First-Generation Managed Packaging Developer Guide

Version 60.0, Spring '24

@salesforcedocs

Last updated: January 12, 2024

CONTENTS

CHAPTER 1 First-Generation Managed Packages

In this chapter ...

Managed packages are used by Salesforce partners to distribute and sell applications to customers. Using AppExchange and the License Management Application (LMA), developers can sell and manage user-based licenses to their app. Managed packages are upgradeable.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

- Developing and Distributing Unmanaged Packages

Why Switch to Second-Generation Managed Packaging?

You've been using first-generation managed packages to develop your apps, so you're probably pretty familiar with what works well, and what's a bit more painful than you'd like. And no doubt, you're aware that second-generation managed packages is our newer technology, but maybe you aren't so sure why switching to second-generation managed packaging (managed 2GP) will improve your package development experience. So let's talk about that.

Source-Driven Development

The source-driven development model used in managed 2GP is a big shift from the org-based development used in managed 1GP. Say goodbye to packaging orgs as your source of truth. Instead, your source of truth with managed 2GP is the package metadata in your version control system. And as you develop your managed 2GP package, you create and update your package metadata in a version control system, not in an org.

Minimal Interaction with Salesforce Orgs

As you probably know well, with managed 1GP development, every package and patch version requires a unique Salesforce org, so it's not uncommon for you to own 100s of Salesforce orgs in which your package metadata is deployed. Managing these orgs and their credentials can become a nightmare.

Managed 2GP takes away the hassle of managing orgs, and instead you use a single org, the Dev Hub org, to manage all your packages. And even when you do need to connect to your Dev Hub org you'll use Salesforce CLI (Command Line Interface) or a script to log in.

By eliminating the need to manually log in and keep track of hundreds of packaging and patch orgs (and their login credentials), managed 2GP simplifies package development and promotes modern, programmatic Application Lifecycle Management (ALM).

API- and CLI-first Model

Unlike managed 1GP, which has only partial API coverage, you can perform every managed 2GP packaging operation using an API or CLI command. You can completely automate packaging operations and be more productive. Repeatable, scriptable, and track-able ALM is truly possible with managed 2GP.

Flexible Versioning

Managed 1GP packaging follows a linear versioning model that requires you to build upon the previous package version. This approach is very restrictive, and for metadata that can't be removed from a package, you're stuck with that metadata in your managed 1GP.

Enter managed 2GP and flexible versioning. If you create a managed-released package version that you haven't yet distributed to a customer, you can abandon that package version and select a previous package version as the ancestor you want to build upon. Flexible versioning also allows you to use branches and do parallel package development. You can iterate fast, learn from, and move on from any mistakes.

One Namespace Shared Across Multiple Packages

Managed 1GP packages require each package to have a unique namespace. This restriction can lead to a proliferation of global Apex because sharing code among packages is only possible by declaring Apex classes and methods as global.

Managed 2GP changes the game by allowing multiple packages to share the same namespace. The `@namespaceAccessible` annotation then lets you share public Apex classes and methods across all packages in the same namespace. By using public Apex, you don't increase your global Apex footprint by exposing a global API.

Declarative Dependencies

In managed 2GP packaging, you specify dependencies among packages declaratively in a `.json` file. Which as you know, is a more developer-friendly approach than how managed 1GP dependencies are declared.

Simplified Patch Versioning

Creating a patch version of a managed 2GP is as easy as creating a new major or minor package version. You use a Salesforce CLI command and specify a non-zero number for the patch version number. And that's it!

Because your version control system is the source of truth for managed 2GP, creating patch versions is straightforward. We promise you won't miss the laborious and error-prone patch org process of managed 1GP.

Avoid Having to Migrate Customers in the Future

As you may be aware, we're developing capabilities to migrate your managed 1GP packages to managed 2GP. However, when we launch that capability, there's work that you have to do to migrate your managed 1GP packages and customers from 1GP to 2GP. By adopting managed 2GP today for your new packages, you avoid the hassle of migration in the future.

Set Up Your Environments for First-Generation Managed Packages

The Environment Hub and Dev Hub are essential to your package development workflow.

Environment Hub

Use the Environment Hub to connect, create, view, and log in to Salesforce orgs from one location. If your company has multiple environments for development, testing, and trials, the Environment Hub lets you streamline your approach to org management.

Dev Hub

If you plan to use scratch orgs for your package development, you must first set up a Dev Hub. A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with different features and preferences.

Developer Hub

The Developer Hub (Dev Hub) lets you create and manage scratch orgs. The scratch org is a source-driven and disposable deployment of Salesforce code and metadata, made for developers and automation. A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with different features and preferences. Scratch orgs are a central feature of Salesforce DX, an open developer experience for developing and managing Salesforce apps across their entire lifecycle.

Environment Hub

The Environment Hub lets you connect, create, view, and log in to Salesforce orgs from one location. If your company has multiple environments for development, testing, and trials, the Environment Hub lets you streamline your approach to org management.

Developer Hub

The Developer Hub (Dev Hub) lets you create and manage scratch orgs. The scratch org is a source-driven and disposable deployment of Salesforce code and metadata, made for developers and automation. A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with different features and preferences. Scratch orgs are a central feature of Salesforce DX, an open developer experience for developing and managing Salesforce apps across their entire lifecycle.

To work with scratch orgs, you must first enable the Dev Hub in your Partner Business Org (PBO). You then use the Salesforce command-line interface (CLI) to create scratch orgs.

Note: Use the Dev Hub to manage scratch orgs. Continue using the Environment Hub to manage other types of orgs, including production and trial orgs.

EDITIONS

Available in: Lightning Experience

Available in: **Developer**,**Enterprise**, **Performance**, and **Unlimited** Editions

Scratch Org Allocations for Partners
To ensure optimal performance, partners are allocated a set number of scratch orgs in your business org. These allocations determine how many scratch orgs you can create daily, and how many can be active at a given point.

Enable Dev Hub Features in Your Org
Enable Dev Hub features in your PBO so you can create and manage scratch orgs, create and manage second-generation packages, and use Einstein features. Scratch orgs are disposable Salesforce orgs to support development and testing.

Add Salesforce DX Users
System administrators can access the Dev Hub org by default. You can enable more users to access the Dev Hub org so that they can create scratch orgs and use other developer-specific features.

Free Limited Access License
Request a Salesforce Limited Access - Free license to provide accounts to non-admin users in your production org, when these users require access to only a specific app, feature, or setting. Standard Salesforce objects such as Accounts, Contacts, and Opportunities aren’t accessible with this license.

Manage Scratch Orgs from the Dev Hub Org
You can view and delete your scratch orgs and their associated requests from the Dev Hub org.

Link a Namespace to a Dev Hub Org
To use a namespace with a scratch org, you must link the Developer Edition org where the namespace is registered to a Dev Hub org.

Supported Scratch Org Editions for Partners
Create partner edition scratch orgs from a Dev Hub partner business org.

SEE ALSO:

Salesforce CLI Setup Guide

Salesforce DX Developer Guide

Scratch Org Allocations for Partners

To ensure optimal performance, partners are allocated a set number of scratch orgs in your business org. These allocations determine how many scratch orgs you can create daily, and how many can be active at a given point.

By default, Salesforce deletes scratch orgs and their associated ActiveScratchOrg records from your Dev Hub when a scratch org expires. All partners get 100 Salesforce Limited Access - Free user licenses.

Summit Tier

- 300 active
- 600 daily

Crest Tier

- 150 active
- 300 daily

Ridge Tier

- 80 active
- 160 daily

Base Tier

- 40 active
- 80 daily

Partner Trials

- 20 active
- 40 daily

Enable Dev Hub Features in Your Org

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Dev Hub available in: **Developer**, **Enterprise**, **Performance**, and **Unlimited** Editions

Scratch orgs available in: **Developer**, **Enterprise**, **Group**, and **Professional** Editions

Enable Dev Hub features in your PBO so you can create and manage scratch orgs, create and manage second-generation packages, and use Einstein features. Scratch orgs are disposable Salesforce orgs to support development and testing.

Enabling Dev Hub in your PBO is safe and doesn't cause any performance or customer issues. Dev Hub comprises objects with permissions that allow admins to control the level of access available to a user and an org.

Note: You can't enable Dev Hub in a sandbox.

Consider these factors if you select a trial or Developer Edition org as your Dev Hub.

- You can create up to six scratch orgs and package versions per day, with a maximum of three active scratch orgs.
- Trial orgs expire on their expiration date.
- Developer Edition orgs can expire due to inactivity.
- You can define a namespace in a Developer Edition org that isn't your Dev Hub, and you can enable Dev Hub in a Developer Edition org that doesn't contain a namespace.
- If you plan to create package versions or run continuous integration jobs, it's better to use your PBO as your Dev Hub because of higher scratch org and package version limits. Package versions are associated with your Dev Hub org. When a trial or Developer Edition org expires, you lose access to the package versions.

Note: Partner trial orgs signed up from the partner community have different scratch org limits. See Scratch Org Allocations for Partners. Partners can create partner edition scratch orgs: Partner Developer, Partner Enterprise, Partner Group, and Partner Professional. This feature is available only if creating scratch orgs from a Dev Hub in a partner business org. See Supported Scratch Org Editions for Partners in the *First-Generation Managed Packaging Developer Guide* for details.

The Dev Hub org instance determines where scratch orgs are created.

- Scratch orgs created from a Dev Hub org in Government Cloud are created on a Government Cloud instance.
- Scratch orgs created from a Dev Hub org in Public Cloud are created on a Public Cloud instance.

To enable Dev Hub in an org:

1. Log in as System Administrator to your Developer Edition, trial, or production org (for customers), or your business org (for ISVs).
2. From Setup, enter `Dev Hub` in the Quick Find box and select **Dev Hub**.

   If you don't see Dev Hub in the Setup menu, make sure that your org is one of the supported editions.
3. To enable Dev Hub, click **Enable**.

   After you enable Dev Hub, you can't disable it.

Add Salesforce DX Users

System administrators can access the Dev Hub org by default. You can enable more users to access the Dev Hub org so that they can create scratch orgs and use other developer-specific features.

You can use Salesforce DX with these standard user licenses: Salesforce, Salesforce Platform, and Developer.

If your org has Developer licenses, you can add users with the Developer profile and assign them the provided Developer permission set. Alternatively, you can add users with the Standard User or System Administrator profiles. For a standard user, you must create a permission set with the required Salesforce DX permissions. We recommend that you avoid adding users as system administrators unless their work requires that level of authority and not just Dev Hub org access.

Free Limited Access License

Request a Salesforce Limited Access - Free license to provide accounts to non-admin users in your production org, when these users require access to only a specific app, feature, or setting. Standard Salesforce objects such as Accounts, Contacts, and Opportunities aren't accessible with this license.

Contact your Salesforce account executive to request this license. A Salesforce admin can upgrade a Salesforce Limited Access - Free license to a standard Salesforce license at any time.

Second-Generation Managed Packages and Unlocked Packages

To create scratch orgs and unlocked or second-generation managed packages, developers require access to the Dev Hub org, which is often your production org. A Salesforce admin can then grant appropriate permissions to the Dev Hub objects (ScratchOrgInfo, ActiveScratchOrg, and NamespaceRegistry).

To give developers appropriate access to the Dev Hub org, create a permission set that contains these permissions:

- Object Settings > Scratch Org Info > Read, Create, and Delete
- Object Settings > Active Scratch Org > Read and Delete
- Object Settings > Namespace Registry > Read (to use a linked namespace in a scratch org)

To provide users with the ability to create unlocked or second-generation managed packages and package versions, the permission set must also contain:

- System Permissions > Create and Update Second-Generation Packages

If you choose to test your package in a scratch org, the Create and Update Second-Generation Packages permission is also required when creating the scratch org if you specified an ancestor version in the `sfdx-project.json` file. Alternatively, use the `--noancestors` flag with the `sf org create` command when you create the scratch org.

For more information, see *Salesforce DX Developer Guide*: Add Salesforce DX Users.

DevOps Center

DevOps Center is installed as a managed package. Most team members, such as builders and developers, don't need to install and configure DevOps Center. You can provide these team members minimum access to the org where DevOps Center is installed by assigning them this license and the Limited Access User profile.

See Salesforce Help: Assign the DevOps Center Permission Sets

Features Not Currently Supported

- To use Org Shape for Scratch Orgs or Scratch Org Snapshots (pilot), be sure to assign the Salesforce user license. The Salesforce Limited Access - Free license isn't supported at this time.
- The Salesforce Limited Access - Free license doesn't provide access to some Salesforce CLI commands, such as `sf limits api display`. Contact your Salesforce admin for API limits information.

Manage Scratch Orgs from the Dev Hub Org

You can view and delete your scratch orgs and their associated requests from the Dev Hub org.

In the Dev Hub org, the ActiveScratchOrg standard object represents the scratch orgs that are currently in use. The ScratchOrgInfo standard object represents the requests that were used to create scratch orgs and provides historical context.

1. Log in to the Dev Hub org as the System Administrator or as a user with the Salesforce DX permissions.
2. From the App Launcher, select **Active Scratch Orgs** to see a list of all active scratch orgs.

   To view more details about a scratch org, click the link in the Number column.
3. To delete an active scratch org from the Active Scratch Orgs list view, choose **Delete** from the dropdown.

   Deleting an active scratch org doesn't delete the request (ScratchOrgInfo) that created it, but it does free up a scratch org so that it doesn't count against your allocations.
4. To view the requests that created the scratch orgs, select **Scratch Org Infos** from the App Launcher.

   To view more details about a request, click the link in the Number column. The details of a scratch org request include whether it's active, expired, or deleted.
5. To delete the request that was used to create a scratch org, choose **Delete** from the dropdown.

   Deleting the request (ScratchOrgInfo) also deletes the active scratch org.

Link a Namespace to a Dev Hub Org

To use a namespace with a scratch org, you must link the Developer Edition org where the namespace is registered to a Dev Hub org.

Complete these tasks before you link a namespace.

- If you don't have an org with a registered namespace, create a Developer Edition org that is separate from the Dev Hub or scratch orgs. If you already have an org with a registered namespace, you're good to go.
- In the Developer Edition org, create and register the namespace.

  Important: Choose namespaces carefully. If you're trying out this feature or need a namespace for testing purposes, choose a disposable namespace. Don't choose a namespace that you want to use in the future for a production org or some other real use case. After you associate a namespace with an org, you can't change it or reuse it.

1. Log in to your Dev Hub org as the System Administrator or as a user with the Salesforce DX Namespace Registry permissions.

   Tip: Make sure your browser allows pop-ups from your Dev Hub org.

   a. From the App Launcher menu, select **Namespace Registries**.

   b. Click **Link Namespace**.

2. In the window that pops up, log in to the Developer Edition org in which your namespace is registered using the org's System Administrator's credentials.

   You can't link orgs without a namespace: sandboxes, scratch orgs, patch orgs, and branch orgs require a namespace to be linked to the Namespace Registry.

To view all the namespaces linked to the Namespace Registry, select the **All Namespace Registries** list view.

Supported Scratch Org Editions for Partners

Create partner edition scratch orgs from a Dev Hub partner business org.

Supported partner scratch org editions include:

- Partner Developer
- Partner Enterprise
- Partner Group
- Partner Professional

Indicate the partner edition in the scratch org definition file.

```
"edition": "Partner Enterprise",
```

If you attempt to create a partner scratch org and see this error, confirm that you're using an active partner business org. Contact the Partner Community for further assistance.

```
ERROR:  You don't have permission to create Partner Edition organizations.
To enable this functionality, please log a case in the Partner Community.
```

License limits for partner scratch orgs are similar to partner edition orgs created in Environment Hub. Get the details on the Partner Community.

Environment Hub

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

The Environment Hub lets you connect, create, view, and log in to Salesforce orgs from one location. If your company has multiple environments for development, testing, and trials, the Environment Hub lets you streamline your approach to org management.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

From the Environment Hub, you can:

- Connect existing orgs to the hub with automatic discovery of related orgs.
- Create standard and partner edition orgs for development, testing, and trials.
- View and filter hub members according to criteria that you choose, like edition, creation date, instance, origin, and SSO status.
- Create single sign-on (SSO) user mappings for easy login access to hub members.

Each hub member org corresponds to an EnvironmentHubMember object. EnvironmentHubMember is a standard object, similar to Accounts or Contacts, so you can use the platform to extend or modify the Environment Hub programmatically. For example, you can create custom fields, set up workflow rules, or define user mappings and enable SSO using the API for any hub member org.

Get Started with the Environment Hub

Configure the Environment Hub so that users at your company can access the app to create and manage member orgs. Then connect existing orgs to the hub and create SSO user mappings.

Manage Orgs in the Environment Hub

You can manage all your existing Salesforce orgs from one location by connecting them to the Environment Hub. You can also create orgs using Salesforce templates for development, testing, and trial purposes.

Single Sign-on in the Environment Hub

Developing, testing, and deploying apps means switching between multiple Salesforce environments and providing login credentials each time. Single sign-on (SSO) simplifies this process by letting an Environment Hub user log in to member orgs without reauthenticating. You can set up SSO by defining user mappings manually, using Federation IDs, or creating a formula.

Environment Hub Best Practices

Follow these guidelines and best practices when you use the Environment Hub.

Environment Hub FAQ

Answers to common questions about the Environment Hub.

Considerations for the Environment Hub in Lightning Experience

Be aware of these considerations when creating and managing orgs in the Environment Hub.

Get Started with the Environment Hub

Configure the Environment Hub so that users at your company can access the app to create and manage member orgs. Then connect existing orgs to the hub and create SSO user mappings.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

Configure the Environment Hub

Enable the Environment Hub in your org, and then configure it to give other users access. If you're an ISV partner, the Environment Hub is already installed in your Partner Business Org.

Configure the Environment Hub

USER PERMISSIONS

To set up and configure the Environment Hub:
- Manage Environment Hub

Enable the Environment Hub in your org, and then configure it to give other users access. If you're an ISV partner, the Environment Hub is already installed in your Partner Business Org.

1. Contact Salesforce Customer Support to open a case to enable the Environment Hub in your org.

   If you're an ISV partner, you can skip this step.

2. Log in to the org where the Environment Hub is enabled.
3. Assign users access to features in the Environment Hub by creating or updating a permission set or profile.

   Be sure to assign users the Salesforce or Salesforce Platform license.

Permission Set	**Profile**	**Environment Hub Settings**
N/A	Custom App Settings	Enabled for Lightning Experience by default. Enable the Environment Hub custom app setting to make it available in the App Menu in Salesforce Classic.
System Permissions	Administrative Permissions	Enable "Manage Environment Hub" to allow users to: • Create orgs for development, testing, and trials. • Configure SSO for member orgs.
System Permissions	General User Permissions	Enable "Connect Organization to Environment Hub" to allow users to connect existing orgs to the Environment Hub.
Object Settings	Standard Object Permissions	Grant object permissions based on the required level of access for the Environment Hub user. Hub Members object: • "Tab Settings"—Visible • "Read"—View existing Hub Member records.

Permission Set	Profile	Environment Hub Settings
		• "Create"—This permission has no impact on the ability to create Hub Member records because record creation is handled either by connecting an existing org or creating an org from the Environment Hub. • "Edit"—Edit fields on existing Hub Member records. • "Delete"—Disconnect an org from the Environment Hub and delete its corresponding Hub Member record and Service Provider record (if SSO was enabled for the member). • "View All"—Read all Hub Member records, regardless of who created them. • "Modify All"—Read, edit, and delete all Hub Member records, regardless of who created them. Hub Invitations object: • If you enable the "Connect Organization to Environment Hub" permission, enable "Create", "Read", "Edit", and "Delete" for Hub Invitations. Signup Requests object: • If you enable the "Manage Environment Hub" permission, enable "Create" and "Read" for Signup Requests to allow users to create orgs. Optionally, enable "Delete" to allow users to remove orgs from the hub.
Service Providers	Service Provider Access	When configuring the Environment Hub in a new org, this section is empty. If you enable single sign-on (SSO) in a member org, new entries appear in this section. Entries appear in the format `Service Provider [Organization ID]`, where Organization ID is the ID of the member org. Users who don't have access to the service provider sometimes see this message when attempting to log in via SSO: `User '[UserID]' does not have access to sp '[Service Provider ID]'.`

Manage Orgs in the Environment Hub

You can manage all your existing Salesforce orgs from one location by connecting them to the Environment Hub. You can also create orgs using Salesforce templates for development, testing, and trial purposes.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

Connect an Org to the Environment Hub

You can connect existing Salesforce orgs to the Environment Hub, allowing you to manage all your development, test, and trial environments (except scratch orgs) from one location. When you connect an org to the hub, related orgs are automatically discovered so you don't have to manually connect them.

Create an Org from the Environment Hub

You can create orgs from the Environment Hub for development, testing, and trial purposes. If you're an ISV partner, you can also create partner edition orgs with increased limits, more storage, and other customizations to support app development. When you create an org from the Environment Hub, it becomes a hub member and its default language is set by the user's locale.

Connect an Org to the Environment Hub

You can connect existing Salesforce orgs to the Environment Hub, allowing you to manage all your development, test, and trial environments (except scratch orgs) from one location. When you connect an org to the hub, related orgs are automatically discovered so you don't have to manually connect them.

