.

e. On the *Analyze* Page
i. Pick the attribute from Salesforce and Zendesk you'd like to use to link the accounts. For example, Email. This will perform an exact string comparison and link any account that matches.
ii. Click **Analyze Collected Information**.
iii. Once complete, this provides you with a count of the users linked from the target system.
iv. Click **Commit, Next, Finish**.

You've successfully setup the Zendesk Connector. Now go to the "Testing your Connector" section of this document to test the connector.

Testing Your Connector

You've setup your connector and now you're ready to test. Here are a few steps to get you started:

1. Create a new User in Salesforce. Ensure that all required fields are populated, as well as the **Federation Id** field.
2. In step 7 of the **Setup User Provisioning** section of your connector, you linked a profile or permission set to your new Connected App. Assign that profile or permission set to the new user.
3. Check the target system to confirm that the user was created.
4. Back in Salesforce, update the user's **Firstname** and **Lastname**.
5. Check the target system to confirm that the user's name was changed.
6. Back in Salesforce, disable the user (or remove them from the profile or permission set).
7. Check the target system to confirm that the user is now disabled.
8. Back in Salesforce, continue testing each use case your integration requires.

Note that these changes occur asynchronously and may take a few minutes to complete.

Having Trouble?

In Lightning

Open the App Launcher and select the **Manage Provisioning Requests** link under the **All Items** section to get a list of all provisioning requests and their associated status. If the request is in a 'Failed' state, click the **Logs** button for more information. Once you correct the error, you can then click **Retry** on the record to reattempt the change. You can also enable debug logging for the connector through the following steps:

1. Go to Custom Settings (**Develop > Custom Settings)**.
2. Click **Manage** next to the **<AppName> Prov Connector** Custom Setting.
3. Click **Edit**.
4. Select **Enable Debug Logging**.
5. Click **Save**.

In Classic

Go to Reports, select the **User Provisioning Reports** folder, then open the "User Provisioning Logs" report. This report contains any known errors and can give you more information. You can also enable debug logging for the connector through the following steps:

1. Go to Custom Settings (**Develop > Custom Settings)**.
2. Click **Manage** next to the **<AppName> Prov Connector** Custom Setting.
3. Click **Edit**.
4. Select **Enable Debug Logging**.
5. Click **Save**.