USER PERMISSIONS

To connect or disconnect an org to or from the Environment Hub:

- Connect Organization to Environment Hub

The following types of related orgs are automatically discovered.

- For any organization, all sandbox orgs created from it
- For a managed 1GP packaging org, all its related patch orgs
- For a Trialforce Management Org, all Trialforce Source Orgs created from it
- For an org with the License Management App (LMA) installed, any release org with a managed package registered in the LMA

Note: You can't connect a sandbox org to the Environment Hub directly. If you want to connect a sandbox, first connect the org used to create the sandbox to the Environment Hub. Then, refresh the sandbox org. The refresh automatically adds it as a hub member.

1. Log in to the Environment Hub, and then select **Connect Org**.
2. Enter the admin username for the org that you want to connect and, optionally, a short description. If your hub has many members, a description makes it easier to find the org later.
3. By default, single sign-on (SSO) is enabled for the org you connected. To disable SSO, deselect **Auto-enable SSO for this org**.
4. Select **Connect Org** again.
5. In the pop-up window, enter the org's admin username and password. If you don't see the pop-up, temporarily disable your browser's ad blocking software and try again.
6. Select **Log In**, and then select **Allow**.

   This process creates a connected app to allow connections to the org. If you can't log in and select Allow, check if the Environment Hub org has a connected app called "Environment org". If you don't see this connected app, contact Salesforce Support.

To disconnect an org, locate the listing for the org in the Environments Hub tab, and select **Remove** from the dropdown menu on the far right.

Orgs removed from the Environment Hub aren't deleted, so you can still access the org after you remove it.

Create an Org from the Environment Hub

USER PERMISSIONS

To set up and configure the Environment Hub:

- Manage Environment Hub

You can create orgs from the Environment Hub for development, testing, and trial purposes. If you're an ISV partner, you can also create partner edition orgs with increased limits, more storage, and other customizations to support app development. When you create an org from the Environment Hub, it becomes a hub member and its default language is set by the user's locale.

Note: You can create up to 20 member orgs per day. To create more orgs, log a support case in the Salesforce Partner Community

1. Log in to the Environment Hub, and then select **Create Org**.
2. Choose an org purpose.

Purpose	Lets You Create:
Development	Developer Edition orgs for building and packaging apps.
Test/Demo	Trial versions of standard Salesforce orgs for testing and demos. These orgs are similar to the ones customers create at www.salesforce.com/trial. When you create a Test/Demo org, you can specify a Trialforce template if you want the org to include your customizations.
Trialforce Source Organization	Trialforce Source Organizations (TSOs) as an alternative to using a Trialforce Management Organization (TMO). Unless you need custom branding on your login page or emails, use the Environment Hub to create TSOs.

3. Enter the required information for the org type you selected.
4. Read the Main Services Agreement, and then select the checkbox.
5. Select **Create**.

When your org is ready, you receive an email confirmation, and the org appears in your list of hub members.

Single Sign-on in the Environment Hub

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

Developing, testing, and deploying apps means switching between multiple Salesforce environments and providing login credentials each time. Single sign-on (SSO) simplifies this process by letting an Environment Hub user log in to member orgs without reauthenticating. You can set up SSO by defining user mappings manually, using Federation IDs, or creating a formula.

The Environment Hub supports these SSO methods for matching users.

SSO Method	Description
Mapped Users	Match users in the Environment Hub to users in a member org manually. Mapped Users is the default method for SSO user mappings defined from the member detail page.

SSO Method	Description
Federation ID	Match users who have the same Federation ID in both the Environment Hub and a member org.
User Name Formula	Match users in the Environment Hub and a member org according to a formula that you define.

If you specify multiple SSO methods, they're evaluated in this order: (1) Mapped Users, (2) Federation ID, and (3) User Name Formula. The first method that results in a match is used to log in the user, and the other methods are ignored. If a matching user can't be identified, the Environment Hub directs the user to the standard Salesforce login page.

Note: SSO doesn't work for newly added users or for user mappings defined in a sandbox org. Only add users, edit user information, or define SSO user mappings in the parent org for the sandbox.

Enable SSO for a Member Org

You can enable single sign-on (SSO) to let an Environment Hub user log in to a member org without reauthenticating.

USER PERMISSIONS

To set up and configure the Environment Hub:

- Manage Environment Hub

1. Log in to the Environment Hub, and then select a member org. If you don't see any member orgs, check your list view.
2. Select **Enable SSO**.
3. Confirm that you want to enable SSO for this org, and then select **Enable SSO** again.

Define an SSO User Mapping

You can manually define a single-sign on (SSO) user mapping between a user in the Environment Hub and a user in a member org. Before you define a user mapping, enable SSO in the hub member org.

USER PERMISSIONS

To set up and configure the Environment Hub:

- Manage Environment Hub

User mappings can be many-to-one but not one-to-many. In other words, you can associate multiple users in the Environment Hub to one user in a member org. For example, if you wanted members of your QA team to log in to a test org as the same user, you could define user mappings.

1. Log in to the Environment Hub, and then select a member org. If you don't see any member orgs, check your list view.

2. Go to the Single Sign-On User Mappings related list, and then select **New SSO User Mapping**.
3. Enter the username of the user that you want to map in the member org, and then look up a user in the Environment Hub.
4. Select **Save**.

Use a Federation ID or Formula for SSO

You can match an Environment Hub user with a user in a member org using a Federation ID or a user name formula. For either method, enable SSO in the hub member org first.

USER PERMISSIONS

To set up and configure the Environment Hub:

- Manage Environment Hub

1. Log in to the Environment Hub, and then select a member org. If you don't see any member orgs, check your list view.
2. Go to SSO Settings, and then choose a method.

Method	Steps
`SSO Method 2 - Federation ID` Match users who have the same Federation ID in both the Environment Hub and a member org.	Select the checkbox.
`SSO Method 3 - User Name Formula` Match users in the Environment Hub and a member org according to a formula that you define.	Select the checkbox, and then define a formula. For example, to match the first part of the username (the part before the "@" sign) with an explicit domain name, enter: `LEFT($User.Username, FIND("@", $User.Username)) & ("mydev.org")`

3. Select **Save**.

Disable SSO for a Member Org

If you want Environment Hub users to reauthenticate when they log in to a member org, you can disable SSO. Disabling SSO doesn't remove the user mappings that you've defined, so you can always re-enable SSO later.

USER PERMISSIONS

To set up and configure the Environment Hub:

- Manage Environment Hub

1. Log in to the Environment Hub, and then select a member org. If you don't see any member orgs, check your list view.
2. Select **Disable SSO**.
3. Confirm that you want to disable SSO for this org, and then select **Disable SSO** again.

Environment Hub Best Practices

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

Follow these guidelines and best practices when you use the Environment Hub.

- If you're an admin or developer, choose the org that your team uses most frequently as your hub org. If you're an ISV partner, the Environment Hub is already installed in your Partner Business Org.
- Because each member org is a standard object (of type EnvironmentHubMember), you can modify its behavior or access it programmatically. For example, you can create custom fields, set up workflow rules, or define user mappings and enable single sign-on using the API for any member org.
- Decide on a strategy for enabling SSO access based on your company's security requirements. Then choose the SSO method (explicit mapping, Federation ID, or custom formula) that meets your needs.
- SSO doesn't work for newly added users or for user mappings defined in a sandbox org. Only add users, edit user information, or define SSO user mappings in the parent org for the sandbox.
- The Environment Hub connected app is for internal use only. Don't enable it for any profiles. Unless advised by Salesforce, don't delete the connected app or adjust its settings.

Environment Hub FAQ

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Answers to common questions about the Environment Hub.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

Can I use the Environment Hub in Lightning Experience?

Where do I install the Environment Hub?

Can I install the Environment Hub in more than one org?

Can I enable the Environment Hub in a sandbox org?

What kinds of orgs can I create in the Environment Hub?

You can create orgs for development, testing, and trials. ISV partners can also create partner edition orgs with increased limits, more storage, and other customizations to support app development. If you're a partner but don't see partner edition orgs in the Environment Hub, log a support case in the Salesforce Partner Community.

How is locale determined for the orgs I create in the Environment Hub?

Your Salesforce user locale determines the default locale of orgs that you create.

Are the orgs that I create in the Environment Hub the same as the ones I created in the Partner Portal?

Yes, the orgs are identical to the ones that you created in the Partner Portal.

Can an org be a member of multiple Environment Hubs?

Can I disable the Environment Hub?

Can I use the Environment Hub in Lightning Experience?

Yes, both Salesforce Classic and Lightning Experience support the Environment Hub.

Where do I install the Environment Hub?

If you're an ISV partner, the Environment Hub is already installed in your Partner Business Org.

Otherwise, install the Environment Hub in an org that all your users can access, such as your CRM org. Do not install the Environment Hub in a Developer Edition org that contains your managed package. Doing so can cause problems when you upload a new package version or push an upgrade to customers.

Can I install the Environment Hub in more than one org?

Yes, but you must manage each Environment Hub independently. Although Salesforce recommends one Environment Hub per company, several hubs could make sense for your company. For example, if you want to keep orgs that are associated with product lines separate.

Can I enable the Environment Hub in a sandbox org?

No, you can't enable the Environment Hub in a sandbox org. Enable the Environment Hub in a production org that all your users can access.

What kinds of orgs can I create in the Environment Hub?

You can create orgs for development, testing, and trials. ISV partners can also create partner edition orgs with increased limits, more storage, and other customizations to support app development. If you're a partner but don't see partner edition orgs in the Environment Hub, log a support case in the Salesforce Partner Community.

Org Type	Best Used For	Expires After
Group Edition	Testing	30 days
Enterprise Edition	Testing	30 days
Professional Edition	Testing	30 days
Partner Developer Edition	Developing apps and Lightning components	Never
Partner Group Edition	Robust testing and customer demos	1 year, unless you request an extension
Partner Enterprise Edition	Robust testing and customer demos	1 year, unless you request an extension
Partner Professional Edition	Robust testing and customer demos	1 year, unless you request an extension
Trialforce Source Org	Creating Trialforce templates	1 year, unless you request an extension
Consulting Partner Edition	Customer demos	1 year, unless you request an extension

How is locale determined for the orgs I create in the Environment Hub?

Your Salesforce user locale determines the default locale of orgs that you create.

For example, if your user locale is set to `English (United Kingdom)`, that is the default locale for the orgs you create. In this way, the orgs you create are already customized for the regions where they reside.

Are the orgs that I create in the Environment Hub the same as the ones I created in the Partner Portal?

Yes, the orgs are identical to the ones that you created in the Partner Portal.

The Environment Hub uses the same templates, so the orgs come with the same customizations, such as higher limits and more licenses. You can also use the Environment Hub to create the same Group, Professional, and Enterprise Edition orgs that customers use. That way, you can test your app against realistic customer implementations.

Can an org be a member of multiple Environment Hubs?

No, an org can be a member of only one Environment Hub at a time. To remove an org from an Environment Hub so you can associate it with a different one:

1. Go to the Environment Hub tab.
2. Find the org, from the drop-down select **Remove**.
3. Once removed, connect the org to the desired Environment Hub:
    a. In the Environment Hub tab, click **Connect Org**.
    b. Enter the admin username for the org.
    c. Click **Connect Org**.
    d. Enter the org's password, then click **Allow** to allow the Environment Hub to access org information.

Can I disable the Environment Hub?

After you install the Environment Hub in an org, you can't disable it. However, you can hide the Environment Hub from users. Go to Setup and enter `App Menu` in to the Quick Find box, and then select **App Menu**. From the App Menu, you can choose whether to hide an app or make it visible.

Considerations for the Environment Hub in Lightning Experience

Be aware of these considerations when creating and managing orgs in the Environment Hub.

List View Limitations

You can't filter hub members by org expiration date when creating or updating list views in Lightning Experience. If you have an existing list view that includes org expiration date in its filter criteria, that list view won't work in Lightning Experience. To filter hub members by org expiration date, switch to Salesforce Classic and then use the list view.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Register a Namespace for a First-Generation Managed Package

A namespace is a one to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers on AppExchange. Namespace prefixes are case-insensitive. For example, ABC and abc aren't recognized as unique. Your namespace must be globally unique across all Salesforce orgs.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages.

We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

Warning: When creating a namespace, use something that's useful and informative to users. However, don't name a namespace after a person (for example, by using a person's name, nickname, or private information.)

Salesforce automatically prepends your namespace, followed by two underscores ("__"), to all unique component names in your Salesforce org. A unique package component is one that requires a name that no other component has within Salesforce, such as custom objects, custom fields, custom links, and validation rules. For example, if your namespace is abc and your managed package contains a custom object with the API name, Expense__c, use the API name abc__Expense__c to access this object using the API. The namespace is displayed on all component detail pages.

Your namespace must:

- Begin with a letter
- Contain one to 15 alphanumeric characters
- Not contain two consecutive underscores

For example, `myNp123` and `my_np` are valid namespaces, but `123Company` and `my__np` aren't.

To register a namespace:

1. From Setup, enter *`Package Manager`* in the Quick Find box and select **Package Manager**.
2. In the Namespace Settings panel, click **Edit**.

   Note: After you've configured your namespace settings, this button is hidden.

3. Enter the namespace you want to register.
4. To determine if the namespace is already in use, click **Check Availability**.
5. If the namespace prefix that you entered isn't available, repeat the previous two steps.
6. Click **Review**.
7. Click **Save**.

Create a First-Generation Managed Package Using a UI

USER PERMISSIONS

To create packages:
- Create AppExchange Packages

If your goal is to build an app and distribute it on AppExchange, you use managed packages to do both. Packaging is the container that you fill with metadata, and it holds the set of related features, customizations, and schema that make up your app. A package can include many different metadata components, and you can package a single component, an app, or library.

1. From Setup, in the Quick Find box, enter *`Package Manager`*, and then select **Package Manager**.
2. Click **New**.
3. Enter a name for your package. You can use a different name than what appears on AppExchange.
4. From the dropdown menu, select the default language of all component labels in the package.
5. (Optional) Choose a custom link from the `Configure Custom Link` field to display configuration information to installers of your app. You can select a predefined custom link to a URL that you've created for your home page layouts; see the Configure Option on page 51. The custom link appears as a **Configure** link within Salesforce on AppExchange Downloads page and app detail page of the installer's organization.

6. (Optional) In the `Notify on Apex Error` field, enter the username of the person to notify if an uncaught exception occurs in the Apex code. If you don't specify a username, all uncaught exceptions generate an email notification that is sent to Salesforce. This option is only available for managed packages. For more information, see Handle Apex Exceptions in Managed Packages.
7. (Optional) In the `Notify on Packaging Error` field, enter the email address of the person who receives an email notification if an error occurs when a subscriber's attempt to install, upgrade, or uninstall a packaged app fails. This field appears only if packaging error notifications are enabled. To enable notifications, contact your Salesforce representative.
8. (Optional) Enable language extension packages. (Beta)
   a. Under `Language Settings`, click **Edit**.
   b. Select `Enable Language Extension Package` and save your changes.
9. (Optional) Enter a description that describes the package. You can change this description before you upload it to AppExchange.
10. (Optional) Specify a post install script. You can run an Apex script in the subscriber organization after the package is installed or upgraded. For more information, see Running Apex on Package Install/Upgrade.
11. (Optional) Specify an uninstall script. You can run an Apex script in the subscriber organization after the package is uninstalled. For more information, see Running Apex on Package Uninstall.
12. Save your work.

What Are Beta Versions of Managed Packages?
A beta package is an early version of a managed package. The purpose of a beta package is to allow the developer to test their application in different Salesforce orgs and to share the app with a pilot set of users for evaluation and feedback.

Create a Beta Package for First-Generation Managed Packages
Follow this procedure to create and upload a beta package through the UI. (You can also upload a package using the Tooling API. For sample code and more details, see the PackageUploadRequest object in the *Tooling API Developer Guide*.)

Create and Upload a First-Generation Managed Package
Use the following procedure to create and upload a managed package through the UI. You can also upload a package using the Tooling API. For sample code and more details, see the PackageUploadRequest object in the *Tooling API Developer Guide*.

Publish Extensions to Managed Packages
An *extension* is any package, component, or set of components that adds to the functionality of a managed package. An extension requires that the base managed package is installed in the org. For example, if you have built a recruiting app, an extension to this app can include a component for performing background checks on candidates.

View Package Details in First-Generation Managed Packages
From Setup, enter *`Packages`* in the `Quick Find` box, then select **Packages**. Click the name of a package to view its details, including added components, whether it's a managed package, whether the package has been uploaded, and so on.

Notifications for Package Errors
Accurately track failed package installations, upgrades, and uninstallations in subscriber orgs with the Notifications for Package Errors feature. Proactively address issues with managed and unmanaged packages and provide support to subscribers so that they can successfully install and upgrade your apps.

What Are Beta Versions of Managed Packages?

A beta package is an early version of a managed package. The purpose of a beta package is to allow the developer to test their application in different Salesforce orgs and to share the app with a pilot set of users for evaluation and feedback.

Before installing a beta version of a managed package, review the following notes:

- Beta packages can be installed in scratch, sandbox, or Developer Edition orgs, or test orgs furnished through the Environment Hub only.
- The components of a beta package are editable in the packaging org until a Managed - Released package is uploaded.
- Beta versions aren't considered major releases, so the package version number doesn't change.
- Beta packages aren't upgradeable. Because developers can still edit the components of a beta package, the Managed - Released version might not be compatible with the beta package installed. To install a new beta package or released version, first, uninstall the beta package. For more information, see Uninstall a Managed Package on page 70 and Install a Managed Package on page 60.

Create a Beta Package for First-Generation Managed Packages

Follow this procedure to create and upload a beta package through the UI. (You can also upload a package using the Tooling API. For sample code and more details, see the PackageUploadRequest object in the *Tooling API Developer Guide*.)

USER PERMISSIONS

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

1. Create a package:
   a. From Setup, enter `Package Manager` in the `Quick Find` box, then select **Package Manager**.
   b. Click **New**.
   c. Enter a name for your package. You can use a different name than what appears on AppExchange.
   d. From the dropdown menu, select the default language of all component labels in the package.
   e. Optionally, choose a custom link from the `Configure Custom Link` field to display configuration information to installers of your app. You can select a predefined custom link to a URL or s-control that you've created for your home page layouts; see the Configure Option. The custom link displays as a **Configure** link within Salesforce on AppExchange Downloads page and app detail page of the installer's organization.
   f. Optionally, in the `Notify on Apex Error` field, enter the username of the person to notify if an uncaught exception occurs in the Apex code. If you don't specify a username, all uncaught exceptions generate an email notification that is sent to Salesforce. This option is only available for managed packages. Handling Apex Exceptions in Managed Packages.

      Note: Apex can only be packaged from Developer, Enterprise, Unlimited, and Performance Edition organizations.

   g. Optionally, in the `Notify on Packaging Error` field, enter the email address of the person who receives an email notification if an error occurs when a subscriber's attempt to install, upgrade, or uninstall a packaged app fails. This field appears only if packaging error notifications are enabled. To enable notifications, contact your Salesforce representative.
   h. Optionally, enter a description that describes the package. You can change this description before you upload it to AppExchange.
   i. Optionally, specify a post install script. You can run an Apex script in the subscriber organization after the package is installed or upgraded. For more information, see Running Apex on Package Install or Upgrade.
   j. Optionally, specify an uninstall script. You can run an Apex script in the subscriber organization after the package is uninstalled. For more information, see Running Apex on Package Uninstall.
   k. Click **Save**.
2. Optionally, change the API access privileges. By default, API access is set to `Unrestricted`, but you can change this setting to further restrict API access of the components in the package.

3. Add the necessary components for your app.
    a. Click **Add Components**.
    b. From the dropdown list, choose the type of component.
    c. Select the components you want to add.
    d. Click **Add To Package**.
    e. Repeat these steps until you added all the components you want in your package.

    Note: Some related components are automatically included in the package even though they don't display in the Package Components list. For example, when you add a custom object to a package, its custom fields, page layouts, and relationships with standard objects are automatically included. For a complete list of components, see Components Automatically Added to First-Generation Managed Packages on page 40.

4. Optionally, click **View Dependencies** and review a list of components that rely on other components, permissions, or preferences within the package. To return to the Package detail page, click **Done**.
5. Click **Upload**.
6. On the Upload Package page, do the following:
    a. Enter a `Version Name`, such as *`Spring '22`*. The version name is the marketing name for a specific release of a package and allows you to create a more descriptive title for the version than just a number.
    b. Enter a `Version Number`, such as *`1.0`*. For more information on versions, see Update Your First-Generation Managed Package on page 71.
    c. Select a `Release Type` of Managed - Beta.
    d. (Optional) Enter and confirm a password to share the package privately with anyone who has the password. Don't enter a password if you want to make the package available to anyone on AppExchange and share your package publicly.
    e. Salesforce automatically selects the requirements it finds. In addition, select any other required components from the `Package Requirements` and `Object Requirements` sections to notify installers of any requirements for this package.
    f. Click **Upload**.

After your package has uploaded successfully, you receive an email with an installation link.

Create and Upload a First-Generation Managed Package

Use the following procedure to create and upload a managed package through the UI. You can also upload a package using the Tooling API. For sample code and more details, see the PackageUploadRequest object in the *Tooling API Developer Guide*.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

USER PERMISSIONS

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages

These steps assume you've already created a namespace and beta package. If you're uploading a beta package for testing, see Create and Upload a Beta Package.

1. Create a package:
    a. From Setup, enter *`Package Manager`* in the `Quick Find` box, then select **Package Manager**.

b. Click **New**.

c. Enter a name for your package. You can use a different name than what appears on AppExchange.

d. From the dropdown menu, select the default language of all component labels in the package.

e. Optionally, choose a custom link from the `Configure Custom Link` field to display configuration information to installers of your app. You can select a predefined custom link to a URL or s-control that you've created for your home page layouts; see the Configure Option on page 51. The custom link displays as a **Configure** link within Salesforce on AppExchange Downloads page and app detail page of the installer's organization.

f. Optionally, in the `Notify on Apex Error` field, enter the username of the person to notify if an uncaught exception occurs in the Apex code. If you don't specify a username, all uncaught exceptions generate an email notification that is sent to Salesforce. This option is only available for managed packages.For more information, see Handle Apex Exceptions in Managed Packages.

Note: Apex can only be packaged from Developer, Enterprise, Unlimited, and Performance Edition organizations.

g. Optionally, in the `Notify on Packaging Error` field, enter the email address of the person who receives an email notification if an error occurs when a subscriber's attempt to install, upgrade, or uninstall a packaged app fails. This field appears only if packaging error notifications are enabled. To enable notifications, contact your Salesforce representative.

h. Optionally, enter a description that describes the package. You can change this description before you upload it to AppExchange.

i. Optionally, specify a post install script. You can run an Apex script in the subscriber organization after the package is installed or upgraded. For more information, see Running Apex on Package Install/Upgrade.

j. Optionally, specify an uninstall script. You can run an Apex script in the subscriber organization after the package is uninstalled. For more information, see Running Apex on Package Uninstall.

k. Click **Save**.

2. Salesforce sets your package API access privileges to `Unrestricted`. You can change this setting to further restrict API access of Salesforce components in the package. For more information, see Manage API and Dynamic Apex Access in Packages.

3. Add the necessary components for your app.

a. Click **Add Components**.

b. From the dropdown list, choose the type of component you want to add to your package.

- At the top of the list, click a letter to display the contents of the sorted column that begin with that character.
- If available, click the **Next Page** (or **Previous Page**) link to go to the next or previous set of components.
- If available, click **fewer** or **more** at the bottom of the list to view a shorter or longer display list.

c. Select the components you want to add.

d. Click **Add To Package**.

e. Repeat these steps until you added all the components you want in your package.

Note:

- Some related components are automatically included in the package even if they don't display in the Package Components list. For example, when you add a custom object to a package, its custom fields, page layouts, and relationships with standard objects are automatically included.
- When you package a joined report, each block is included in the package. Although the blocks appear in the package as reports, when you click a block, an error message indicates that you have "insufficient privileges" to view the report. This error message is expected behavior. Instead, click the name of the joined report to run it.

4. Optionally, click **View Dependencies** and review a list of components that rely on other components, permissions, or preferences within the package. An entity can include such things as an s-control, a standard or custom field, or an organization-wide setting like multicurrency. Your package can't be installed unless the installer has the listed components enabled or installed. For more information on dependencies, see Understanding Dependencies on page 44. To return to the Package detail page, click **Done**.

   Note: You can't upload packages that contain any of the following:

   - Workflow rules or workflow actions (such as field updates or outbound messages) that reference record types.
   - Reports that reference record types on standard objects.

5. Click **Upload**.

   Note: If you create a managed package to publish on AppExchange, you must certify your application before you package it. For more information, see Security Review on AppExchange.

6. On the Upload Package page, do the following:

   a. Enter a `Version Name`. As a best practice, it's useful to have a short description and the date.

   b. Enter a `Version Number` for the upload, such as *`1.0`*. The format is *`majorNumber.minorNumber`*.

   Note: If you're uploading a new patch version, you can't change the patch number.

   The version number represents a release of a package. This field is required for managed and unmanaged packages. For a managed package, the version number corresponds to a Managed - Released upload. All beta uploads use the same version number until you upload a Managed - Released package version with a new version number.

   For example, the following is a sequence of version numbers for a series of uploads.

Upload Sequence	Type	Version Number	Notes
First upload	Managed - Beta	1.0	The first Managed - Beta upload.
Second upload	Managed - Released	1.0	A Managed - Released upload. The version number doesn't change.
Third upload	Managed - Released	1.1	Note the change of minor release number for the Managed - Released upload.
Fourth upload	Managed - Beta	2.0	The first Managed - Beta upload for version number 2.0. Note the major version number update.
Fifth upload	Managed - Released	2.0	A Managed - Released upload. The version number doesn't change.

   c. For managed packages, select a `Release Type`:

   - Choose Managed - Released to upload an upgradeable version. After upload, some attributes of the metadata components are locked.
   - Choose Managed - Beta if you want to upload a version of your package to a small sampling of your audience for testing purposes. You can still change the components and upload other beta versions.

     Note: Beta packages can only be installed in Developer Edition,scratch, or sandbox orgs, and thus can't be pushed to customer orgs.

   d. Change the `Description`, if necessary.

e. (Optional) Specify a link to release notes for the package. Click **URL** and enter the details in the text field that appears. This link will be displayed during the installation process, and on the Package Details page after installation.

   Note: As a best practice, point to an external URL, so you can make the information available to customers before the release, and update it independently of the package.

f. (Optional) Specify a link to post install instructions for the package. Click **URL** or **Visualforce page** and enter the details in the text field that appears. This link will be displayed on the Package Details page after installation.

   Note: As a best practice, point to an external URL, so you can update the information independently of the package.

g. (Optional) Enter and confirm a password to share the package privately with anyone who has the password. Don't enter a password if you want to make the package available to anyone on AppExchange and share your package publicly.

h. Salesforce automatically selects the requirements it finds. In addition, select any other required components from the `Package Requirements` and `Object Requirements` sections to notify installers of any requirements for this package.

i. Click **Upload**.

7. After your upload is complete you can do any of the following.
   - To change the password option, click **Change Password** link.
   - To prevent new installations of this package while allowing existing installations to continue operating, click **Deprecate**.

     Note: You can't deprecate the most recent version of a managed package.

     When you deprecate a package, remember to remove it from AppExchange as well.
   - To make a deprecated version available for installation again, click **Undeprecate**.

You receive an email that includes an installation link when your package has been uploaded successfully.

Note:

- When using the install URL, the old installer is displayed by default. You can customize the installation behavior by modifying the installation URL you provide your customers.
  - To access the new installer, append the text `&newui=1` to the installation URL.
  - To access the new installer with the "All Users" option selected by default, append the additional text `&p1=full` to the installation URL.
- If you uploaded from your Salesforce production org, notify installers who want to install it in a sandbox org to replace the *login.salesforce.com* portion of the installation URL with *test.salesforce.com*.t"

Publish Extensions to Managed Packages

An *extension* is any package, component, or set of components that adds to the functionality of a managed package. An extension requires that the base managed package is installed in the org. For example, if you have built a recruiting app, an extension to this app can include a component for performing background checks on candidates.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

The community of developers, users, and visionaries building and publishing apps on AppExchange is part of what makes Salesforce Platform such a rich development platform. Use this community to build extensions to other apps and encourage them to build extensions to your apps.

When working with both first-generation (1GP) and second-generation (2GP) managed packages, only certain combinations of packages are supported.

Can I extend a first-generation managed package with a second-generation managed package?	Yes A second-generation managed package can depend on a first-generation managed package.
Can I extend a second-generation managed package with another second-generation managed package?	Yes
Can I extend a second-generation managed package with a first-generation managed package?	No A first-generation managed package can't depend on a second-generation managed package, and we block the installation of managed 2GP packages in managed 1GP packaging orgs. We can override this behavior on an individual basis. To share your scenario and request an override, log a case with Salesforce Partner Support. We're investigating how to support this dependency scenario more broadly.
Can I extend a first-generation managed package with another first-generation managed package?	Yes

To publish extensions to a managed package:

1. Install the base package in the Salesforce org that you plan to use to upload the extension.
2. Build your extension components.

EDITIONS

Available in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages

Note: To build an extension, install the base package and include a dependency to that base package in your package. The extension attribute automatically becomes active.

3. Create a package and add your extension components. Salesforce automatically includes some related components.
4. Upload the new package that contains the extension components.
5. Proceed with the publishing process as usual. For information on creating a test drive or registering and publishing your app, go to Salesforce Partner Community.

Note: Packages can't be upgraded to Managed - Beta if they're used within the same org as an extension.

View Package Details in First-Generation Managed Packages

From Setup, enter `Packages` in the `Quick Find` box, then select **Packages**. Click the name of a package to view its details, including added components, whether it's a managed package, whether the package has been uploaded, and so on.

From the package detail page:

- To change the package name, the custom link that displays when users click Configure, or the description, click **Edit**.
- To delete the package, click **Delete**. This action doesn't delete the components contained in the package, but the components are no longer bundled together within this package.
- To upload the package, click **Upload**. You're notified by email when the upload is complete.
- You can enable, disable, or change the dynamic Apex and API access that components in the package have to standard objects in the installing org by using the links next to `API Access`.

View Package Details

For package developers, the package detail section displays these attributes (in alphabetical order).

Attribute	Description
`API Access`	The type of access that the API and dynamic Apex that package components have. The default is **Unrestricted**, which means that all package components that access the API have the same access as the user who is logged in. Click **Enable Restrictions** or **Disable Restrictions** to change the API and dynamic Apex access permissions for a package.
`Created By`	The name of the developer that created this package, including the date and time.
`Description`	A description of the package.
`Language`	The language used for the labels on components. The default value is your user language.
`Last Modified By`	The name of the last user to modify this package, including the date and time.
`Notify on Apex Error`	The username of the person who receives email notifications when an exception occurs in Apex that isn't caught by the code. If you

Attribute	Description
	don't specify a username, all uncaught exceptions send an email notification to Salesforce. Available only for managed packages. Apex can be packaged only from Developer, Enterprise, Unlimited, and Performance Edition orgs.
`Notify on Packaging Error`	The email address of the person who receives email notifications if an error occurs when a subscriber's attempt to install, upgrade, or uninstall a packaged app fails. This field appears only if packaging error notifications are enabled. To enable notifications, contact your Salesforce representative.
`Package Name`	The name of the package, provided by the publisher.
`Push Upgrade Exclusion List`	A comma-separated list of org IDs to exclude when you push a package upgrade to subscribers.
`Post Install Script`	The Apex code that runs after this package is installed or upgraded. For more information, see Run Apex on Package Install/Upgrade on page 64.
`Type`	Indicates whether it's a managed or unmanaged package.
`Uninstall Script`	The Apex code that runs after this package is uninstalled. For more information, see Run Apex on Package Uninstall on page 68.

View Package Components

The Components tab lists each package component contained in the package, including the name and type of each component.

Note: Some related components are automatically included in the package even though they aren't displayed in the Package Components list. For example, when you add a custom object to a package, its custom fields, page layouts, and relationships with standard objects are included. For a list of components that Salesforce automatically includes, see Components Automatically Added on page 40.

Package components frequently depend on other components that aren't always added to the package explicitly. Each time you change a package, Salesforce checks for dependencies and displays the components as package members. Package Manager checks for dependencies and shows the component relationship to the package in the Include By column of the Package Details.

When your package contains 1,000 or more components, you can decide when to refresh the components list in the Package Details and avoid a long wait while this page loads. The components list refreshes automatically for packages with less than 1,000 components. Click **Refresh Components** if the package has new or changed components, and wait for the list to refresh.

Click **View Dependencies** to review a list of components that rely on other components, permissions, or preferences within the package. An entity can include such things as a standard or custom field, or an organization-wide setting like multicurrency. Your package can't be installed unless the installer has the listed components enabled or installed. Click **Back to Package** to return to the Package detail page.

Click **View Deleted Components** to see which components were deleted from the package across all its versions.

View Version History

For package developers, the Versions tab lists all the previous uploads of a package.

Click **Push Upgrades** to automatically upgrade subscribers to a specific version. Orgs entered in the Push Upgrade Exclusion List are omitted from the upgrade. The orgs can still install the upgrade when you publish the new version.

Click the version number of a listed upload to manage that upload. For more information, see Manage Versions of First-Generation Managed Packages on page 82.

Note: Push Upgrades is available for patches and major upgrades. Registered ISV partners can request Push Major Upgrade functionality. Log a support case in the Salesforce Partner Community.

The versions table displays the package attributes (in alphabetical order).

Attribute	Description
`Action`	Lists the actions you can perform on the package. The possible actions are: • **Deprecate**—Deprecates a package version. Users can no longer download or install this package. However, existing installations continue to work. • **Undeprecate**—Enables a previously deprecated package version to be installed again.
`Status`	The status of the package. The possible statuses are: • Released: The package is Managed - Released. • Beta: The package is Managed - Beta. • Deprecated: The package version is deprecated.
`Version Name`	The version name for this package. The version name is the marketing name for a specific release of a package. It's more descriptive than `Version Number`.
`Version Number`	The version number for the latest installed package version. The format is *`majorNumber.minorNumber.patchNumber`*, such as 2.1.3. The version number represents a release of a package. `Version Name` is a more descriptive name for the release. The `patchNumber` is generated only when you create a patch. If there's no `patchNumber`, it's assumed to be zero (0).

View Patch Development Orgs

Each patch is developed in a *patch development org*, which is the org where patch versions are developed, maintained, and uploaded. To start developing a patch, create a patch development org. Create and Upload Patches Patch development orgs permit developers to change existing components without causing incompatibilities between existing subscriber installations. Click **New** to create a patch for this package.

The Patch Organizations table lists all the patch development orgs created. It lists these attributes (in alphabetical order).

Attribute	Description
`Action`	Lists the actions that you can perform on a patch development org. The possible actions are: • **Login**—Log in to your package version. • **Reset**—Emails a temporary password for your patch development org.
`Administrator Username`	The login associated with the patch org.
`Patching Major Release`	The package version number that you're patching.

Notifications for Package Errors

Accurately track failed package installations, upgrades, and uninstallations in subscriber orgs with the Notifications for Package Errors feature. Proactively address issues with managed and unmanaged packages and provide support to subscribers so that they can successfully install and upgrade your apps.

You can choose to send a notification to an email address in your org when a subscriber's attempt to install, upgrade, or uninstall a packaged app fails. To enable this feature, contact your Salesforce representative.

Errors can happen with these package operations:

- Installation
- Upgrade
- Push upgrade
- Uninstallation

When an installation fails, an email is sent to the specified address with the following details:

- Reason for the failure
- Subscriber org information
- Metadata of the package that wasn't installed properly
- Who attempted to install the package

This example email is for a package installation that failed because the base package wasn't installed before the subscriber tried to install an extension.

```
On Mon, Jul 13, 2022 at 11:51 AM, NO REPLY <no-reply@salesforce.com> wrote:
The install of your package failed. Here are the details:

Error Message: 00DD00000007uJp: VALIDATION_FAILED [DB 0710 DE1 Pkg1 1.2: A required package
 is missing: Package "DB 0710 DE1 Pkg1", Version 1.2 or later must be installed first.]
Date/Time of Occurrence = Mon Jul 13 18:51:20 GMT 2015

Subscriber Org Name = DB 071015 EE 1
Subscriber Org ID = 00DD00000007uJp
Subscriber Org Status = TRIAL
Subscriber Org Edition = Enterprise Edition

Package Name = DB 0710 DE2 Pkg1
Package ID = 033D000000060EE
Package Namespace = DB_0710_DE2
Package Type = MANAGED
Package Version Name = 1.2
Package Version Number = 1.2
Package Version Id = 04tD00000006QoF

Installer Name = Admin User
Installer Email Address = db@salesforce.com
```

Set the Notification Email Address
Specify which address to email when a package installation, upgrade, or uninstallation fails.

Set the Notification Email Address

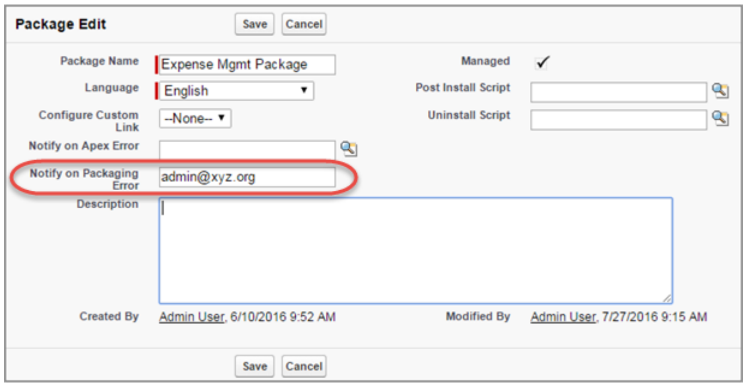
Specify which address to email when a package installation, upgrade, or uninstallation fails.

Notifications are sent only for package versions that are uploaded after the address is added. For example, if you upload package version 1.0 and then set the notification address, notifications aren't sent for failures related to version 1.0. Notifications start when version 2.0 is uploaded.

Also, you can't change or remove the notification email address for the package after it's been uploaded.

1. To enable this feature, contact your Salesforce representative.
2. From Setup, enter *`Packages`* in the `Quick Find` box, then select **Packages**.
3. Click the package name, and then click **Edit** on the package detail page.
4. Enter the email address to send notifications to, and click **Save**.

Notifications for Package Errors Configured in a Partner Org



Package Name	Expense Mgmt Package	Managed	✓
Language	English	Post Install Script	
Configure Custom Link	--None--	Uninstall Script	
Notify on Apex Error			
Notify on Packaging Error	admin@xyz.org		
Description			
Created By	Admin User, 6/10/2016 9:52 AM	Modified By	Admin User, 7/27/2016 9:15 AM

Create a First-Generation Managed Package using Salesforce DX

If you're an ISV, you want to build a managed package. A managed package is a bundle of components that make up an application or piece of functionality. A managed package is a great way to release an app for sale and to support licensing your features. You can protect intellectual property because the source code of many components isn't available through the package. You can also roll out upgrades to the package.

When you're working with your production org, you create a .zip file of metadata components and deploy them through Metadata API. The .zip file contains:

- A package manifest (`package.xml`) that lists what to retrieve or deploy
- One or more XML components organized into folders

If you don't have the packaged source already in the source format, you can retrieve it from the org and convert it using the CLI.

Build and Release Your App with Managed Packages
If you developed and tested your app, you're well on your way to releasing it. Luckily, when it's time to build and release an app as a managed package, you've got options. You can package an app you developed from scratch. If you're experimenting, you can also build the sample app from Salesforce and emulate the release process.

View Information About a Package
View the details about a specific package version, including its metadata package ID, package name, release state, and build number.

Build and Release Your App with Managed Packages

If you developed and tested your app, you're well on your way to releasing it. Luckily, when it's time to build and release an app as a managed package, you've got options. You can package an app you developed from scratch. If you're experimenting, you can also build the sample app from Salesforce and emulate the release process.

Working with a package is an iterative process. You typically retrieve, convert, and deploy source multiple times as you create scratch orgs, test, and update the package components.

Chances are, you already have a namespace and package defined in your packaging org. If not, run this command to open the packaging org in your browser.

```
sf org open --target-org me@my.org --path lightning/setup/Package/home
```

In the Salesforce UI, you can define a namespace and a package. Each packaging org can have a single managed package and one namespace.

Be sure to link the namespace to your Dev Hub org.

Packaging Checklist
Ready to deploy your packaging metadata and start creating a package? Take a few minutes to verify that you covered the items in this checklist, and you're good to go.

Deploy the Package Metadata to the Packaging Org
Before you deploy the package metadata into your packaging org, you convert from source format to metadata format.

Create a Beta Version of Your App
Test your app in a scratch org, or share the app for evaluation by creating a beta version.

Install the Package in a Target Org
After you create a package with the CLI, install the package in a target org. You can install the package in any org you can authenticate, including a scratch org.

Create a Managed Package Version of Your App
After your testing is done, your app is almost ready to be published in your enterprise or on AppExchange. Generate a new managed package version in your Dev Hub org.

SEE ALSO:

Link a Namespace to a Dev Hub Org

Retrieve Source from an Existing Managed Package

Packaging Checklist

Ready to deploy your packaging metadata and start creating a package? Take a few minutes to verify that you covered the items in this checklist, and you're good to go.

1. Link the namespace of each package you want to work with to the Dev Hub org.
2. Copy the metadata of the package from your version control system to a local project.
3. Update the config files, if needed.

   For example, to work with managed packages, sfdx-project.json must include the namespace.

   ```
   "namespace": "acme_example",
   ```

4. (Optional) Create an alias for each org you want to work with.

   If you haven't yet created an alias for each org, consider doing that now. Using aliases is an easy way to switch between orgs when you're working in the CLI.

5. Authenticate the Dev Hub org.
6. Create a scratch org.

   A scratch org is different than a sandbox org. You specify the org shape using project-scratch.json. To create a scratch org and set it as the `defaultusername` org, run this command from the project directory.