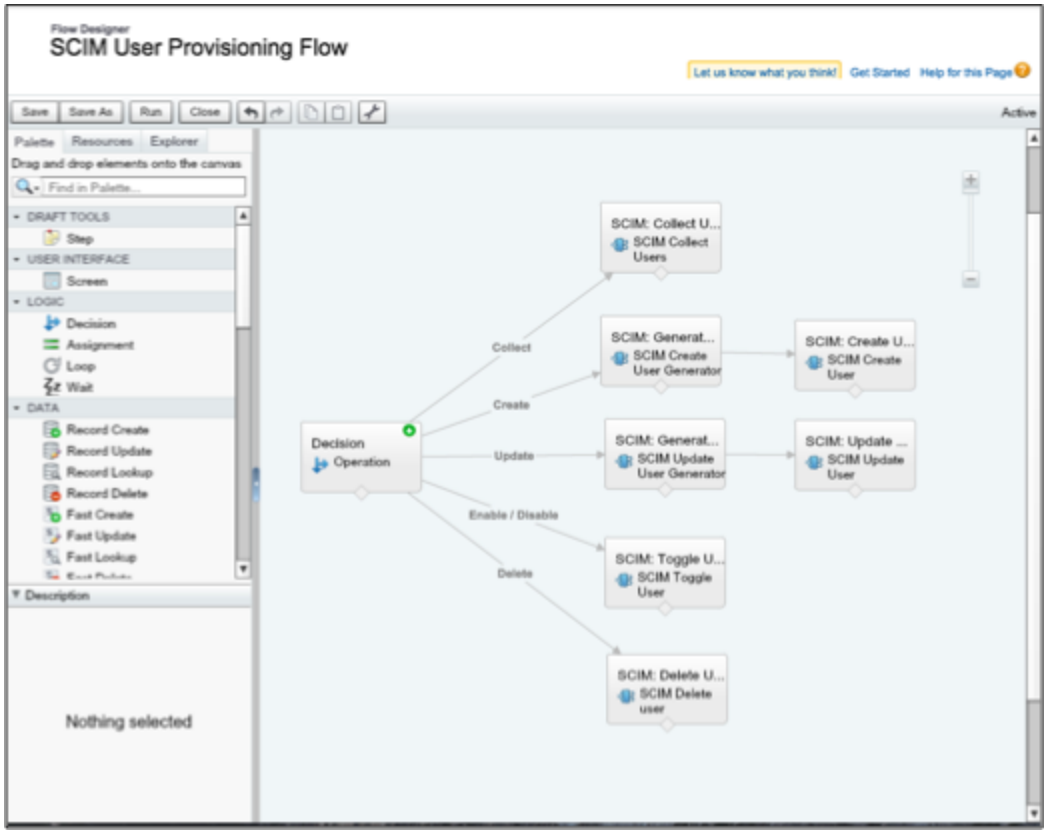
Additional Resources

For more information, see the following:

- Webinar: User Provisioning for Connected Apps
- Salesforce help: User Provisioning for Connected Apps
- Salesforce help: Create User Provisioning for Connected Apps Custom Reports

Advanced: Customize the Attributes Used by Your Provisioning Connector

This section is intended for developers who need to customize or extend the set of attributes managed on a target system from Salesforce. Before we dive into the details, let's review the overall design. We'll start with the user provisioning flow:



The flow is used to connect a provisioning event triggered in Salesforce to corresponding plugins or code that can perform the required actions against a specific target. In the example above, we see that these events correspond to different types of operations including Collect, Create, and Update. These operations are all tied to a different plugin. You can also see that the Create and Update operations use two plugins. The two plugins are used to separate the logic used to generate and map attributes to a third-party system from the logic used to actually interact with the third party and handle the response. So if you want to manage additional attributes or change how the attributes are derived, you only need to update the first plugin for the operation. Also, if you want to use the same set of attributes and logic for both the Create

and Update operations, you use the same code for both Attribute Generation plugins in your flow.

Generate Attribute Code Structure

This structure was designed such that the connectors are extensible, even if distributed as a managed package by our partners on App Exchange. There are three classes used to generate and map attributes between Salesforce and a third-party system. Depending on the changes you want to make, you may need to all three classes. Let's review each class in detail.

<Target>User.cls

Every connector comes with a <Target>User class. This class represents how each User attribute is structured on the third-party system (aka the User schema). Depending on which connector you're working on, to find the corresponding class, simply replace the <Target> keyword with the name of the connector, for example, `BoxUser.cls`, `GoogleUser.cls`, `WebExUser.cls`, and so on.

Here's an example SCIMUser.cls class from our SCIM v1.1 Connector:

```
global class SCIMUser {

    public class Meta {
        public String created;
        public String lastModified;
        public String version;
        public String location;
    }

    public class Roles {
        public String type;
        public String value;
        public String display;
        public Boolean primary;
    }

    public class Groups {
        public String type;
        public String value;
        public String display;
        public Boolean primary;
    }

    public class Entitlements {
        public String type;
        public String value;
```

```
    public String display;
    public Boolean primary;
}


public class Name {
    public String formatted;
    public String familyName;
    public String givenName;
}

public class Emails {
    public String type;
    public String display;
    public Boolean primary;
    public String value;
}


public class PhoneNumbers {
    public String type;
    public String value;
}

public class Photos {
    public String type;
    public String value;
}

// Public fields to serialize / deserialize
public List<String> schemas;
public String id;
public String userName;
public String externalId;
public Name name;
public String displayName;
public String nickName;
public List<Emails> emails;
public List<PhoneNumbers> phoneNumbers;
public List<Photos> photos;
public String userType;
public String preferredLanguage;
public String locale;
public Boolean active;
public String title;
public List<Entitlements> entitlements;
public List<Groups> groups;
public List<Roles> roles;
```

```
    public Meta meta;

    public SCIMUser() {
        this(null, null, null, null, null, null);
    }

    // Simple JSON deserialization
    public static SCIMUser parse(String json) {
        return (SCIMUser)System.JSON.deserialize(json, SCIMUser.class);
    }
}
```