   ```
   sf org create scratch --definition-file config/project-scratch-def.json
   ```

7. Push source to the scratch org.
8. Update source in the scratch org as needed.
9. Pull the source from the scratch org if you used declarative tools to make changes there.

With these steps complete, you're ready to deploy your package metadata to the packaging org.

Deploy the Package Metadata to the Packaging Org

Before you deploy the package metadata into your packaging org, you convert from source format to metadata format.

It's likely that you have some files that you don't want to convert to metadata format. Create a `.forceignore` file to indicate which files to ignore.

1. Convert from source format to the metadata format.
   ```
   sf project convert source --output-dir mdapi_output_dir --package-name
   managed_pkg_name
   ```
   Create the output directory in the root of your project, not in the package directory. If the output directory doesn't exist, it's created. Be sure to include the `--package-name` so that the converted metadata is added to the managed package in your packaging org.

2. Review the contents of the output directory.
   ```
   ls -lR mdapi_output_dir
   ```
3. Authenticate the packaging org, if needed. This example specifies the org with an alias called MyPackagingOrgAlias, which helps you refer to the org more easily in subsequent commands.
   ```
   sf org login web --alias MyPackagingOrgAlias
   ```
   You can also authenticate with an OAuth client ID: `sf org login web --client-id oauth_client_id`

4. Deploy the package metadata back to the packaging org.

```
sf project deploy start --metadata-dir mdapi_output_dir --target-org me@example.com
```

The `--target-org` is the username. Instead of the username, you can use `-u MyPackagingOrgAlias` to refer to your previously defined org alias. You can use other options, like `--wait` to specify the number of minutes to wait. Use the `--metadata-dir` parameter to provide the path to a zip file that contains your metadata. Don't run tests at the same time as you deploy the metadata. You can run tests during the package upload process.

A message displays the job ID for the deployment.

5. Check the status of the deployment.

   When you run `sf project deploy report`, the job ID and target username are stored, so you don't have to specify these required parameters to check the status. These stored values are overwritten when you run `sf project deploy start` again.

   If you want to check the status of a different deploy operation, specify the job ID on the command line, which overrides the stored job ID.

SEE ALSO:

Salesforce CLI Command Reference

How to Exclude Source When Syncing or Converting

Create a Beta Version of Your App

Test your app in a scratch org, or share the app for evaluation by creating a beta version.

If you specified the package name when you converted source to metadata format, both the changed and new components are automatically added to the package. Including the package name in that stage of the process lets you take full advantage of end-to-end automation.

If, for some reason, you don't want to include new components, you have two choices. You can omit the package name when you convert source or remove components from the package in the Salesforce UI before you create the package version.

Create the beta version of a managed package by running the commands against your packaging org, not the Dev Hub org.

1. Ensure that you've authorized the packaging org.

   ```
   sf org login web --set-default me@example.com
   ```

2. Create the beta version of the package.

   ```
   sf package1 version create --package-id package_id --name package_version_name
   ```

   You can get the package ID on the package detail page in the packaging org. If you want to protect the package with an installation key, add it now or when you create the released version of your package. The `--installation-key` supplied from the CLI is equivalent to the Password field that you see when working with packages through the Salesforce user interface. When you include a value for `--installation-key`, you or a subscriber must supply the key before you can install the package in a target org.

You're now ready to create a scratch org and install the package there for testing. By default, the `create` command generates a beta version of your managed package.

Later, when you're ready to create the Managed - Released version of your package, include the `-m` (`--managed-released true`) parameter.

Note: After you create a managed-released version of your package, many properties of the components added to the package are no longer editable. Refer to the *First-Generation Managed Packaging Developer Guide* to understand the differences between beta and managed-released versions of your package.

SEE ALSO:

Salesforce CLI Command Reference

Link a Namespace to a Dev Hub Org

Install the Package in a Target Org

After you create a package with the CLI, install the package in a target org. You can install the package in any org you can authenticate, including a scratch org.

If you want to create a scratch org and set it as the `defaultusername` org, run this command from the project directory.

```
sf org create scratch -definition-file config/project-scratch-def.json
```

To locate the ID of the package version to install, run `sf package1 version list`.

```
METADATAPACKAGEVERSIONID   METADATAPACKAGEID   NAME  VERSION  RELEASESTATE  BUILDNUMBER
────────────────────────   ─────────────────   ────  ───────  ────────────  ───────────
04txx000000069oAAA         033xx00000007coAAA  r00   1.0.0    Released      1
04txx000000069tAAA         033xx00000007coAAA  r01   1.1.0    Released      1
04txx000000069uAAA         033xx00000007coAAA  r02   1.2.0    Released      1
04txx000000069yAAA         033xx00000007coAAA  r03   1.3.0    Released      1
04txx000000069zAAA         033xx00000007coAAA  r04   1.4.0    Released      1
```

You can then copy the package version ID you want to install. For example, the ID 04txx000000069zAAA is for version 1.4.0.

1. Install the package. You supply the package alias or version ID, which starts with 04t, in the required `--package` parameter.

   ```
   sf package install --package 04txx000000069zAAA
   ```

   If you've set a default target org, the package is installed there. You can specify a different target org with the `--target-org` parameter. If the package is protected by an installation key, supply the key with the `--installation-key` parameter.

To uninstall a package, open the target org and choose **Setup**. On the Installed Packages page, locate the package and choose **Uninstall**.

Create a Managed Package Version of Your App

After your testing is done, your app is almost ready to be published in your enterprise or on AppExchange. Generate a new managed package version in your Dev Hub org.

Ensure that you've authorized the packaging org and can view the existing package versions.

```
sf org login web --instance-url https://test.salesforce.com --set-default org_alias
```

View the existing package versions for a specific package to get the ID for the version you want to install.

```
sf package1 version list --package-id 033...
```

To view details for all packages in the packaging org, run the command with no parameters.

More than one beta package can use the same version number. However, you can use each version number for only one *managed* package version. You can specify major or minor version numbers.

You can also include URLs for a post-installation script and release notes. Before you create a managed package, make sure that you've configured your developer settings, including the namespace prefix.

Note: After you create a managed package version, you can't change some attributes of Salesforce components used in the package. See: Components Available in Managed Packages for information on editable components.

1. Create the managed package. Include the `--managed-released` parameter.

   ```
   sf package1 version create --package-id 033xx00000007oi --name ”Spring 22” --description
    ”Spring 22 Release” --version 3.2 --managed-released
   ```

   You can use other options, like `--wait` to specify the number of minutes to wait.

   To protect the package with an installation key, include a value for `--installation-key`. Then, you or a subscriber must supply the key before you can install the package in a target org.

   After the managed package version is created, you can retrieve the new package version ID using `sf package1 version list`.

View Information About a Package

View the details about a specific package version, including its metadata package ID, package name, release state, and build number.

1. From the project directory, run this command, supplying a package version ID.
   `sf package1 version display -i 04txx000000069yAAA`
   The output is similar to this example.

   ```
   METADATAPACKAGEVERSIONID  METADATAPACKAGEID   NAME  VERSION  RELEASESTATE  BUILDNUMBER
   ────────────────────────  ──────────────────  ────  ───────  ────────────  ───────────
   04txx000000069yAAA        033xx00000007coAAA  r03   1.3.0    Released      1
   04txx000000069yAAA        033xx00000011coAAA  r03   1.4.0    Released      1
   ```

   View All Package Versions in the Org
   View the details about all package versions in the org.

   Package IDs
   When you work with packages using the CLI, the package IDs refer either to a unique package or a unique package version.

SEE ALSO:

Salesforce CLI Command Reference

View All Package Versions in the Org

View the details about all package versions in the org.

1. From the project directory, run the `list` command.
   `sf package1 version list`
   The output is similar to this example. When you view the package versions, the list shows a single package for multiple package versions.

   ```
   METADATAPACKAGEVERSIONID  METADATAPACKAGEID   NAME  VERSION  RELEASESTATE  BUILDNUMBER
   ────────────────────────  ──────────────────  ────  ───────  ────────────  ───────────
   ```

```
04txx000000069oAAA        033xx00000007coAAA  r00   1.0.0    Released      1
04txx000000069tAAA        033xx00000007coAAA  r01   1.1.0    Released      1
04txx000000069uAAA        033xx00000007coAAA  r02   1.2.0    Released      1
04txx000000069yAAA        033xx00000007coAAA  r03   1.3.0    Released      1
04txx000000069zAAA        033xx00000007coAAA  r04   1.4.0    Released      1
```

SEE ALSO:

Salesforce CLI Command Reference

Package IDs

When you work with packages using the CLI, the package IDs refer either to a unique package or a unique package version.

The relationship of package version to package is one-to-many.

ID Example	Description	Used Where
033xx00000007oi	Metadata Package ID	Generated when you create a package. A single package can have one or more associated package version IDs. The package ID remains the same, whether it has a corresponding beta or released package version.
04tA000000081MX	Metadata Package Version ID	Generated when you create a package version.

Components Available in First-Generation Managed Packages

Each metadata component that you include in a managed 1GP or 2GP package has certain rules that determine its behavior in a subscriber org. Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and installed.

Components

A component is one constituent part of a package. It defines an item, such as a custom object or a custom field. You can combine components in a package to produce powerful features or applications. In a managed package, some components can be upgraded while others can't.

For details about components supported in first-generation and second-generation managed packages, see Components Available in Managed Packages in the Second-Generation Managed Packaging Developer Guide.

Attributes

An attribute is a field on a component, such as the name of an email template or the `Allow Reports` checkbox on a custom object. The attributes associated with a non-upgradeable component are editable by both the package developer and the subscriber. On an upgradeable component in a managed package, some attributes can be edited by the developer, some can be edited by the subscriber, and some can't be edited by anyone.

Components Automatically Added to First-Generation Managed Packages

When adding components to your first-generation managed package, related components are automatically added. For example, if you add a Visualforce page to a package that references a custom controller, that Apex class is also added.

Protected Components in Managed Packages

Developers can mark certain components as *protected*. Protected components can't be linked to or referenced by components created in a subscriber org. A developer can delete a protected component in a future release without worrying about failing installations. However, after a component is marked as unprotected and is released globally, the developer can't delete it.

Set Up a Platform Cache Partition with Provider Free Capacity

Salesforce provides 3 MB of free Platform Cache capacity for security-reviewed managed packages. This is made available through a capacity type called Provider Free capacity and is automatically enabled in all Developer edition orgs.

Package Dependencies in First-Generation Managed Packages

Package dependencies are created when a component references another component, permission, or preference that is required for the component to be valid.

Metadata Access in Apex Code

Use the `Metadata` namespace in Apex to access metadata in your package.

Permission Sets and Profile Settings in Packages

Permission sets, permission set groups, and profile settings are all ways to grant permissions and other access settings to a package. Only use a profile setting if permission sets don't support the specific access you need to grant. In all other instances, use permission sets or permission set groups.

Permission Set Groups

You can organize permission sets into groups and include them in first and second-generation managed packages. Permission set groups can be updated when you upgrade the package.

Custom Profile Settings

Create profiles to define how users access objects and data, and what they can do within your app. For example, profiles specify custom object permissions and the tab visibility for your app. When installing or upgrading your app, admins can associate your custom profiles with existing non-standard profiles. Permissions in your custom profile that are related to new components created as part of the install or upgrade are added to the existing profile. The security settings associated with standard objects and existing custom objects in an installer's organization are unaffected.

Protecting Your Intellectual Property

The details of your custom objects, custom links, reports, and other installed items are revealed to installers so that they can check for malicious content. However, revealing an app's components prevents developers from protecting some intellectual property.

Call Salesforce URLs Within a Package

The URLs that Salesforce serves for a target org vary based on the org type and configuration. To build packages that support all possible URL formats, use relative URLs whenever possible. If your package functionality requires a full URL, use the Apex `DomainCreator` class to get the corresponding hostname. This method allows your package to work in all orgs, regardless of the org type and My Domain settings.

Develop App Documentation

To help your subscribers make the most of your app, provide documentation about how to configure and customize your app.

API and Dynamic Apex Access in Packages

Apex Package components have access via dynamic Apex and the API to standard and custom objects in the organization where they're installed.

Connected Apps

A connected app is a framework that enables an external application to integrate with Salesforce using APIs and standard protocols, such as SAML, OAuth, and OpenID Connect. Connected apps use these protocols to authenticate, authorize, and provide single sign-on (SSO) for external apps. The external apps that are integrated with Salesforce can run on the customer success platform, other platforms, devices, or SaaS subscriptions. For example, when you log in to your Salesforce mobile app and see your data from your Salesforce org, you're using a connected app.

Components Automatically Added to First-Generation Managed Packages

When adding components to your first-generation managed package, related components are automatically added. For example, if you add a Visualforce page to a package that references a custom controller, that Apex class is also added.

To understand what components are automatically included in first-generation managed packages, review the following list:

When you add this component	These components are automatically added
Action	Action target object (if it's a custom object), action target field, action record type, predefined field values, action layout; and any custom fields that the action layout or predefined values refer to on the target object
Apex class	Custom fields, custom objects, and other explicitly referenced Apex classes, and anything else that the Apex class references directly Note: If an Apex class references a custom label, and that label has translations, you must explicitly package the individual languages desired for those translations to be included.
Apex trigger	Custom fields, custom objects, and any explicitly referenced Apex classes, and anything else that the Apex trigger references directly
Article type	Custom fields, the default page layout
Compact layout	Custom fields
Custom app	Custom tabs (including web tabs), documents (stored as images on the tab), documents folder, asset files
Custom button or link	Custom fields and custom objects
Custom field	Custom objects
Custom home page layouts	Custom home page components on the layout
Custom settings	Apex sharing reasons, Apex sharing recalculations, Apex triggers, custom fields, list views, page layouts, record types, validation rules, or custom buttons or links.
Custom object	Custom fields, validation rules, page layouts, list views, custom buttons, custom links, record types, Apex sharing reasons, Apex sharing recalculations, and Apex triggers Note: • Apex sharing reasons are unavailable in extensions. • When packaged and installed, only public list views from an app are installed. If a custom object has any custom list views that you want to include in your package, ensure that the list view is accessible by all users.

When you add this component	These components are automatically added
Custom object (as an external object)	External data source, custom fields, page layouts, list views, custom buttons, and custom links Note: • When packaged and installed, only public list views from an app are installed. If an external object has any custom list views that you want to include in your package, ensure that the list view is accessible by all users. • In managed and unmanaged packages, external objects are included in the custom object component.
Custom tab	Custom objects (including all of its components), s-controls, and Visualforce pages
Dashboard	Folders, reports (including all of its components), s-controls, and Visualforce pages
Document	Folder
Email template (Classic)	• Folder • Letterhead • Custom fields • Documents (stored as images on the letterhead or template)
Email template (Lightning)	• Custom object • Custom field references (in Handlebars Merge Language syntax) • Enhanced folder (except the default public and private folders) • Inline images referencing Salesforce Files • Attachments referencing Salesforce Files For Lightning email templates created before Spring '21, attachments aren't automatically added to the package. Open and resave these templates to turn the attachments into content assets, which are then automatically added to the package These items aren't included and can't be added to a package: • Enhanced letterhead • The associated FlexiPage • CMS files (Account Engagement only)
Email template (Lightning) created in Email Template Builder	• Custom object • Custom field references (in Handlebars Merge Language syntax) • Enhanced folder (except the default public and private folders) • Inline images referencing Salesforce Files • Attachments referencing Salesforce Files • The associated FlexiPage These items aren't included and can't be added to a package: • Enhanced letterhead • CMS files (Account Engagement only)

When you add this component	These components are automatically added
External Credential	Permission set and authentication provider Note: External credentials that use the Oauth 2.0 authentication protocol must reference an authentication provider to capture the details of the authorization endpoint. If you add an external credential that references an authentication provider, the authentication provider is added to the package. See Authentication Providers for information on which elements of an authentication provider are and aren't packageable.
Field set	Any referenced fields
Lightning page	All Lightning resources referenced by the page, such as record types, actions, custom components, events, and interfaces. Custom fields, custom objects, list views, page layouts, Visualforce pages, and Apex classes referenced by the components on the page.
Lightning page tab	Lightning page
Flow	Custom objects, custom fields, Apex classes, and Visualforce pages
Folder	Everything in the folder
Lightning application	All Lightning resources referenced by the application, such as components, events, and interfaces. Custom fields, custom objects, list views, page layouts, and Apex classes referenced by the application.
Lightning component	All Lightning resources referenced by the component, such as nested components, events, and interfaces. Custom fields, custom objects, list views, page layouts, and Apex classes referenced by the component.
Lightning event	Custom fields, custom objects, list views, and page layouts
Lightning interface	Custom fields, custom objects, list views, and page layouts
Lightning web component	All Lightning web component resources referenced by the component, such as nested components and modules. Custom fields, custom objects, list views, page layouts, and Apex classes referenced by the component
Named Credential	External credential; for legacy named credentials, an authentication provider
Page layout	Actions, custom buttons, custom links, s-controls, and Visualforce pages
Permission set	Any custom permissions, external data sources, Visualforce pages, record types, and Apex classes that are assigned in the permission set
Record type	Record type mappings, compact layout
Report	Folder, custom fields, custom objects, custom report types, and custom s-controls
Reporting Snapshot	Reports
S-control	Custom fields and custom objects
Translation	Translated terms for the selected language on any component in the package
Validation rule	Custom fields (referenced in the formula)

When you add this component	These components are automatically added
Visualforce home page component	Associated Visualforce page
Visualforce pages	Apex classes that are used as custom controllers, Visualforce custom components, and referenced field sets
Workflow rule	All associated workflow alerts, field updates, outbound messages, and tasks; also, if the workflow rule is designed for a custom object, the custom object is automatically included

Note: Some package components, such as validation rules or record types, don't appear in the list of package components, but are included and install with the other components.

Protected Components in Managed Packages

Developers can mark certain components as *protected*. Protected components can't be linked to or referenced by components created in a subscriber org. A developer can delete a protected component in a future release without worrying about failing installations. However, after a component is marked as unprotected and is released globally, the developer can't delete it.

Developers can mark these components as protected in managed packages.

- Custom labels
- Custom links (for Home page only)
- Custom metadata types
- Custom objects
- Custom permissions
- Custom settings
- Workflow alerts
- Workflow field updates
- Workflow outbound messages
- Workflow tasks

Considerations for Protected Custom Objects in Subscriber Sandboxes

When a subscriber creates a partial copy sandbox, protected custom objects don't display in the list of objects to copy. Data contained in the records of protected custom objects is never copied to sandboxes, regardless of sandbox type.

Set Up a Platform Cache Partition with Provider Free Capacity

Salesforce provides 3 MB of free Platform Cache capacity for security-reviewed managed packages. This is made available through a capacity type called Provider Free capacity and is automatically enabled in all Developer edition orgs.

Follow the steps here to allocate the Provider Free capacity to a Platform Cache partition before adding it to your managed package.

Note: If a Platform Cache partition is already part of your managed package, you can choose to edit the existing partition and allocate the Provider Free capacity to it.

Create a partition from the Platform Cache page and then set it up to use the Provider Free capacity

1. From Setup, in the Quick Find box, enter `Platform Cache`, and then select **Platform Cache**.

As the Provider Free capacity is automatically enabled in all Developer edition orgs, the Org's Capacity Breakdown donut chart shows the Provider Free capacity.

2. Click **New Platform Cache Partition**.
3. In the `Label` box, enter a name for the partition. The name can contain alphanumeric characters only and must be unique in your org.
4. In the `Description` box, enter an optional description for the partition.
5. In the Capacity section, allocate separate capacities for session cache and org cache from the available Provider Free capacity.
6. Save the new Platform Cache partition.

You can add this new Platform Cache partition to your managed package. When a security-reviewed managed package with Platform Cache partition is installed on the subscriber org, the Provider Free capacity is allocated and automatically made available to the installed partition. The managed package can start using the Platform Cache partition; no post-install script or manual allocation is required.

Note: If the managed package is not AppExchange-certified and security-reviewed, the Provider Free capacity resets to zero and will not be allocated to the installed Platform Cache partition.

When a Platform Cache partition with Provider Free capacity is installed in a subscriber org, the Provider Free capacity allocated is non-editable. The provider free capacity of one installed partition can't be used for any other partition.

Tip: After you install a Platform Cache partition with Provider Free capacity, you can edit the partition and make additional allocations from the available platform cache capacity of the org.

Package Dependencies in First-Generation Managed Packages

Package dependencies are created when a component references another component, permission, or preference that is required for the component to be valid.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

Packages, Apex classes, Apex triggers, Visualforce components, and Visualforce pages can have dependencies on components within an org. These dependencies are recorded on the Show Dependencies page.

Dependencies are important for packaging because any dependency in a component of a package is considered a dependency of the package as a whole.

Note: An installer's organization must meet all dependency requirements listed on the Show Dependencies page or else the installation fails. For example, the installer's org must have divisions enabled to install a package that references divisions.

Dependencies are important for Apex classes or triggers. Any component on which a class or trigger depends must be included with the class or trigger when the code is deployed or packaged.

In addition to dependencies, the *operational scope* is also displayed on the Show Dependencies page. The operational scope is a table that lists any data manipulation language (DML) operations (such as `insert` or `merge`) that Apex executes on a specified object. The operational scope can be used when installing an application to determine the full extent of the application's database operations.