Each connector supports a default set of attributes. If you're working within the set of attributes supported by your connector and simply want to change how the attribute values are set, you won't need to modify this class. However, if you need to extend the connector to support additional attributes, you should add them here first before moving forward.

<Target>UserAttributeGenerator.cls

Every connector comes with a `<Target>UserAttributeGenerator.cls` class.This class is used to derive and set each attribute value to a corresponding attribute on the target system's schema. For example, with this class, you can determine what value to set for the user's first name and assign it to the FirstName field on the target. This class takes an instance of `Process.PluginRequest` (the values passed into the plugin) as input and returns an instance of `<Target>User.cls`.

Here's an example `<Target>UserAttributeGenerator.cls` class from our SCIM v1.1 Connector:

```
global class SCIMUserAttributeGenerator {

    public static final String PARAM_FIRSTNAME = 'firstName';
    public static final String PARAM_LASTNAME = 'lastName';
    public static final String PARAM_EMAIL = 'email';
    public static final String PARAM_USERNAME = 'username';
    public static final String PARAM_EXTERNALID = 'externalId';
    public static final String PARAM_ENTITLEMENT = 'entitlement';

    global SCIMUser getMappedAttributes(Process.PluginRequest request){
        String userName = (String)request.inputParameters.get(PARAM_USERNAME);
        String firstName =
(String)request.inputParameters.get(PARAM_FIRSTNAME);
        String lastName = (String)request.inputParameters.get(PARAM_LASTNAME);
        String externalid =
(String)request.inputParameters.get(PARAM_EXTERNALID);
```

```
        String email = (String)request.inputParameters.get(PARAM_EMAIL);
        String entitlement =
(String)request.inputParameters.get(PARAM_ENTITLEMENT);

        SCIMUser user = new SCIMUser();

        user.schemas = new List<String> {'urn:scim:schemas:core:1.0'};

        if (userName != null) user.userName = userName;

        if ((lastName != null) || (firstName != null))  {
            SCIMUser.Name name = new SCIMUser.Name();
            if (lastName != null) name.familyName = lastName;
            if (firstName != null) name.givenName = firstName;
            user.name = name;
        }

        if (externalid != null) user.externalId = externalid;

        if (email != null) {
            SCIMUser.Emails emails = new SCIMUser.Emails();
            emails.value = email;
            List<SCIMUser.Emails> emailsList = new List<SCIMUser.Emails>();
            emailsList.add(emails);
            user.emails = emailsList;
        }

        if (entitlement != null) {
            SCIMUser.Entitlements entitlements = new SCIMUser.Entitlements();
            entitlements.value = entitlement;
            entitlements.primary = true;
            List<SCIMUser.Entitlements> entitlementsList = new
List<SCIMUser.Entitlements>();
            entitlementsList.add(entitlements);
            user.entitlements = entitlementsList;
        }

        return user;
    }

    global String getSerializedAttributes(SCIMUser user){
        return System.JSON.serialize(user, true);
    }

}
```

Each connector supports a default set of attributes. If you're working within the set of attributes supported by your connector and simply want to change how the attribute values are set, you can continue to use the <Target>UserAttributeGenerator class that comes with the connector. You might find this surprising because this is where the attribute mapping occurs. However, we've designed this connector class in such a way that you can make minor adjustments without a significant amount of coding. If this class was packaged, you would not need to reimplement it to change your attributes. For unmanaged code, you are welcome to make your changes here.