EDITIONS

AppExchange packages and Visualforce are available in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Apex available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To upload packages:
- Upload AppExchange Packages

To view Visualforce dependencies:
- Developer Mode

To view the dependencies and operational scope for a package, Apex class, Apex trigger, or Visualforce page:

1. Navigate to the appropriate component from Setup.
   - For packages, enter *`Packages`* in the `Quick Find` box, then select **Packages**.
   - For Apex classes, enter *`Apex Classes`* in the `Quick Find` box, then select **Apex Classes**.
   - For Apex triggers, from the management settings for the appropriate object, go to Triggers.
   - For Visualforce pages, enter *`Visualforce Pages`* in the `Quick Find` box, then select **Visualforce Pages**.
2. Select the name of the component.
3. Click **View Dependencies** for a package, or **Show Dependencies** for all other components, to see a list of objects that depend upon the selected component.

If a list of dependent objects displays, click **Fields** to access the field-level detail of the operational scope. The field-level detail includes information, such as whether Apex updates a field. For more information, see Field Operational Scope.

Packages, Apex code, and Visualforce pages can depend many components, including but not limited to:

- Custom field definitions
- Validation formulas
- Reports
- Record types
- Apex
- Visualforce pages and components

For example, if a Visualforce page includes a reference to a multicurrency field, such as `{!contract.ISO_code}`, that Visualforce page has a dependency on multicurrency. If a package contains this Visualforce page, it also has a dependency on multicurrency. Any organization that wants to install this package must have multicurrency enabled.

Metadata Access in Apex Code

Use the `Metadata` namespace in Apex to access metadata in your package.

Your package may need to retrieve or modify metadata during installation or update. The `Metadata` namespace in Apex provides classes that represent metadata types, as well as classes that let you retrieve and deploy metadata components to the subscriber org. These considerations apply to metadata in Apex:

- You can create, retrieve, and update metadata components in Apex code, but you can't delete components.
- You can currently access records of custom metadata types and page layouts in Apex.
- Managed packages not approved by Salesforce can't access metadata in the subscriber org, unless the subscriber org enables the **Allow metadata deploy by Apex from non-certified Apex package version** org preference. Use this org preference when doing test or beta releases of your managed packages.

If your package accesses metadata during installation or update, or contains a custom setup interface that accesses metadata, you must notify the user. For installs that access metadata, notify the user in the description of your package. The notice should let customers know that your package has the ability to modify the subscriber org's metadata.

You can write your own notice, or use this sample:

`This package can access and change metadata outside its namespace in the Salesforce org where it's installed.`

Salesforce verifies the notice during the security review.

For more information, see Metadata in the *Apex Developer Guide*.

Permission Sets and Profile Settings in Packages

EDITIONS

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Permission sets are available in: **Contact Manager**, **Professional**, **Group**, **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

Permission sets, permission set groups, and profile settings are all ways to grant permissions and other access settings to a package. Only use a profile setting if permission sets don't support the specific access you need to grant. In all other instances, use permission sets or permission set groups.

Important: Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

Behavior	Permission Sets	Profile Settings
What permissions and settings are included?	• Assigned custom apps • Custom object permissions • External object permissions • Custom field permissions • Custom metadata types permissions • Custom permissions • Custom settings permissions • Custom tab visibility settings • Apex class access • Visualforce page access • External data source access • Record types Note: Although permission sets include standard tab visibility settings, these settings can't be packaged as permission set components. If a permission set includes an assigned custom app, it's possible that a subscriber can delete the app. In that case, when the package is later upgraded, the assigned custom app is removed from the permission set.	• Assigned custom apps • Assigned connected apps • Tab settings • Page layout assignments • Record type assignments • Custom field permissions • Custom metadata type permissions • Custom object permissions • Custom permissions • Custom settings permissions • External object permissions • Apex class access • Visualforce page access • External data source access

Behavior	Permission Sets	Profile Settings
Can they be upgraded in managed packages?	Yes.	Profile settings are applied to existing profiles in the subscriber's org on install or upgrade. Only permissions related to new components created as part of the install or upgrade are applied.
Can subscribers edit them?	No.	Yes.
Can you clone or create them?	Yes. However, if a subscriber clones a permission set or creates one that's based on a packaged permission set, it isn't updated in subsequent upgrades. Only the permission sets included in a package are upgraded.	Yes. Subscribers can clone any profile that includes permissions and settings related to packaged components.
Do they include standard object permissions?	No. Also, you can't include object permissions for a custom object in a master-detail relationship where the master is a standard object.	No.
Do they include user permissions?	No.	No.
Are they included in the installation wizard?	No. Subscribers must assign permission sets after installation.	Yes. Profile settings are applied to existing profiles in the subscriber's org on install or upgrade. Only permissions related to new components created as part of the install or upgrade are applied. Affected components (listed with the developerName) can include new: • Fields (CustomField) • Objects (CustomObject), • Tabs (CustomTab) • Apps (CustomApplication) • Apex classes (ApexClass) • Apex pages (ApexPage) • Layouts (Layout) • Record types (RecordType) • Custom permissions (CustomPermission) • Custom settings (CustomSetting) • Custom metadata types (CustomMetadata)
What are the user license requirements?	A permission set is only installed if the subscriber org has at least one user license that matches the permission set. For example, permission sets with the Salesforce	None. In a subscriber org, the installation overrides the profile settings, not their user licenses.

Behavior	Permission Sets	Profile Settings
	Platform user license aren't installed in an org that has no Salesforce Platform user licenses. If a subscriber later acquires a license, the subscriber must reinstall the package to get the permission sets associated with the newly acquired license. Permission sets with no user license are always installed. If you assign a permission set that doesn't include a user license, the user's existing license must allow its enabled settings and permissions. Otherwise, the assignment fails.	
How are they assigned to users?	Subscribers must assign packaged permission sets after installing the package.	Profile settings are applied to existing profiles.
Can permission sets in an extension package grant access to objects installed in a base package?	A permission set in the extension package can't modify access permissions for either the parent objects in the base package or the associated child objects in the extension package.	Same behavior as for permission sets.

Best Practices

- If users need access to apps, standard tabs, page layouts, and record types, don't use permission sets as the sole permission-granting model for your app.
- Create packaged permission sets that grant access to the custom components in a package, but not standard Salesforce components.

Permission Set Groups

You can organize permission sets into groups and include them in first and second-generation managed packages. Permission set groups can be updated when you upgrade the package.

Keep these considerations in mind when you organize permission sets into groups to include in your managed packages:

Important: You can't include object permissions for standard objects in managed packages. During package installation, all object permissions for standard objects are ignored, and aren't installed in the org.

Also:

- You can't add permission sets constrained by a permission set license to managed or unmanaged packages.
- You can only package permissions for metadata that's included in your package.

- You can add or remove permission sets in permission set groups as part of a package upgrade. Subscribers can also modify the permission set groups by muting permissions or adding or removing local permissions sets.

SEE ALSO:

Create a Permission Set Group

Permission Set Groups Considerations

Custom Profile Settings

Create profiles to define how users access objects and data, and what they can do within your app. For example, profiles specify custom object permissions and the tab visibility for your app. When installing or upgrading your app, admins can associate your custom profiles with existing non-standard profiles. Permissions in your custom profile that are related to new components created as part of the install or upgrade are added to the existing profile. The security settings associated with standard objects and existing custom objects in an installer's organization are unaffected.

Consider these tips when creating custom profiles for apps you want to publish.

- Give each custom profile a name that identifies the profile as belonging to the app. For example, if you're creating a Human Resources app named "HR2GO," a good profile name would be "HR2GO Approving Manager."
- If your custom profiles have a hierarchy, use a name that indicates the profile's location in the hierarchy. For example, name a senior-level manager's profile "HR2GO Level 2 Approving Manager."
- Avoid custom profile names that can be interpreted differently in other organizations. For example, the profile name "HR2GO Level 2 Approving Manager" is open to less interpretation than "Sr. Manager."
- Provide a meaningful description for each profile. The description displays to the user installing your app.

Alternatively, you can use permission sets to maintain control of permission settings through the upgrade process. Permission sets contain a subset of profile access settings, including object permissions, field permissions, Apex class access, and Visualforce page access. These permissions are the same as those available on profiles. You can add a permission set as a component in a package.

Note: In packages, assigned apps and tab settings aren't included in permission set components.

Protecting Your Intellectual Property

The details of your custom objects, custom links, reports, and other installed items are revealed to installers so that they can check for malicious content. However, revealing an app's components prevents developers from protecting some intellectual property.

To protect your intellectual property, consider the following:

- Only publish package components that are your intellectual property and that you have the rights to share.
- After your components are available on AppExchange, you can't recall them from anyone who has installed them.
- The information in the components that you package and publish might be visible to customers. Use caution when adding your code to a formula, Visualforce page, or other component that you can't hide in your app.
- The code contained in an Apex class, trigger, Lightning, or Visualforce component that's part of a managed package is obfuscated and can't be viewed in an installing org. The only exceptions are methods declared as global. You can view global method signatures in an installing org. In addition, License Management Org users with the View and Debug Managed Apex permission can view their packages' obfuscated Apex classes when logged in to subscriber orgs via the Subscriber Support Console.
- If a custom setting is contained in a managed package, and the `Visibility` is specified as Protected, the custom setting isn't contained in the list of components for the package on the subscriber's org. All data for the custom setting is hidden from the subscriber.

Call Salesforce URLs Within a Package

The URLs that Salesforce serves for a target org vary based on the org type and configuration. To build packages that support all possible URL formats, use relative URLs whenever possible. If your package functionality requires a full URL, use the Apex `DomainCreator` class to get the corresponding hostname. This method allows your package to work in all orgs, regardless of the org type and My Domain settings.

The formats for My Domain URLs vary between production and sandbox orgs. With partitioned domains, hostname formats also vary for demo, Developer Edition, free, patch, and scratch orgs, plus Trailhead playgrounds. For example, there are currently two possible formats for sandbox My Domain login hostname formats and ten possible Visualforce hostname formats. For more information, see My Domain URL Formats and Partitioned Domains in Salesforce Help.

In general, use relative URLs whenever possible within your packages. If a full URL is required, use the `System.DomainCreator` Apex class to get the URL's hostname.

Note: The `System.DomainCreator` Apex class is available in API version 54.0 and later.

Use the My Domain Login URL for Logins

All Salesforce orgs have a My Domain, an org-specific subdomain for the URLs that Salesforce hosts for that org. Customers have the option to prevent user and SOAP API logins from the generic `login.salesforce.com` and `test.salesforce.com` hostnames. When those options are enabled, logins require the My Domain login URL.

To get the My Domain login URL format for an org, use the `getOrgMyDomainHostname()` method of the `System.DomainCreator` Apex class.

```
//Get the My Domain login hostname
String myDomainHostname = DomainCreator.getOrgMyDomainHostname();
```

In this case, in a production org with a My Domain name of `mycompany`, `myDomainHostname` returns `mycompany.my.salesforce.com`.

Use Relative URLs

Whenever possible, we recommend that you use a relative URL, which only includes the path within your packages.

For example, assume that you want to add a link on the Visualforce page with a URL of `https://`*`MyDomainName`*`--`*`PackageName`*`.vf.force.com/apex/myCases` to a Visualforce page with the URL, `https://`*`MyDomainName`*`--`*`PackageName`*`.vf.force.com/apex/newCase`. In this case, use the relative path when referencing the page: `/apex/newCase`.

Generate Hostnames for Full URLs

Sometimes a full URL is required. For example, when your package delivers a Visualforce page that includes content delivered by your package. If your package includes full URLs, use the `System.DomainCreator` Apex class to get the associated hostnames. Otherwise, users can experience issues with your package functionality.

For example, to return the hostname for Visualforce pages, use the `getVisualforceHostname(packageName)` method of the `System.DomainCreator` Apex class.

```
//Define the name of your package as a string
String packageName = 'abcpackage';

//Get the Visualforce hostname
```

```
String vfHostname = DomainCreator.getVisualforceHostname(packageName);

//Build the URL for creating a new case
System.URL vfNewCaseUrl = new URL('https', vfHostname, '/apex/newCase');
```

In this example, in a production org with enhanced domains and a My Domain name of `mycompany`, `vfNewCaseUrl` returns `https://mycompany--abcpackage.vf.force.com/apex/newCase`.

Get Part of a Domain

If you find code in your package that parses a known URL or domain to get a value, we recommend that you update that code to use one of the newer Apex classes. Code that assumes a specific URL format can fail.

If you need a hostname, assess whether you can use the `System.DomainCreator` class.

If you need that value for another reason, use the Apex `System.DomainParser` or `System.Domain` class instead.

In this example, we parse a known URL to get the domain type, the org's My Domain name, and the package name.

```
//Parse a known URL
System.Domain domain = DomainParser.parse('https://mycompany--abcpackage.vf.force.com');

//Get the domain type
System.DomainType domainType = domain.getDomainType(); // Returns VISUALFORCE_DOMAIN

//Get the org’s My Domain name
String myDomainName = domain.getMyDomainName(); // Returns mycompany

//Get the package name
String packageName = domain.getPackageName(); // Returns abcpackage
```

Develop App Documentation

To help your subscribers make the most of your app, provide documentation about how to configure and customize your app.

- Configure Option—You can include a **Configure** option for installers. This option can link to installation and configuration details, such as:
  - Provisioning the external service of a client app
  - Custom app settings

  The **Configure** option is included in your package as a custom link. You can create a custom link for your home page layouts and add it to your package.

  1. Create a custom link to a URL that contains configuration information, or a Visualforce page that implements the configuration. When you create your custom link, set the display properties to `Open in separate popup window` so that the user returns to the same Salesforce page when done.
  2. When you create the package, choose this custom link in the `Configure Custom Link` field of your package detail.
- Data Sheet—Give installers the fundamental information they must know about your app before they install.
- Customization and Enhancement Guide—Let installers know what they must customize after installation as part of their implementation.
- Custom Help—You can provide custom help for your custom object records and custom fields.

API and Dynamic Apex Access in Packages

Apex Package components have access via dynamic Apex and the API to standard and custom objects in the organization where they're installed.

`API Access` is a package setting that controls the dynamic Apex and API access that package components have to standard and custom objects. The setting displays for both the developer and installer on the package detail page. With this setting:

- The developer of an AppExchange package can restrict API access for a package before uploading it to AppExchange. After it's restricted, the package components receive Apex and API sessions that are restricted to the custom objects in the package. The developer can also enable access to specific standard objects, and any custom objects in other packages that this package depends on.
- The installer of a package can accept or reject package access privileges when installing the package to their organization.
- After installation, an administrator can change Apex and API access for a package at any time. The installer can also enable access on additional objects such as custom objects created in the installer's organization or objects installed by unrelated packages.

There are two possible options for the `API Access` setting:

- The default `Unrestricted`, which gives the package components the same API access to standard objects as the user who is logged in when the component sends a request to the API. Apex runs in system mode. Unrestricted access gives Apex read access to all standard and custom objects.
- `Restricted`, which allows the administrator to select which standard objects the components in the package can access. Further, the components in restricted packages can only access custom objects in the current package if the user has the object permissions that provide access to them.

Considerations for API and Dynamic Apex Access in Packages

By default, dynamic Apex can only access the components with which the code is packaged. To provide access to standard objects not included in the package, the developer must set the `API Access`.

1. From Setup, enter *`Packages`* in the `Quick Find` box, then select **Packages**.
2. Select the package that contains a dynamic Apex that needs access to standard objects in the installing organization.
3. In the Package Detail related list, click **Enable Restrictions** or `Restricted`, whichever is available.
4. Set the access level (Read, Create, Edit, Delete) for the standard objects that the dynamic Apex can access.
5. Click **Save**.

Choosing `Restricted` for the `API Access` setting in a package affects the following:

- API access in a package overrides the following user permissions:
  - Author Apex
  - Customize Application
  - Edit HTML Templates
  - Edit Read Only Fields
  - Manage Billing
  - Manage Call Centers
  - Manage Categories
  - Manage Custom Report Types
  - Manage Dashboards
  - Manage Letterheads

  - Manage Package Licenses
  - Manage Public Documents
  - Manage Public List Views
  - Manage Public Reports
  - Manage Public Templates
  - Manage Users
  - Transfer Record
  - Use Team Reassignment Wizards
  - View Setup and Configuration
  - Weekly Export Data

- If `Read`, `Create`, `Edit`, and `Delete` access aren't selected in the API access setting for objects, users don't have access to those objects from the package components, even if the user has the Modify All Data and View All Data permissions.
- A package with `Restricted` API access can't create users.
- Salesforce denies access to Web service and `executeanonymous` requests from an AppExchange package that has `Restricted` access.

The following considerations also apply to API access in packages:

- Workflow rules and Apex triggers fire regardless of API access in a package.
- If a component is in more than one package in an organization, API access is unrestricted for that component in all packages in the organization regardless of the access setting.
- If Salesforce introduces a new standard object after you select restricted access for a package, access to the new standard object isn't granted by default. You must modify the restricted access setting to include the new standard object.
- When you upgrade a package, changes to the API access are ignored even if the developer specified them, which ensures that the administrator installing the upgrade has full control. Installers must carefully examine the changes in package access in each upgrade during installation and note all acceptable changes. Then, because those changes are ignored, the administrator must manually apply any acceptable changes after installing an upgrade.
- S-controls are served by Salesforce and rendered inline in Salesforce. Because of this tight integration, there are several means by which an s-control in an installed package could escalate its privileges to the user's full privileges. In order to protect the security of organizations that install packages, s-controls have the following limitations:
  - For packages you're developing (that is, not installed from AppExchange), you can only add s-controls to packages with the default `Unrestricted` API access. After a package has an s-control, you can't enable `Restricted` API access.
  - For packages you've installed, you can enable access restrictions even if the package contains s-controls. However, access restrictions provide only limited protection for s-controls. Salesforce recommends that you understand the JavaScript in an s-control before relying on access restriction for s-control security.
  - If an installed package has `Restricted` API access, upgrades are successful only if the upgraded version doesn't contain any s-controls. If s-controls are present in the upgraded version, you must change the currently installed package to `Unrestricted` API access.

Manage API and Dynamic Apex Access in Packages

`API Access` is a package setting that controls the dynamic Apex and API access that package components have to standard and custom objects. The setting displays for both the developer and installer on the package detail page.

Configure Default Package Versions for API Calls

You can specify the default package versions for enterprise API and partner API calls.

About the Partner WSDL

The Partner Web Services WSDL is used for client applications that are metadata-driven and dynamic in nature. It's particularly—but not exclusively—useful to Salesforce partners who are building client applications for multiple organizations.

Generate an Enterprise WSDL with Managed Packages

If you're downloading an enterprise WSDL and you have managed packages installed in your organization, you must take an extra step to select the version of each installed package to include in the generated WSDL.

Work with Services Outside of Salesforce

Manage API and Dynamic Apex Access in Packages

`API Access` is a package setting that controls the dynamic Apex and API access that package components have to standard and custom objects. The setting displays for both the developer and installer on the package detail page.

- The developer of an AppExchange package can restrict API access for a package before uploading it to AppExchange. After it's restricted, the package components receive Apex and API sessions that are restricted to the custom objects in the package. The developer can also enable access to specific standard objects, and any custom objects in other packages that this package depends on.
- The installer of a package can accept or reject package access privileges when installing the package to their organization.
- After installation, an administrator can change Apex and API access for a package at any time. The installer can also enable access on additional objects such as custom objects created in the installer's organization or objects installed by unrelated packages.

USER PERMISSIONS

To edit API and dynamic Apex access for a package you've created or installed:
- Create AppExchange packages

To accept or reject package API and dynamic Apex access for a package as part of installation:
- Download AppExchange packages

Setting API and Dynamic Apex Access in Packages

To change package access privileges in a package you or someone in your organization has created:

1. From Setup, enter *`Packages`* in the `Quick Find` box, then select **Packages**.
2. Select a package.
3. The `API Access` field displays the current setting, `Restricted` or `Unrestricted`, and a link to either **Enable Restrictions** or **Disable Restrictions**. If `Read`, `Create`, `Edit`, and `Delete` access aren't selected in the API access setting for objects, users don't have access to those objects from the package components, even if the user has the Modify All Data and View All Data permissions.

   Use the `API Access` field to:

   - **Enable Restrictions**— This option is available only if the current setting is `Unrestricted`. Select this option if you want to specify the dynamic Apex and API access that package components have to standard objects in the installer's organization. When you select this option, the Extended Object Permissions list is displayed. To enable access for each object in the list, select the `Read`, `Create`, `Edit`, or `Delete` checkboxes. This selection is disabled in some situations. Click **Save** when finished. For more information about choosing the `Restricted` option, including information about when it's disabled, see Considerations for API and Dynamic Apex Access in Packages on page 52.
   - **Disable Restrictions**—This option is available only if the current setting is `Restricted`. Select this option if you don't want to restrict the Apex and API access privileges that the components in the package have to standard and custom objects. This option gives all the components in the package the same API access as the user who is logged in. For example, if a user can access accounts, an Apex class in the package that accesses accounts would succeed when triggered by that user.
   - **Restricted**—Click this link if you already restricted API access and wish to edit the restrictions.

Accepting or Rejecting API and Dynamic Apex Access Privileges during Installation

To accept or reject the API and dynamic Apex access privileges for a package you're installing:

- Start the installation process on AppExchange.
- In **Approve API Access**, either accept by clicking **Next**, or reject by clicking **Cancel**. Complete the installation steps if you haven't canceled.

Changing API and Dynamic Apex Access Privileges After Installation

To edit the package API and dynamic Apex access privileges after you've installed a package:

1. From Setup, enter *`Installed Packages`* in the `Quick Find` box, then select **Installed Packages**.
2. Click the name of the package you wish to edit.
3. The `API Access` field displays the current setting, `Restricted` or `Unrestricted`, and a link to either **Enable Restrictions** or **Disable Restrictions**. If `Read`, `Create`, `Edit`, and `Delete` access aren't selected in the API access setting for objects, users don't have access to those objects from the package components, even if the user has the Modify All Data and View All Data permissions.

   Use the `API Access` field to:

   - **Enable Restrictions**— This option is available only if the current setting is `Unrestricted`. Select this option if you want to specify the dynamic Apex and API access that package components have to standard objects in the installer's organization. When you select this option, the Extended Object Permissions list is displayed. To enable access for each object in the list, select the `Read`, `Create`, `Edit`, or `Delete` checkboxes. This selection is disabled in some situations. Click **Save** when finished. For more information about choosing the `Restricted` option, including information about when it's disabled, see Considerations for API and Dynamic Apex Access in Packages on page 52.
   - **Disable Restrictions**—This option is available only if the current setting is `Restricted`. Select this option if you don't want to restrict the Apex and API access privileges that the components in the package have to standard and custom objects. This option gives all the components in the package the same API access as the user who is logged in. For example, if a user can access accounts, an Apex class in the package that accesses accounts would succeed when triggered by that user.
   - **Restricted**—Click this link if you have already restricted API access and wish to edit the restrictions.

Configure Default Package Versions for API Calls

You can specify the default package versions for enterprise API and partner API calls.

A package version is a number that identifies the set of components uploaded in a package. The version number has the format *`majorNumber.minorNumber.patchNumber`* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *`patchNumber`* is generated and updated only for a patch release. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package.

Default package versions for API calls provide fallback settings if package versions aren't provided by an API call. Many API clients don't include package version information, so the default settings maintain existing behavior for these clients.

You can specify the default package versions for enterprise API and partner API calls. The enterprise WSDL is for customers who want to build an integration with their Salesforce organization only. It's strongly typed, which means that calls operate on objects and fields with specific data types, such as `int` and `string`. The partner WSDL is for customers, partners, and ISVs who want to build

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer**, Editions

USER PERMISSIONS

To configure default package versions for API calls:
- Customize Application

an integration that can work across multiple Salesforce organizations, regardless of their custom objects or fields. It is loosely typed, which means that calls operate on name-value pairs of field names and values instead of specific data types.

You must associate the enterprise WSDL with specific package versions to maintain existing behavior for clients. There are options for setting the package version bindings for an API call from client applications using either the enterprise or partner WSDL. The package version information for API calls issued from a client application based on the enterprise WSDL is determined by the first match in the following settings.

1. The PackageVersionHeader SOAP header.
2. The SOAP endpoint contains a URL with a format of `serverName/services/Soap/c/api_version/ID` where `api_version` is the version of the API, such as 60.0, and `ID` encodes your package version selections when the enterprise WSDL was generated.
3. The default enterprise package version settings.

The partner WSDL is more flexible as it's used for integration with multiple organizations. If you choose the Not Specified option for a package version when configuring the default partner package versions, the behavior is defined by the latest installed package version. This means that behavior of package components, such as an Apex trigger, could change when a package is upgraded and that change would immediately impact the integration. Subscribers may want to select a specific version for an installed package for all partner API calls from client applications to ensure that subsequent installations of package versions don't affect their existing integrations.

The package version information for partner API calls is determined by the first match in the following settings.

1. The PackageVersionHeader SOAP header.
2. An API call from a Visualforce page uses the package versions set for the Visualforce page.
3. The default partner package version settings.

To configure default package versions for API calls:

1. From Setup, enter `API` in the `Quick Find` box, then select **API**.
2. Click **Configure Enterprise Package Version Settings** or **Configure Partner Package Version Settings**. These links are only available if you have at least one managed package installed in your organization.
3. Select a `Package Version` for each of your installed managed packages. If you're unsure which package version to select, you should leave the default selection.
4. Click **Save**.

Note: Installing a new version of a package in your organization doesn't affect the current default settings.

About the Partner WSDL

The Partner Web Services WSDL is used for client applications that are metadata-driven and dynamic in nature. It's particularly—but not exclusively—useful to Salesforce partners who are building client applications for multiple organizations.

As a loosely typed representation of the Salesforce data model that works with name-value pairs of field names and values instead of specific data types, it can be used to access data within any organization. This WSDL is most appropriate for developers of clients that can issue a query call to get information about an object before the client acts on the object. The partner WSDL document needs to be downloaded and consumed only once per version of the API.

For more information about the Partner WSDL, see Using the Partner WSDL in *SOAP API Developer Guide*.

Generate an Enterprise WSDL with Managed Packages

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer**, Editions

USER PERMISSIONS

To download a WSDL:
- Customize Application

If you're downloading an enterprise WSDL and you have managed packages installed in your organization, you must take an extra step to select the version of each installed package to include in the generated WSDL.

The enterprise WSDL is strongly typed, which means that it contains objects and fields with specific data types, such as `int` and `string`.

A package version is a number that identifies the set of components uploaded in a package. The version number has the format *`majorNumber.minorNumber.patchNumber`* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *`patchNumber`* is generated and updated only for a patch release. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package. A subscriber can select a package version for each installed managed package to allow their API client to continue to function with specific, known behavior even when they install subsequent versions of a package. Each package version can have variations in the composition of its objects and fields, so you must select a specific version when you generate the strongly typed WSDL.

To download an enterprise WSDL when you have managed packages installed:

1. From Setup, enter *`API`* in the `Quick Find` box, then select **API**.
2. Click **Generate Enterprise WSDL**.
3. Select the `Package Version` for each of your installed managed packages. If you're unsure which package version to select, you should leave the default, which is the latest package version.
4. Click **Generate**.
5. Use the **File** menu in your browser to save the WSDL to your computer.
6. On your computer, import the local copy of the WSDL document into your development environment.

Note the following in your generated enterprise WSDL:

- Each of your managed package version selections is included in a comment at the top of the WSDL.
- The generated WSDL contains the objects and fields in your organization, including those available in the selected versions of each installed package. If a field or object is added in a later package version, you must generate the enterprise WSDL with that package version to work with the object or field in your API integration.
- The SOAP endpoint at the end of the WSDL contains a URL with a format of *`serverName`*`/services/Soap/c/`*`api_version`*`/`*`ID`* where *`api_version`* is the version of the API, such as 52.0, and *`ID`* encodes your package version selections when you communicate with Salesforce.

You can also select the default package versions for the enterprise WSDL without downloading a WSDL from the API page in Setup. Default package versions for API calls provide fallback settings if package versions aren't provided by an API call. Many API clients don't include package version information, so the default settings maintain existing behavior for these clients.

Work with Services Outside of Salesforce

You might want to update your Salesforce data when changes occur in another service. Likewise, you might also want to update the data in a service outside of Salesforce based on changes to your Salesforce data. For example, you might want to send a mass email to more contacts and leads than Salesforce allows. You can use an external mail service that allows users to build a recipient list of names and email addresses using the contact and lead information in your Salesforce organization.

An app built on the Salesforce Platform can connect with a service outside of Salesforce in many ways. For example, you can:

- create a custom link or custom formula field that passes information to an external service.
- use the Platform APIs to transfer data in and out of Salesforce.
- use an Apex class that contains a Web service method.

Warning: Don't store usernames and passwords within any external service.

Provisioning a Service External to Salesforce

If your app links to an external service, users who install the app must be signed up to use the service. Provide access in one of two ways:

- Access by all active users in an organization with no real need to identify an individual
- Access on a per user basis where identification of the individual is important

The Salesforce service provides two globally unique IDs to support these options. The user ID identifies an individual and is unique across all organizations. User IDs are never reused. Likewise, the organization ID uniquely identifies the organization.

Avoid using email addresses, company names, and Salesforce usernames when providing access to an external service. Usernames can change over time and email addresses and company names can be duplicated.

If you're providing access to an external service, we recommend the following:

- Use Single Sign-On (SSO) techniques to identify new users when they use your service.
- For each point of entry to your app, such as a custom link or web tab, include the user ID in the parameter string. Have your service examine the user ID to verify that the user ID belongs to a known user. Include a session ID in the parameter string so that your service can read back through the Lightning Platform API and validate that this user has an active session and is authenticated.
- Offer the external service for any known users. For new users, display an alternative page to collect the required information.
- Don't store passwords for individual users. Besides the obvious security risks, many organizations reset passwords on a regular basis, which requires the user to update the password on your system as well. We recommend designing your external service to use the user ID and session ID to authenticate and identify users.
- If your application requires asynchronous updates after a user session has expired, dedicate a distinct administrator user license for this.

Connected Apps

A connected app is a framework that enables an external application to integrate with Salesforce using APIs and standard protocols, such as SAML, OAuth, and OpenID Connect. Connected apps use these protocols to authenticate, authorize, and provide single sign-on (SSO) for external apps. The external apps that are integrated with Salesforce can run on the customer success platform, other platforms, devices, or SaaS subscriptions. For example, when you log in to your Salesforce mobile app and see your data from your Salesforce org, you're using a connected app.

By capturing metadata about an external app, a connected app tells Salesforce which authentication protocol—SAML, OAuth, and OpenID Connect—the external app uses, and where the external app runs. Salesforce can then grant the external app access to its data, and attach policies that define access restrictions, such as when the app's access expires. Salesforce can also audit connected app usage.

To learn more about how to use, configure, and manage connected apps, see the following topics in *Salesforce Help*:

- Connected App Use Cases
- Create a Connected App
- Edit a Connected App
- Manage Access to a Connected App

More Resources

Here are some additional resources to help you navigate connected apps:

- *Salesforce Help*: Connected Apps
- *Salesforce Help*: Authorize Apps with OAuth
- *Trailhead*: Build Integrations Using Connected Apps

Package and Test Your First-Generation Managed Package

Learn how to package, upload, and install a beta version of your first-generation managed package as part an iterative development approach. After your beta is up and running, learn how to test, fix, extend, and uninstall the package.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

Install a Managed Package

During the development and testing cycle, you might need to periodically install and uninstall packages before you install the next beta. Follow these steps to install a package.

Install First-Generation Managed Packages Using Metadata API

You can install, upgrade, and uninstall managed packages using the Metadata API, instead of the user interface. Automating these repeated tasks can help you can work more efficiently and speed up application development.

Component Availability After Deployment

Many components have an **Is Deployed** attribute that controls whether they're available for end users. After installation, all components are immediately available if they were available in the developer's organization.

Install Notifications for Unauthorized Managed Packages

When you distribute a managed package that AppExchange Partner Program hasn't authorized, we notify customers during the installation process. The notification is removed after the package is approved.

Resolve Apex Test Failures

Run Apex on Package Install/Upgrade

App developers can specify an Apex script to run automatically after a subscriber installs or upgrades a managed package. This script makes it possible to customize the package install or upgrade, based on details of the subscriber's organization. For instance, you can use the script to populate custom settings, create sample data, send an email to the installer, notify an external system, or kick off a batch operation to populate a new field across a large set of data. For simplicity, you can only specify one post install script. It must be an Apex class that is a member of the package.

Run Apex on Package Uninstall

App developers can specify an Apex script to run automatically after a subscriber uninstalls a managed package. This script makes it possible to perform cleanup and notification tasks based on details of the subscriber's organization. For simplicity, you can only specify one uninstall script. It must be an Apex class that is a member of the package.

Uninstall a Managed Package

Uninstalling a managed package removes its components and data from the org. During the uninstall process, any customizations, including custom fields or links, that you've made to the package are removed.

Install a Managed Package

During the development and testing cycle, you might need to periodically install and uninstall packages before you install the next beta. Follow these steps to install a package.

Pre-Installation

1. In a browser, type in the installation URL you received when you uploaded the package.
2. Enter your username and password for the Salesforce organization in which you want to install the package, and then click **Log In**.
3. If the package is password-protected, enter the password you received from the publisher.

Default Installation

Click **Install**. You'll see a message that describes the progress and a confirmation message after the installation is complete.

Custom Installation

Follow these steps if you need to modify the default settings, as an administrator.

1. Choose one or more of these options, as appropriate.
   - Click **View Components**. You see an overlay with a list of components in the package. For managed packages, the screen also contains a list of connected apps (trusted applications that are granted access to a user's Salesforce data after the user and the application are verified). To confirm that the components and any connected apps shown are acceptable, review the list and then close the overlay.

     Note: Some package items, such as validation rules, record types, or custom settings don't appear in the Package Components list but are included in the package and installed with the other items. If there are no items in the Package Components list, it's likely that the package contains only minor changes.

   - If the package contains a remote site setting, you must approve access to websites outside of Salesforce. The dialog box lists all the websites that the package communicates with. We recommend that a website uses SSL (secure sockets layer) for transmitting data. After you verify that the websites are safe, select **Yes, grant access to these third-party websites** and click **Continue**, or click **Cancel** to cancel the installation of the package.

     Warning: By installing remote site settings, you're allowing the package to transmit data to and from a third-party website. Before using the package, contact the publisher to understand what data is transmitted and how it's used. If you have an internal security contact, ask the contact to review the application so that you understand its impact before use.

   - Click **API Access**. You see an overlay with a list of the API access settings that package components have been granted. Review the settings to verify they're acceptable, and then close the overlay to return to the installer screen.
   - In Enterprise, Performance, Unlimited, and Developer Editions, choose one of the following security options.

     Note: This option is visible only in specific types of installations. For example, in Group and Professional Editions, or if the package doesn't contain a custom object, Salesforce skips this option, which gives all users full access.

     Install for Admins Only

     Specifies the following settings on the installing administrator's profile and any profile with the "Customize Application" permission.

     - Object permissions—`Read`, `Create`, `Edit`, `Delete`, `View All`, and `Modify All` enabled
     - Field-level security—set to visible and editable for all fields

- Apex classes—enabled
- Visualforce pages—enabled
- App settings—enabled
- Tab settings—determined by the package developer
- Page layout settings—determined by the package developer
- Record Type settings—determined by the package developer

After installation, if you have Enterprise, Performance, Unlimited, or Developer Edition, set the appropriate user and object permissions on custom profiles as needed.

Install for All Users

Specifies the following settings on all internal custom profiles.

- Object permissions— `Read`, `Create`, `Edit`, and `Delete` enabled
- Field-level security—set to visible and editable for all fields
- Apex classes—enabled
- Visualforce pages—enabled
- App settings—enabled
- Tab settings—determined by the package developer
- Page layout settings—determined by the package developer
- Record Type settings—copied from admin profile

Note: The Customer Portal User, Customer Portal Manager, High Volume Customer Portal, Authenticated Website, Partner User, and standard profiles receive no access.

Install for Specific Profiles...

Lets you determine package access for all custom profiles in your org. You can set each profile to have full access or no access for the new package and all its components.

- Full Access—Specifies the following settings for each profile.
  - Object permissions`Read`, `Create`, `Edit`, and `Delete` enabled
  - Field-level security—set to visible and editable for all fields
  - Apex classes—enabled
  - Visualforce pages—enabled
  - App settings—enabled
  - Tab settings—enabled
  - Page layout settings—determined by the package developer
  - Record Type settings—determined by the package developer
- No Access—Page layout and Record Type settings are determined by the package developer. All other settings are hidden or disabled.

If the package developer has included settings for custom profiles, you can incorporate the settings of the publisher's custom profiles into your profiles without affecting your settings. Choose the name of the profile settings in the dropdown list next to the profile that you're applying them to. The current settings in that profile remain intact.

Alternatively, click **Set All** next to an access level to give this setting to all user profiles.

2. Click **Install**. You'll see a message that describes the progress and a confirmation message after the installation is complete.

Post-Installation Steps

If the package includes post-installation instructions, they're displayed after the installation is completed. Review and follow the instructions provided. In addition, before you deploy the package to your users, make any necessary changes for your implementation. Depending on the contents of the package, some of the following customization steps are required.

- If the package includes permission sets, assign the included permission sets to your users who need them. In managed packages, you can't edit permission sets that are included in the package, but subsequent upgrades happen automatically. If you clone a permission set that comes with a managed package or create your own, you can edit the permission set, but subsequent upgrades won't affect it.
- If you're reinstalling a package and need to reimport the package data by using the export file that you received after uninstalling, see Import Package Data.
- If you installed a managed package, click **Manage Licenses** to assign licenses to users.

  Note: You can't assign licenses in Lightning Experience. To assign a license, switch to Salesforce Classic.

- Configure components in the package as required.

Install First-Generation Managed Packages Using Metadata API

You can install, upgrade, and uninstall managed packages using the Metadata API, instead of the user interface. Automating these repeated tasks can help you can work more efficiently and speed up application development.

To install, upgrade, or uninstall a package, use the standard Metadata API `deploy()` call with the `InstalledPackage` metadata type. The following operations are supported.

- Deploying an `InstalledPackage` installs the package in the deploying organization.
- Deploying a newer version of a currently installed package upgrades the package.
- Deploying an `InstalledPackage` using a manifest called `destructiveChanges.xml`, instead of `package.xml`, uninstalls it from the organization.

To specify whether all users, or only admins, can access the package you're installing, use the `securityType` field on the `InstalledPackage` metadata type. The default value is `AllUsers`. This field is available in API version 57.0 and later.

Note: `InstalledPackage` must be the only metadata type specified in the manifest file.

The following is a typical project manifest (`package.xml`) for installing a package. The manifest must not contain a `fullName` or `namespacePrefix` element.

```
<?xml version="1.0" encoding="UTF-8"?>
  <Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
      <members>*</members>
       <name>InstalledPackage</name>
     </types>
    <version>28.0</version>
  </Package>
```

The package is specified in a file called ***MyNamespace***`.installedPackage`, where ***MyNamespace*** is the namespace prefix of the package. The file must be in a directory called `installedPackages`, and its contents must have this format.

```
<?xml version="1.0" encoding="UTF-8"?>
  <InstalledPackage xmlns="http://soap.sforce.com/2006/04/metadata">
    <versionNumber>1.0</versionNumber>
    <password>optional_password</password>
    <securityType>AdminsOnly</securityType>
  </InstalledPackage>
```

The `securityType` field is optional. If it's not specified, the default security type is `AllUsers`.

InstalledPackage in API version 43.0 and later must include the `activateRSS` field set to either of these values.

true
: Keep the isActive state of any Remote Site Settings(RSS) or Content Security Policies(CSP) in the package.

false
: Override the isActive state of any RSS or CSP in the package and set it to `false`.

The default value is `false`.

Note: Regardless of what `activateRSS` is set to, a retrieve of `InstalledPackage` always returns `<activateRSS xsi:nil="true"/>`. Therefore, before you deploy a package, inspect the information you've retrieved from `InstalledPackage` and set `activateRSS` to the desired value.