<Target>UserAttributeGeneratorPlugin.cls

Every connector comes with a `<Target>UserAttributeGeneratorPlugin.cls` class.This class is used to connect the flow to the corresponding classes, `<Target>User.cls` class and `<Target>UserAttributeGenerator.cls` class described above. If you want to make any changes to the logic used to generator, map, or transform attribute values, you have to modify this class. Even though we've designed this class to be easy to implement, it provides a lot of flexibility for customization.

Here's an example `<Target>UserAttributeGeneratorPlugin.cls` class from our SCIM v1.1 Connector:

```
global class SCIMUserAttributeGeneratorPlugin extends
UserProvisioning.UserProvisioningPlugin {

    //Describes the plugins inputs parameters. Used by the flow to determine
what
   //values to pass into the plugin.
   //Update this to include all input and output parameters
    global override Process.PluginDescribeResult buildDescribeCall() {
        Process.PluginDescribeResult describeResult = new
Process.PluginDescribeResult();
        // A Group for the Plugins
        describeResult.tag = 'SCIM';
        // The specific operation
        describeResult.Name = 'SCIM: Generate User Attributes';
        describeResult.inputParameters = new
List<Process.PluginDescribeResult.InputParameter>{
                    new
Process.PluginDescribeResult.InputParameter(SCIMUserAttributeGenerator.PARAM_US
ERNAME, Process.PluginDescribeResult.ParameterType.STRING, false),
                    new
Process.PluginDescribeResult.InputParameter(SCIMUserAttributeGenerator.PARAM_LA
STNAME, Process.PluginDescribeResult.ParameterType.STRING, false),
```

```
                    new
Process.PluginDescribeResult.InputParameter(SCIMUserAttributeGenerator.PARAM_FI
RSTNAME, Process.PluginDescribeResult.ParameterType.STRING, false),
                    new
Process.PluginDescribeResult.InputParameter(SCIMUserAttributeGenerator.PARAM_EX
TERNALID, Process.PluginDescribeResult.ParameterType.STRING, false),
                    new
Process.PluginDescribeResult.InputParameter(SCIMUserAttributeGenerator.PARAM_EM
AIL, Process.PluginDescribeResult.ParameterType.STRING, false),
                    new
Process.PluginDescribeResult.InputParameter(SCIMUserAttributeGenerator.PARAM_EN
TITLEMENT, Process.PluginDescribeResult.ParameterType.STRING, false)
        };

        describeResult.outputParameters = new
List<Process.PluginDescribeResult.OutputParameter>{
           new
Process.PluginDescribeResult.OutputParameter(UserProvisioningUtils.PARAM_USERPA
YLOAD, Process.PluginDescribeResult.ParameterType.STRING)
        };

        return describeResult;
    }

    //Method used to generate the attributes and map them to the third party.
    global override Process.PluginResult invoke(Process.PluginRequest request)
{
        // Attribute generation & serialization
        SCIMUserAttributeGenerator generator = new
SCIMUserAttributeGenerator();
        SCIMUser user = generator.getMappedAttributes(request);
        String payload = generator.getSerializedAttributes(user);

        Map<String, Object> result = new Map<String, Object>();
        result.put(UserProvisioningUtils.PARAM_USERPAYLOAD, payload);

        return new Process.PluginResult(result);
    }
}
```

This class by far provides the most power and flexibility, so it's important to understand it in complete detail. The first section under the `buildDescribeCall` method describes the inputs that are passed into this plugin. This typically consists of each user attribute you'd like map from the target user's User record. For example, the users FirstName, LastName, Email Address, and so on. The flow has this contextual information and can pass it into your plugin at runtime. You can also use these attributes to lookup additional information about the user. For example,

you can pass in the user's userID, and use SOQL to pull in corresponding attributes, entitlements, and so on.

After the data is passed into the plugin, there are two key lines of code, which we go into detail below:

```
SCIMUser user = generator.getMappedAttributes(request);
String payload = generator.getSerializedAttributes(user);
```

SCIMUser user = generator.getMappedAttributes(request);

This call will passes the instance of `Process.PluginRequest` (all your input parameters and values) into the `<Target>UserAttributeGenerator.cls` and returns an instance of `<Target>User.cls`. If you're using the set of attributes supported by the connector and you want to transform some of the attribute values, this is the perfect place to do so. Just modify the values in the `<Target>User.cls` instance directly.

For example, the SCIM connector used in the example above sets a single entitlement. If you want to add a secondary entitlement to each user, you can do so in the plugin:

```
...
 SCIMUser user = generator.getMappedAttributes(request);

      //<--Start Custom Code Here-->
      //Create new instances for your entitlements
      SCIMUser.Entitlements entitlement1 = new SCIMUser.Entitlements();
      SCIMUser.Entitlements entitlement2 = new SCIMUser.Entitlements();

      //Set primary entitlement
            entitlement1.value =
(String)request.inputParameters.get('entitlement');
            entitlement1.primary = true;

      //Set new entitlement
      entitlement2.value = '0PS61000000t5PlGAI'
      entitlement2.primary = false;

      //Add new entitlements to an entitlement list
            List<SCIMUser.Entitlements> entitlementsList = new
List<SCIMUser.Entitlements>();
            entitlementsList.add(entitlement1);
      entitlementsList.add(entitlement2);

      //Overwrite the entitlement generated by the
<Target>UserAttributeGenerator.cls class
```

```
            user.entitlements = entitlementsList;

      //<--End Custom Code Here-->
String payload = generator.getSerializedAttributes(user);
...
```

By changing only one class, you've modified how the Entitlement attribute is set. Repeat this same logic for any other attribute you want to override.

String payload = generator.getSerializedAttributes(user);

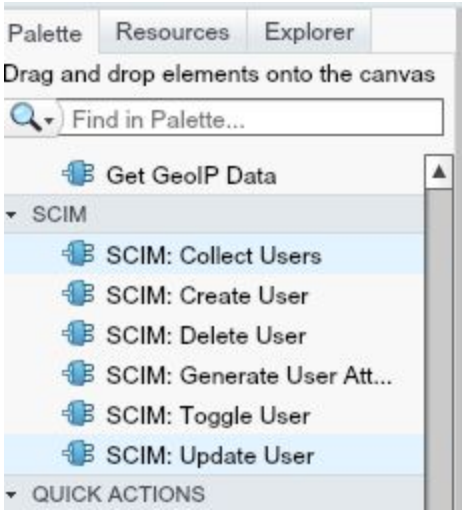
After the instance of `<Target>User.cls` has the correct attribute values set, it's serialized and returned to the flow as a JSON string. This gives you an additional opportunity to make modifications while modifying only the `<Target>UserAttributeGeneratorPlugin.cls` class. If you have additional attributes that you want to set beyond what's provided out of the box by the connector, you can choose to modify the JSON directly instead of reimplementing the `<Target>User.cls` and `<Target>UserAttributeGenerator.cls` classes. For example, if you need to add a new attribute Quota__c to your call, and don't want to reimplement the classes above, here's how you do it:

```
...
SCIMUser user = generator.getMappedAttributes(request);
String payload = generator.getSerializedAttributes(user);

      //<--Start Custom Code Here-->
      //Deserialize the payload JSON into a map of type String/Object
      Map<String, Object> userMap = (Map<String,
Object>)JSON.deserializeUntyped(payload);

      String quota = 'quote value';
      userMap.add('Quota__c',quota);
      payload = System.JSON.serialize(userMap, true);
      //<--End Custom Code Here-->
```

This approach does add some overhead because you'll be serializing and deserializing multiple times. It's also not as clean of a solution as updating `<Target>User` and `<Target>UserAttributeGenerator` classes. However, if you code is managed, it provides a quick way to make updates without having to re-write all three classes. Evaluate your specific situation and use case when determining which method to use.

Reference the Plugin in Your Flow

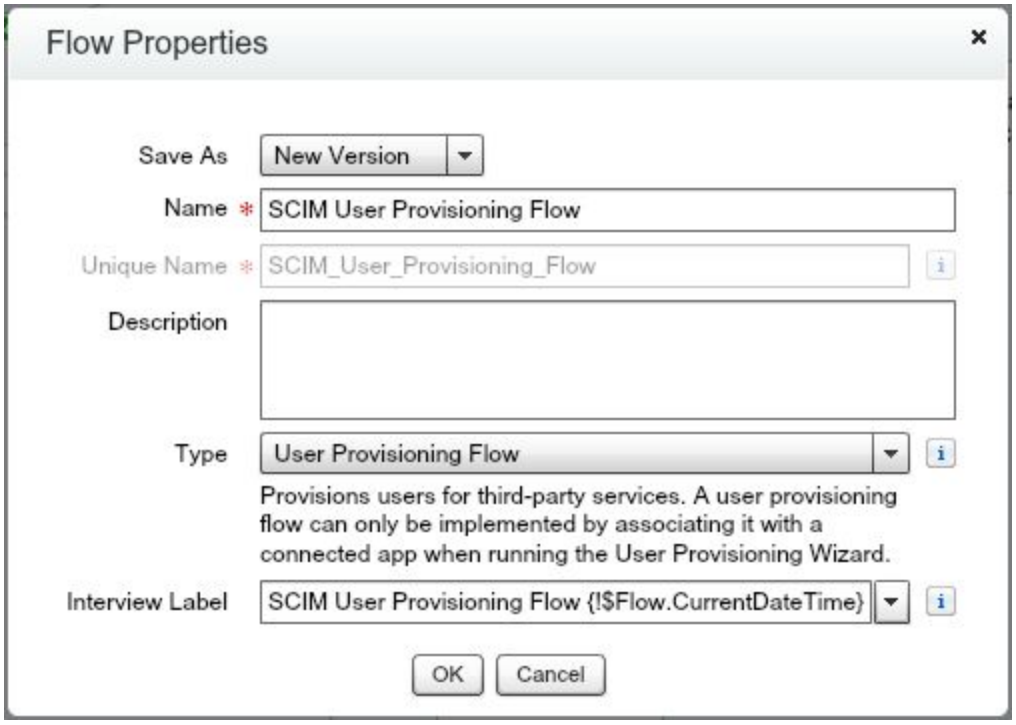
After your plugin class is completed, you need to update your flow so that it calls the right code. Since your plugin extends the `UserProvisioning.UserProvisioningPlugin` class, the plugin is automatically available on the Flow Palette.



Simply open your provisioning flow and drag your new plugin from the Palette onto the Flow Canvas. Then map your plugin's input attributes, and link the plugin appropriately within your flow.

If you're unfamiliar with Visual Workflows, see the Business Process Automation with Visual Workflow Trailhead module for more information.

Also after you complete your changes, ensure that you **save your flow as type "User Provisioning Flow."** The provisioning engine can't access your flow if it's saved as any other type.



Conclusion

Salesforce provisioning connectors are fully extensible and provide a significant amount of flexibility to ensure that you can achieve your desired use cases. You can make small modifications to a single class (`<Target>UserAttributeGeneratorPlugin.cls`), or you can fully customize all aspects of the attribute generation process. Because the plugins that interact with the third party are unchanged, your code changes and testing can be kept to a minimal. Plus, you're still utilizing the benefits of the out-of-box connectors.