To uninstall a package, deploy this `destructiveChanges.xml` manifest file in addition to the `package.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
  <Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
      <members>MyNamespace</members>
      <name>InstalledPackage</name>
     </types>
  </Package>
```

Retrieving an `InstalledPackage`, using the `retrieve()` call creates an XML representation of the package installed in an org. If the installed package has a password or security type specified, that information isn't retrieved. Deploying the retrieved file in a different org installs the package in that organization.

For more information on the `deploy()` and `retrieve()` commands, see the *Metadata API Developer's Guide*.

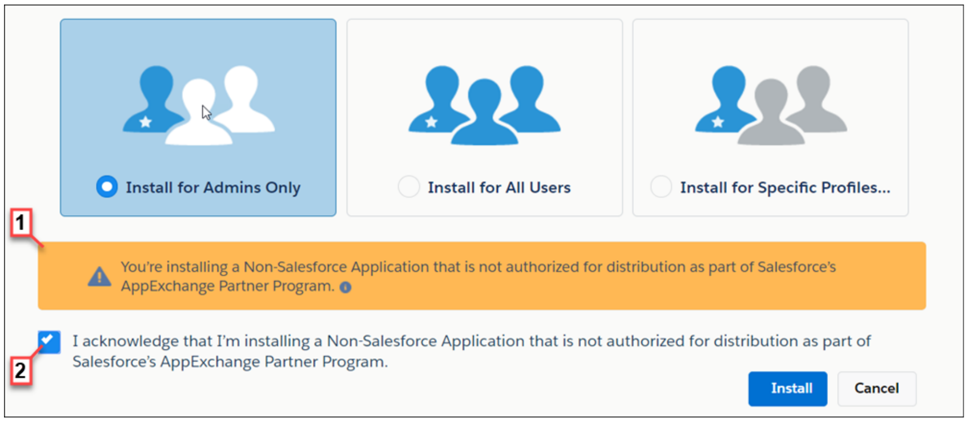
Component Availability After Deployment

Many components have an **Is Deployed** attribute that controls whether they're available for end users. After installation, all components are immediately available if they were available in the developer's organization.

Installed packages are available to users in your organization with the appropriate permissions and page layout settings.

Install Notifications for Unauthorized Managed Packages

When you distribute a managed package that AppExchange Partner Program hasn't authorized, we notify customers during the installation process. The notification is removed after the package is approved.



You're installing a Non-Salesforce Application that is not authorized for distribution as part of Salesforce's AppExchange Partner Program.

✓ I acknowledge that I'm installing a Non-Salesforce Application that is not authorized for distribution as part of Salesforce's AppExchange Partner Program.

The notification appears when customers configure the package installation settings (1). Before customers install the package, they must confirm that they understand that the package isn't authorized for distribution (2).

The notification displays when a managed package:

- Has never been through security review or is under review
- Didn't pass the security review
- Isn't authorized by AppExchange Partner Program for another reason

If the AppExchange Partner Program approves the package, it's authorized for distribution, and the notification is removed. When you publish a new version of the package, it's automatically authorized for distribution.

For information about the AppExchange Partner Program and its requirements, visit the Salesforce Partner Community.

Resolve Apex Test Failures

Package installs or upgrades may fail for not passing Apex test coverage. However, some of these failures can be ignored. For example, a developer might write an Apex test that makes assumptions about a subscriber's data.

If your install fails due to an Apex test failure, check for the following:

- Make sure that you're staging all necessary data required for your Apex test, instead of relying on subscriber data that exists.
- If a subscriber creates a validation rule, required field, or trigger on an object referenced by your package, your test might fail if it performs DML on this object. If this object is created only for testing purposes and never at runtime, and the creation fails due to these conflicts, you might be safe to ignore the error and continue the test. Otherwise, contact the customer and determine the impact.

Run Apex on Package Install/Upgrade

App developers can specify an Apex script to run automatically after a subscriber installs or upgrades a managed package. This script makes it possible to customize the package install or upgrade, based on details of the subscriber's organization. For instance, you can use the script to populate custom settings, create sample data, send an email to the installer, notify an external system, or kick off a batch operation to populate a new field across a large set of data. For simplicity, you can only specify one post install script. It must be an Apex class that is a member of the package.

The post install script is invoked after tests have been run, and is subject to default governor limits. It runs as a special system user that represents your package, so all operations performed by the script appear to be done by your package. You can access this user by using UserInfo. You can only see this user at runtime, not while running tests.

If the script fails, the install/upgrade is aborted. Any errors in the script are emailed to the user specified in the **Notify on Apex Error** field of the package. If no user is specified, the install/upgrade details are unavailable.

The post install script has the following additional properties.

- It can initiate batch, scheduled, and future jobs.
- It can't access Session IDs.
- It can only perform callouts using an async operation. The callout occurs after the script is run and the install is complete and committed.
- It can't call another Apex class in the package if that Apex class uses the `with sharing` keyword. This keyword can prevent the package from successfully installing. To learn more, see the *Apex Developer Guide*.

Note: You can't run a post install script in a new trial organization provisioned using Trialforce. The script only runs when a subscriber installs your package in an existing organization.

How Does a Post Install Script Work?
A post install script is an Apex class that implements the `InstallHandler` interface.

Example of a Post Install Script

Specifying a Post Install Script
After you've created and tested the post install script, you can specify it in the **Post Install Script** lookup field on the Package Detail page. In subsequent patch releases, you can change the contents of the script but not the Apex class.

How Does a Post Install Script Work?

A post install script is an Apex class that implements the `InstallHandler` interface.

This interface has a single method called `onInstall` that specifies the actions to be performed on installation.

```
global interface InstallHandler {
  void onInstall(InstallContext context)
}
```

The `onInstall` method takes a context object as its argument, which provides the following information.

- The org ID of the organization in which the installation takes place.
- The user ID of the user who initiated the installation.
- The version number of the previously installed package (specified using the `Version` class). This is always a three-part number, such as 1.2.0.
- Whether the installation is an upgrade
- Whether the installation is a push

The context argument is an object whose type is the `InstallContext` interface. This interface is automatically implemented by the system. The following definition of the `InstallContext` interface shows the methods you can call on the context argument.

```
global interface InstallContext {
  ID organizationId();
  ID installerId();
  Boolean isUpgrade();
  Boolean isPush();
  Version previousVersion();
}
```

Version Methods and Class

You can use the methods in the `System.Version` class to get the version of a managed package and to compare package versions. A package version is a number that identifies the set of components in a package. The version number has the format *`majorNumber.minorNumber.patchNumber`* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every non-patch release. Major and minor number increases always use a patch number of 0.

The following are instance methods for the `System.Version` class.

Method	Arguments	Return Type	Description
`compareTo`	System.Version *`version`*	Integer	Compares the current version with the specified version and returns one of the following values: • Zero if the current package version is equal to the specified package version • An Integer value greater than zero if the current package version is greater than the specified package version • An Integer value less than zero if the current package version is less than the specified package version If a two-part version is being compared to a three-part version, the patch number is ignored and the comparison is based only on the major and minor numbers.
`major`		Integer	Returns the major package version of the calling code.
`minor`		Integer	Returns the minor package version of the calling code.
`patch`		Integer	Returns the patch package version of the calling code or `null` if there's no patch version.

The `System` class contains two methods that you can use to specify conditional logic, so different package versions exhibit different behavior.

- `System.requestVersion`: Returns a two-part version that contains the major and minor version numbers of a package.Using this method, you can determine the version of an installed instance of your package from which the calling code is referencing your package. Based on the version that the calling code has, you can customize the behavior of your package code.
- `System.runAs(System.Version)`: Changes the current package version to the package version specified in the argument.

When a subscriber has installed multiple versions of your package and writes code that references Apex classes or triggers in your package, they must select the version they're referencing. You can execute different code paths in your package's Apex code based on the version setting of the calling Apex code making the reference. You can determine the calling code's package version setting by calling the `System.requestVersion` method in the package code.

Example of a Post Install Script

The following sample post install script performs these actions on package install/upgrade.

- If the previous version is null, that is, the package is being installed for the first time, the script:
  - Creates a new Account called Newco and verifies that it was created.
  - Creates a new instance of the custom object Survey, called Client Satisfaction Survey.
  - Sends an email message to the subscriber confirming installation of the package.
- If the previous version is 1.0, the script creates a new instance of Survey called "Upgrading from Version 1.0".
- If the package is an upgrade, the script creates a new instance of Survey called "Sample Survey during Upgrade".
- If the upgrade is being pushed, the script creates a new instance of Survey called "Sample Survey during Push".

```
public class PostInstallClass implements InstallHandler {
  global void onInstall(InstallContext context) {
    if(context.previousVersion() == null) {
      Account a = new Account(name='Newco');
      insert(a);

      Survey__c obj = new Survey__c(name='Client Satisfaction Survey');
      insert obj;

      User u = [Select Id, Email from User where Id =:context.installerID()];
      String toAddress= u.Email;
      String[] toAddresses = new String[]{toAddress};
      Messaging.SingleEmailMessage mail =
        new Messaging.SingleEmailMessage();
      mail.setToAddresses(toAddresses);
      mail.setReplyTo('support@package.dev');
      mail.setSenderDisplayName('My Package Support');
      mail.setSubject('Package install successful');
      mail.setPlainTextBody('Thanks for installing the package.');
      Messaging.sendEmail(new Messaging.Email[] { mail });
      }
    else
      if(context.previousVersion().compareTo(new Version(1,0)) == 0) {
      Survey__c obj = new Survey__c(name='Upgrading from Version 1.0');
      insert(obj);
      }
    if(context.isUpgrade()) {
      Survey__c obj = new Survey__c(name='Sample Survey during Upgrade');
      insert obj;
      }
    if(context.isPush()) {
      Survey__c obj = new Survey__c(name='Sample Survey during Push');
      insert obj;
      }
    }
  }
```

You can test a post install script using the new `testInstall` method of the `Test` class. This method takes the following arguments.

- A class that implements the `InstallHandler` interface.
- A `Version` object that specifies the version number of the existing package.
- An optional Boolean value that is `true` if the installation is a push. The default is `false`.

This sample shows how to test a post install script implemented in the `PostInstallClass` Apex class.

```
@isTest
static void testInstallScript() {
  PostInstallClass postinstall = new PostInstallClass();
    Test.testInstall(postinstall, null);
    Test.testInstall(postinstall, new Version(1,0), true);
    List<Account> a = [Select id, name from Account where name ='Newco'];
    System.assertEquals(1, a.size(), 'Account not found');
  }
```

Specifying a Post Install Script

After you've created and tested the post install script, you can specify it in the **Post Install Script** lookup field on the Package Detail page. In subsequent patch releases, you can change the contents of the script but not the Apex class.

The class selection is also available via the Metadata API as `Package.postInstallClass`. This is represented in package.xml as a `<postInstallClass>foo</postInstallClass>` element.

Run Apex on Package Uninstall

App developers can specify an Apex script to run automatically after a subscriber uninstalls a managed package. This script makes it possible to perform cleanup and notification tasks based on details of the subscriber's organization. For simplicity, you can only specify one uninstall script. It must be an Apex class that is a member of the package.

The uninstall script is subject to default governor limits. It runs as a special system user that represents your package, so all operations performed by the script appear to be done by your package. You can access this user by using UserInfo. You can only see this user at runtime, not while running tests.

If the script fails, the uninstall continues but none of the changes performed by the script are committed. Any errors in the script are emailed to the user specified in the **Notify on Apex Error** field of the package. If no user is specified, the uninstall details are unavailable.

The uninstall script has the following restrictions. You can't use it to initiate batch, scheduled, and future jobs, to access Session IDs, or to perform callouts.

How Does an Uninstall Script Work?

An uninstall script is an Apex class that implements the `UninstallHandler` interface. This interface has a single method called `onUninstall` that specifies the actions to be performed on uninstall.

Example of an Uninstall Script

This sample uninstall script performs the following actions on package uninstall.

Specifying an Uninstall Script

After you've created and tested the uninstall script and included it as a member of your package, you can specify it in the **Uninstall Script** lookup field on the Package Detail page.

How Does an Uninstall Script Work?

An uninstall script is an Apex class that implements the `UninstallHandler` interface. This interface has a single method called `onUninstall` that specifies the actions to be performed on uninstall.

```
global interface UninstallHandler {
  void onUninstall(UninstallContext context)
}
```

The `onUninstall` method takes a context object as its argument, which provides the following information.

- The org ID of the organization in which the uninstall takes place.
- The user ID of the user who initiated the uninstall.

The context argument is an object whose type is the `UninstallContext` interface. This interface is automatically implemented by the system. The following definition of the `UninstallContext` interface shows the methods you can call on the context argument.

```
global interface UninstallContext {
  ID organizationId();
  ID uninstallerId();
}
```

Example of an Uninstall Script

This sample uninstall script performs the following actions on package uninstall.

- Inserts an entry in the feed describing which user did the uninstall and in which organization
- Creates and sends an email message confirming the uninstall to that user

```
global class UninstallClass implements UninstallHandler {
  global void onUninstall(UninstallContext ctx) {
    FeedItem feedPost = new FeedItem();
    feedPost.parentId = ctx.uninstallerID();
    feedPost.body = 'Thank you for using our application!';
    insert feedPost;

    User u = [Select Id, Email from User where Id =:ctx.uninstallerID()];
    String toAddress= u.Email;
    String[] toAddresses = new String[] {toAddress};
    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
    mail.setToAddresses(toAddresses);
    mail.setReplyTo('support@package.dev');
    mail.setSenderDisplayName('My Package Support');
    mail.setSubject('Package uninstall successful');
    mail.setPlainTextBody('Thanks for uninstalling the package.');
    Messaging.sendEmail(new Messaging.Email[] { mail });
  }
}
```

You can test an uninstall script using the `testUninstall` method of the `Test` class. This method takes as its argument a class that implements the `UninstallHandler` interface.

This sample shows how to test an uninstall script implemented in the `UninstallClass` Apex class.

```
@isTest
static void testUninstallScript() {
  Id UninstallerId = UserInfo.getUserId();
  List<FeedItem> feedPostsBefore =
    [SELECT Id FROM FeedItem WHERE parentId=:UninstallerId AND CreatedDate=TODAY];
  Test.testUninstall(new UninstallClass());
  List<FeedItem> feedPostsAfter =
    [SELECT Id FROM FeedItem WHERE parentId=:UninstallerId AND CreatedDate=TODAY];
  System.assertEquals(feedPostsBefore.size() + 1, feedPostsAfter.size(),
    'Post to uninstaller failed.');
}
```

Specifying an Uninstall Script

After you've created and tested the uninstall script and included it as a member of your package, you can specify it in the **Uninstall Script** lookup field on the Package Detail page.

In subsequent patch releases, you can change the contents of the script but not the Apex class.

The class selection is also available via the Metadata API as `Package.uninstallClass`. This is represented in package.xml as an `<uninstallClass>foo</uninstallClass>` element.

Uninstall a Managed Package

Uninstalling a managed package removes its components and data from the org. During the uninstall process, any customizations, including custom fields or links, that you've made to the package are removed.

1. From Setup, enter *`Installed Packages`* in the Quick Find box, then select **Installed Packages**.
2. Click **Uninstall** next to the package that you want to remove.
3. Determine whether to save and export a copy of the package's data, and then select the corresponding radio button.
4. Select **Yes, I want to uninstall** and click **Uninstall**.

When you uninstall packages, consider the following:

- If you're uninstalling a package that includes a custom object, all components on that custom object are also deleted. Deleted items include custom fields, validation rules, custom buttons, and links, workflow rules, and approval processes.
- You can't uninstall a package whenever a component not included in the uninstall references any component in the package. For example:
  - When an installed package includes any component on a standard object that another component references, Salesforce prevents you from uninstalling the package. An example is a package that includes a custom user field with a workflow rule that gets triggered when the value of that field is a specific value. Uninstalling the package would prevent your workflow from working.
  - When you've installed two unrelated packages that each include a custom object and one custom object component references a component in the other, you can't uninstall the package. An example is if you install an expense report app that includes a custom user field and create a validation rule on another installed custom object that references that custom user field. However, uninstalling the expense report app prevents the validation rule from working.
  - When an installed folder contains components you added after installation, Salesforce prevents you from uninstalling the package.
  - When an installed letterhead is used for an email template you added after installation, Salesforce prevents you from uninstalling the package.

  - When an installed package includes a custom field that's referenced by Einstein Prediction Builder or Case Classification, Salesforce prevents you from uninstalling the package. Before uninstalling the package, edit the prediction in Prediction Builder or Case Classification so that it no longer references the custom field.

- You can't uninstall a package that removes all active business and person account record types. Activate at least one other business or person account record type, and try again.
- You can't uninstall a package if a background job is updating a field added by the package, such as an update to a roll-up summary field. Wait until the background job finishes, and try again.

Update Your First-Generation Managed Package

Your app is ready for an update. Learn how to fix small issues with patches and make major changes with upgrades.

Package Versions in First-Generation Managed Packages
A package version is a number that identifies the set of components uploaded in a package. The version number has the format `majorNumber.minorNumber.patchNumber` (for example, 2.1.3).

Create and Upload Patches in First-Generation Managed Packages
Patch versions are developed and maintained in a patch development org.

Work with Patch Versions
A *patch version* enables a developer to change the functionality of existing components in a managed package. Subscribers experience no visible changes to the package. Patches are minor upgrades to a Managed - Released package and only used for fixing bugs or other errors.

Versioning Apex Code
When subscribers install multiple versions of your package and write code that references Apex classes or triggers in your package, they must specify the version that they're referencing.

Apex Deprecation Effects for Subscribers
Explore how deprecation of an Apex method impacts subscribers that install your managed package.

Package Versions in First-Generation Managed Packages

A package version is a number that identifies the set of components uploaded in a package. The version number has the format `majorNumber.minorNumber.patchNumber` (for example, 2.1.3).

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

Version numbers depend on the package release type, which identifies the way packages are distributed. There are two kinds:

Major Release
A major release denotes a Managed - Released package. During these releases, the major and minor numbers of a package version increase to a chosen value.

Patch Release
A patch release is only for patch versions of a package. During these releases, the patch number of a package version increments.

The following table shows a sequence of version numbers for a series of uploads:

Upload Sequence	Type	Version Number	Notes
First upload	Managed - Beta	1.0	The firstManaged - Beta upload.
Second upload	Managed - Released	1.0	A Managed - Released upload. The version number doesn't change.
Third upload	Managed - Released	1.1	Note the change of the minor release number for this Managed - Released upload. If you're uploading a new patch version, you can't change the patch number.
Fourth upload	Managed - Beta	2.0	The first> Managed - Beta upload for version number 2.0. Note the major version number update.
Fifth upload	Managed - Released	2.0	A Managed - Released upload. The version number doesn't change.

When an existing subscriber installs a new package version, there's still only one instance of each component in the package, but the components can emulate older versions. For example, a subscriber can use a managed package that contains an Apex class. If the publisher decides to deprecate a method in the Apex class and release a new package version, the subscriber still sees only one instance of the Apex class after installing the new version. However, this Apex class can still emulate the previous version for any code that references the deprecated method in the older version.

Package developers can use conditional logic in Apex classes and triggers to exhibit different behavior for different versions. Conditional logic lets the package developer support existing behavior in classes and triggers in previous package versions while evolving the code.

When you're developing client applications using the API, you can specify the version of each package that you use in your integrations.

Create and Upload Patches in First-Generation Managed Packages

Patch versions are developed and maintained in a patch development org.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

To create a patch version:

1. From Setup, enter `Packages` in the Quick Find box, then select **Packages**.
2. Click the name of your managed package.
3. On the Patch Organization tab, click **New**.
4. Select the package version that you want to create a patch for in the Patching Major Release dropdown. The release type must be Managed - Released.
5. Enter a username for a login to your patch org.
6. Enter an email address associated with your login.
7. Click **Save**.

EDITIONS

Available in: **Developer** Edition

USER PERMISSIONS

To push an upgrade or create a patch development org

- Upload AppExchange Packages

Note: If you ever lose your login information, click **Reset** on the package detail page under Patch Development Organizations to reset the login to your patch development org.

The name of the patch development org's My Domain name is randomly generated.

After you receive an email that Salesforce has created your patch development org, you can click **Login** to begin developing your patch version.

Development in a patch development org is restricted.

- You can't add package components.
- You can't delete existing package components.
- API and dynamic Apex access controls can't change for the package.
- No deprecation of any Apex code.
- You can't add new Apex class relationships, such as `extends`.
- You can't add Apex access modifiers, such as `virtual` or `global`.
- You can't add new web services.
- You can't add feature or component dependencies.

  You can remove a feature or component dependency from a patch, but after the dependency is removed and the patch version is uploaded, you can't reinstate that dependency in a new patch version. To reinstate the dependency, create a new major or minor package version.

When you finish developing your patch, upload it through the UI in your patch development org. You can also upload a package using the Tooling API. For sample code and more details, see the PackageUploadRequest object in the *Tooling API Developer Guide*.

Note: When you upload a new package in your patch development org, the upload process is asynchronous. Because the time to process the request varies, the package isn't available immediately after the upload. While waiting, you can run SOQL queries on the PackageUploadRequest status field to monitor the request.

1. From Setup, enter *`Packages`* in the Quick Find box, then select **Packages**.
2. Click the name of the package.
3. On the Upload Package page, click **Upload**.
4. Enter a `Version Name`. As a best practice, it's useful to have a short description and the date.
5. Notice that the `Version Number` has had its `patchNumber` incremented.
6. For managed packages, select a `Release Type`:
   - Choose Managed - Released to upload an upgradeable version. After upload, some attributes of the metadata components are locked.
   - Choose Managed - Beta if you want to upload a version of your package to a small sampling of your audience for testing purposes. You can still change the components and upload other beta versions.

     Note: Beta packages can only be installed in Developer Edition,scratch, or sandbox orgs, and thus can't be pushed to customer orgs.
7. Change the `Description`, if necessary.
8. (Optional) Enter and confirm a password to share the package privately with anyone who has the password. Don't enter a password if you want to make the package available to anyone on AppExchange and share your package publicly.
9. Salesforce automatically selects the requirements it finds. In addition, select any other required components from the `Package Requirements` and `Object Requirements` sections to notify installers of any requirements for this package.

10. Click **Upload**.

To distribute your patch, you can either share the upload link or schedule a push upgrade.

Work with Patch Versions

A *patch version* enables a developer to change the functionality of existing components in a managed package. Subscribers experience no visible changes to the package. Patches are minor upgrades to a ✓ Managed - Released package and only used for fixing bugs or other errors.

Patch versions can only be created for Major Releases. Subscribers can receive patch upgrades just like any other package version. However, you can also distribute a patch by using push upgrades.

When you create a patch, the `patchNumber` on a package's `Version Number` increments by one. For example, suppose that you release a package with the version number 2.0. When you release a patch, the number changes to 2.0.1. This value can't be changed manually.

Patch Development Organizations

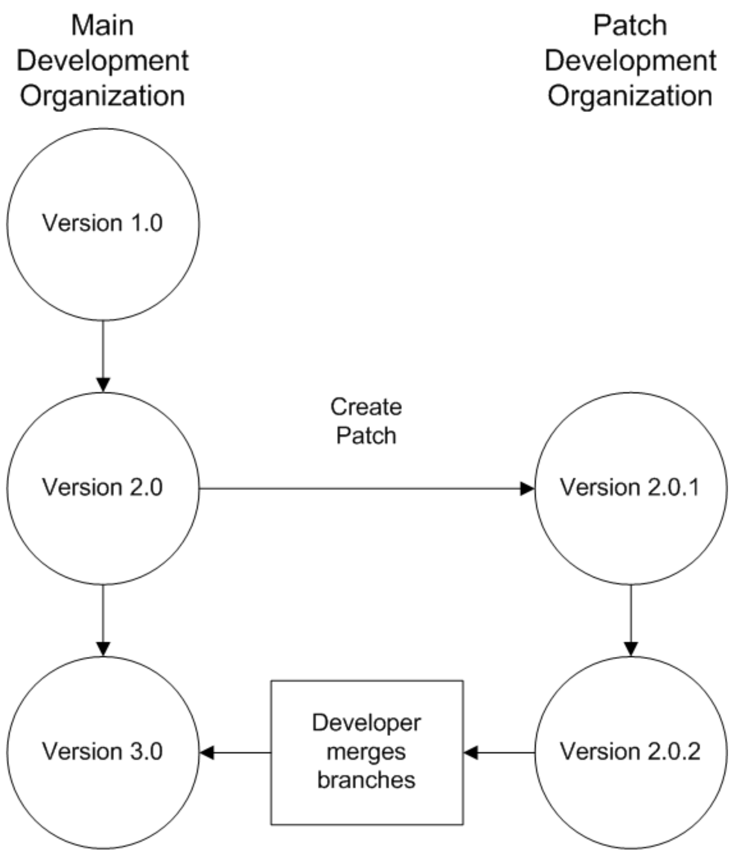
Every patch is developed in a *package version*, which is the organization where patch versions are developed, maintained, and uploaded. To start developing a patch, create a package version. See Create and Upload Patches in First-Generation Managed Packages on page 72. Patch development organizations are necessary to permit developers to change existing components without causing incompatibilities between existing subscriber installations.

A package version can upload an unlimited number of patches. Only one package version can exist per major release of your package. A package version for a package with a version number of 4.2 can only work on patches such as 4.2.1, 4.2.2, 4.2.3, and so on. It doesn’t work on version 4.1 or 4.3.

Integrating Patch Development

The following diagram illustrates the workflow of creating a patch and integrating any work into future versions:After version 2.0 is released, the developer creates a patch. The package version number in the package version starts at 2.0.1. As the main development organization moves towards a released version of 3.0, a second patch is created for 2.0.2. Finally, the developer merges the changes between the main development organization, and the package version, and releases the package as version 3.0.

Patch Development Workflow


Main Development Organization
Patch Development Organization
Version 1.0
Version 2.0
Create Patch
Version 2.0.1
Version 3.0
Developer merges branches
Version 2.0.2

Git source control is the best way to monitor your package versions. To learn about Git, complete the Git and GitHub Basics Trailhead module.

Version control is integrated into Visual Studio Code. See Salesforce Extensions for Visual Studio Code and Version Control in Visual Studio Code for details.

Versioning Apex Code

When subscribers install multiple versions of your package and write code that references Apex classes or triggers in your package, they must specify the version that they're referencing.

Within the Apex code that is being referenced in your package, you can conditionally execute different code paths based on the version setting of the calling Apex code that is making the reference. The package version setting of the calling code can be determined within the package code by calling the `System.requestVersion` method. In this way, package developers can determine the request context and specify different behavior for different versions of the package.

The following sample shows different behavior in a trigger for different package versions:

```
trigger oppValidation on Opportunity (before insert, before update) {

    for (Opportunity o : Trigger.new){

        // Add a new validation to the package
        // Applies to versions of the managed package greater than 1.0
        if (System.requestVersion().compareTo(new Version(1,0)) > 0) {
            if (o.Probability >= 50 && o.Description == null) {
                o.addError('All deals over 50% require a description');
            }
        }

        // Validation applies to all versions of the managed package.
        if (o.IsWon == true && o.LeadSource == null) {
            o.addError('A lead source must be provided for all Closed Won deals');
        }
    }
}
```

To compare different versions of your Apex classes, click the **Class Definition** tab when viewing the class details.

For more information about the `System.requestVersion` method, see the *Apex Developer Guide*.

Apex Deprecation Effects for Subscribers

Explore how deprecation of an Apex method impacts subscribers that install your managed package.

The table shows a typical sequence of actions by a package developer in the first column and actions by a subscriber in the second column. Each row in the table denotes either a package developer or subscriber action.

Package Developer Action	Subscriber Action	Notes
Create a global Apex class, `PackageDevClass`, containing a global method `m1`.		
Upload as Managed - Released version 1.0 of a package that contains `PackageDevClass`.		
	Install version 1.0 of the package.	The `Version Number` for the package is 1.0. The `First Installed Version Number` is 1.0.
	Create an Apex class, `SubscriberClass`, that references `m1` in `PackageDevClass`.	
Deprecate `m1` and create a new method, `m2`.		
Upload as Managed - Released version 2.0 of the package.		
	Install version 2.0 of the package.	The `Version Number` for the package is 2.0. The `First Installed Version Number` is still 1.0. `SubscriberClass` still references version 1.0 of the package and continues to function, as before.

Package Developer Action	Subscriber Action	Notes
	Edit the version settings for `SubscriberClass` to reference version 2.0 of the package. Save the class. Note an error message indicating that `m1` can't be referenced in version 2.0 of the package.	
	Change `SubscriberClass` to reference `m2` instead of `m1`. Successfully save the class.	

Publish Upgrades to First-Generation Managed Packages

USER PERMISSIONS

To configure namespace settings:
- Customize Application

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages

As a publisher, first ensure that your app is upgradeable by converting it to a managed package.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

Any changes you make to the components in a managed package are automatically included in subsequent uploads of that package, with one exception. When you upgrade a package, changes to the API access are ignored even if the developer specified them. These changes are ignored so that the administrator installing the upgrade has full control. Installers must carefully examine the changes in package access in each upgrade during installation and note all acceptable changes. Then, because those changes are ignored, the admin must manually apply any acceptable changes after installing an upgrade.

1. From Setup, enter *`Package Manager`* in the `Quick Find` box, then select **Package Manager**.
2. Select the package from the list of available packages.
3. View the list of package components. Changes you have made to components in this package are automatically included in this list. If the changes reference additional components, those components are automatically included as well. To add new components, click **Add** to add them to the package manually.
4. Click **Upload** and upload it as usual.

   After you upload a new version of your Managed - Released package, you can click **Deprecate** so installers can't install an older version. Deprecation prevents new installations of older versions without affecting existing installations.

   You can't deprecate the most recent version of a managed package upload.

5. When you receive an email with the link to the upload on AppExchange, notify your installed users that the new version is ready. To distribute this information, use the list of installed users from the License Management Application (LMA). The License Management Application (LMA) automatically stores the version number that your installers have in their organizations.

Plan the Release of First-Generation Managed Packages
Releasing a managed package is similar to releasing any other program in software development.

Remove Components from First-Generation Managed Packages
Remove metadata components such as Apex classes that you no longer want in your first-generation managed packages.

Delete Components from First-Generation Managed Packages
After you've uploaded a Managed - Released first-generation managed package, you may find that a component needs to be deleted from your packaging org.

Modifying Custom Fields after a Package Is Released
The following changes are allowed to custom fields in a package, after it's released.

Manage Versions of First-Generation Managed Packages
After you upload a package to AppExchange, you can still manage it from the Package Manager page.

View Unused Components in a Managed Package
View components no longer being used in the current version of a package.

Push Package Upgrades to Subscribers
A push upgrade is a method of automatically upgrading your customers to a newer version of your package. This feature can be used to ensure that all your customers are on the same or latest version of your package. You can push an upgrade to any number of organizations that have installed your managed package.

Plan the Release of First-Generation Managed Packages

Releasing a managed package is similar to releasing any other program in software development.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

After you release a package by publishing it on AppExchange, anyone can install it. So, plan your release carefully. Review the states defined in the following table to familiarize yourself with the release process. Salesforce automatically applies the appropriate state to your package and components depending on the upload settings you choose and where it is in the release process.

State	Description
Managed - Beta	The package or component was created in the current Salesforce org and is managed, but it isn't released because of one of these reasons: • It hasn't been uploaded. • It has been uploaded with `Managed – Beta` option selected. This option prevents it from being published, publicly available on AppExchange. The developer can still edit any component but the installer isn't able to depending on which components were packaged. Don't install a Managed - Beta package over a Managed - Released package. If you do, the package is no longer upgradeable and your only option is to uninstall and reinstall it.
Managed - Released	The package or component was created in the current Salesforce org and is managed. It's also uploaded with the `Managed – Released` option selected, indicating that it can be published on AppExchange and is publicly available. After you've moved a package to this state, some properties of the components can't be editable. This type of release is considered a major release.

State	Description
Patch	If you must provide a minor upgrade to a managed package, consider creating a patch instead of a new major release. A patch enables a developer to change the functionality of existing components in a managed package. Subscribers experience no visible changes to the package. This type of release is considered a patch release.
Managed - Installed	The package or component was installed from another Salesforce org but is managed.
Unmanaged (Legacy)	The package hasn't been converted into a managed package.

A developer can refine the functionality in a managed package over time, uploading and releasing new versions as the requirements evolve. Updates can involve redesigning some of the components in the managed package. Developers can delete some, but not all, types of components in a Managed - Released package when upgrading it.

Remove Components from First-Generation Managed Packages

Remove metadata components such as Apex classes that you no longer want in your first-generation managed packages.

After a managed package has been promoted to the Managed-Released state, only certain components can be removed. To confirm whether a specific component can be removed, see Components Available in Managed Packages in the Second-Generation Managed Packaging Developer Guide.

Impact of Component Removal in Subscriber Orgs

During package upgrade only certain component types are hard deleted and removed from the subscriber org. Most metadata components that were removed in a package version, will remain in the subscriber org after package upgrade, and marked as deprecated. When a package is upgraded in the subscriber org, the Setup Audit Trail logs which components were removed. Admins of a subscriber org can delete deprecated metadata.

To enable component deletion in your packaging org, log a support case in the Salesforce Partner Community.

Before you remove a component, ensure that there aren't any dependencies on the metadata you plan to remove. If any component in the package depends on or references the component you're removing, the package version creation operation fails. After you remove a component or field, you can't access the component, or any customizations that depend on the removed component.

When you delete a component, you also permanently delete the data that exists in that component. Delete tracked history data is also deleted, and integrations that rely on the component, such as assignment or escalation rules, are changed. After you delete a component in a managed package, you can't restore it or create another component with the same name.

Note: In a managed package, the API names of fields must be unique and can't be reused even after you delete the component. This restriction prevents conflicts during package installation and upgrade.

Data and metadata are never deleted in a subscriber org without specific action by the customer. When a subscriber upgrades to the new package version, the deleted components are still available in the subscriber's org. The components are displayed in the Unused Components section of the Package Details page. This section ensures that subscribers have the opportunity to export data and modify custom integrations involving those components before explicitly deleting them. For example, before deleting custom objects or fields, customers can preserve a record of their data from Setup by entering *Data Export* in the Quick Find box and then selecting **Data Export**.

Note: Educate your customers about the potential impact of deleted components. Consider listing all custom components that you've deleted and specifying any actions needed in the Release Notes for your upgraded package.

These restrictions apply when deleting managed components.

- If a component is referenced by any other metadata, such as workflow rules, validation rules, or Apex classes, you can't delete it.
- You can't delete a custom object if it includes Apex Sharing Reason, Apex Sharing Recalculation, Related Lookup Filter, Compact Layout, or Action.
- Salesforce doesn't recommend deleting a custom field that is referenced by a custom report type in the same package. This type of deletion causes an error when installing the upgraded package.
- When you delete a field that is used for bucketing or grouping in a custom report type that is part of a managed package, you receive an error message.
- When you remove a connected app that is a component of a package, the app remains available until you update the package with a new version. But if you delete the connected app, it's permanently deleted. Any version of the package that contains the deleted connected app is invalidated and can't be installed. You can update a version of the package that *doesn't* contain the connected app as a component. Never delete a connected app that Salesforce distributes, such as the Salesforce app.

You can delete managed components either declaratively from the user interface or programmatically using Metadata API. With Metadata API, specify the components that you want to delete in a `destructiveChanges.xml` manifest file and then use the standard `deploy()` call. The process is identical to deleting components that aren't managed. For more information, see the Metadata API Developer Guide .

Removing Public Apex Classes and Public Visualforce Components

Because the behavior of managed package components differs from public Apex classes and public Visualforce components, you use a two-stage process to delete Visualforce pages, global Visualforce components, and global Lightning components from a managed package. When you upgrade a package in a subscriber org, the Visualforce pages, global Visualforce components, and Lightning components that you deleted aren't removed. Although a Delete button or link is available to org administrators, many orgs continue using the obsolete pages and components. However, public Apex classes and public Visualforce components are deleted as part of the upgrade process. If you delete pages and components without performing this two-stage procedure, Salesforce can't warn you when later deletions of public classes and components break your subscribers' obsolete pages and components.

If you're deleting these types of components, perform this two-stage process in the order presented.

- - A Visualforce page or global Visualforce component that refers to or uses public Apex classes or public Visualforce components
  - An Aura component with global access
  - A Lightning web component with an `isExposed` value of `true`

  1. **Stage one:** Remove references.
     - **i.** Edit the global component that you want to delete.
       - For Visualforce, edit your Visualforce page or global Visualforce component to remove all references to public Apex classes or public Visualforce components.
       - For Lightning components, edit the global Lightning component to remove all references to other Lightning components.
     - **ii.** Upload your new package version.
     - **iii.** Push the stage-one upgrade to your subscribers.
  2. Stage two: Delete your obsolete pages or components.
     - **i.** Delete your Visualforce page, global Visualforce component, or global Lightning component.
     - **ii.** Optionally, delete other related components and classes.
     - **iii.** Upload your new package version.
     - **iv.** Push the stage-two upgrade to your subscribers.

Delete Components from First-Generation Managed Packages

After you've uploaded a Managed - Released first-generation managed package, you may find that a component needs to be deleted from your packaging org.

One of the following situations may occur:

- The component, after it's added to a package, can't be deleted.
- The component can be deleted, but can only be undeleted from the Deleted Package Components page.
- The component can be deleted, but can be undeleted from either the Deleted Package Components page or through the Recycle Bin

After a package is uploaded with a component marked for deletion, the component is deleted forever.

Warning: When a component is deleted, its **Name** remains within Salesforce, and you can't create a new component that uses the deleted component's name. The Deleted Package Components page lists the names that can no longer be used.

To access the Deleted Package Components page, from Setup, enter `Package Manager` in the `Quick Find` box, then select **Package Manager**. Select the package that the component was uploaded to, and then click **View Deleted Components**. You can retrieve components from the Recycle Bin and Deleted Package Components page any time *before* uploading a new version of the package. To do this, click **Undelete** next to the component.

You can retrieve these types of components.

- Apex classes and triggers that don't have `global` access.
- Visualforce components with `public` access. (If the ability to remove components has been enabled for your packaging org then these Visualforce components can't be undeleted. As a result, they don't show up in the Recycle Bin or the Deleted Package Components page after they've been deleted.)
- Protected components, including:
  - Custom labels
  - Custom links (for Home page only)
  - Custom metadata types
  - Custom permissions
  - Custom settings
  - Workflow alerts
  - Workflow field updates
  - Workflow outbound messages
  - Workflow tasks
  - Workflow flow triggers

    The pilot program for flow trigger workflow actions is closed. If you've already enabled the pilot in your org, you can continue to create and edit flow trigger workflow actions. If you didn't enable the pilot in your org, use Flow Builder to create a record-triggered flow, or use Process Builder to launch a flow from a process.

- Data components, such as Documents, Dashboards, and Reports. These components are the only types that can also be undeleted from the Recycle Bin.

You can retrieve components from the Recycle Bin and Deleted Package Components page any time *before* uploading a new version of the package. To do this, click **Undelete** next to the component.

The Deleted Components displays the following information (in alphabetical order):

Attribute	Description
`Action`	If the ✓ Managed - Released package hasn't been uploaded with the component deleted, this contains an **Undelete** link that allows you to retrieve the component.
`Available in Versions`	Displays the version number of the package in which a component exists.
`Name`	Displays the name of the component.
`Parent Object`	Displays the name of the parent object a component is associated with. For example, a custom object is the parent of a custom field.
`Type`	Displays the type of the component.

Modifying Custom Fields after a Package Is Released

EDITIONS

Available in: **Developer** Edition

The following changes are allowed to custom fields in a package, after it's released.

- The length of a text field can be increased or decreased.
- The number of digits to the left or right of the decimal point in a number field can be increased or decreased.
- A required field can be made non-required and vice versa. If a default value was required for a field, that restriction can be removed and vice versa.

Manage Versions of First-Generation Managed Packages

USER PERMISSIONS

To upload packages:
- Upload AppExchange Packages

After you upload a package to AppExchange, you can still manage it from the Package Manager page.

Note: Building a new app? Have you considered using second-generation managed packages? Flexible versioning and the ability to share a namespace across packages are just two reasons why developers love creating second-generation managed packages. We think you'd love it, too. To learn more, see: Why Switch to Second-Generation Managed Packages, and Comparison of First- and Second-Generation Managed Packages.

To manage your versions:

1. From Setup, enter *`Packages`* in the `Quick Find` box, then select **Packages**.
2. Select the package that contains the app or components you uploaded.
3. Select the version number listed in the Versions tab.
   - To change the password option, click **Change Password** link.
   - To prevent new installations of this package while allowing existing installations to continue operating, click **Deprecate**.

     Note: You can't deprecate the most recent version of a managed package.

   - To make a deprecated version available for installation again, click **Undeprecate**.

View Unused Components in a Managed Package

View components no longer being used in the current version of a package.

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Any component shown here that's part of a managed package is safe to delete unless you've used it in custom integrations. After you've deleted an unused component, it appears in this list for 15 days. During that time, you can either undelete it to restore the component and all data stored in it, or delete the component permanently. When you undelete a custom field, some properties on the field will be lost or changed. After 15 days, the field and its data are permanently deleted.

Note: Before deleting a custom field, you can keep a record of its data. From Setup, enter `Data Export` in the `Quick Find` box, then select **Data Export**.

The following component information is displayed (in alphabetical order):

Attribute	Description
`Action`	Can be one of two options: • **Undelete** • **Delete**
`Name`	Displays the name of the component.
`Parent Object`	Displays the name of the parent object a component is associated with. For example, a custom object is the parent of a custom field.
`Type`	Displays the type of the component.

Push Package Upgrades to Subscribers

A push upgrade is a method of automatically upgrading your customers to a newer version of your package. This feature can be used to ensure that all your customers are on the same or latest version of your package. You can push an upgrade to any number of organizations that have installed your managed package.

A package subscriber doesn't need to do anything to receive the push upgrade. The only indication a subscriber receives after a successful push upgrade is that the package's `Version Number` on the Package Detail page has a higher value. The developer initiating the push resolves upgrades that fail.

Use the Push Upgrade Exclusion List to exclude specific subscriber orgs from a push upgrade. You can specify up to 500 comma-separated org IDs.

Push upgrades minimize the potential risks and support costs of having multiple subscribers running different versions of your app. You can also automate many post-upgrade configuration steps, further simplifying the upgrade process for your customers.

The push upgrade feature is only available to first- and second-generation managed packages that have passed the AppExchange security review. To enable push upgrades for your managed package, log a support case in the Salesforce Partner Community. For details on the security review process, see Pass the AppExchange Security Review in the *ISVforce Guide*.

Push Upgrades

Push Upgrade Best Practices

Push Upgrade is one of the most powerful features we provide to our partners. Pushing an upgrade without proper planning and preparation can result in significant customer satisfaction issues. Here are some best practices to consider.

Assign Access to New and Changed Features in First- and Second-Generation Managed Packages
Determine how to provide existing non-admin users access to new and changed features. By default, any new components included in the push upgrade package version are assigned only to admins.

Sample Post Install Script for a Push Upgrade for First- and Second-Generation Managed Packages
Automate the assignment of new components to existing users of a package.

Scheduling Push Upgrades
After you've created an updated version of your package, you can automatically deploy it to customers using a push upgrade.

Push Upgrades

Overview of Push Upgrade Steps

- Upgrade your managed package installed in a customer organization from version X to version Y
- Select one, many, or all customer organizations to upgrade and select a particular version to upgrade to
- Schedule the upgrade to start at a particular date and time
- View progress of upgrades, abort upgrades in progress, or view the result of a push upgrade
- In conjunction with push, you can use a post-install Apex script to automate post-upgrade configurations that your customers have previously performed manually

Warning: When you push an upgrade, you're making changes to a subscriber's org without explicit consent. Therefore, it's important to plan ahead and exercise caution. You can also exclude specific subscriber orgs from a push upgrade by entering the org IDs, separated by a comma, in the Push Upgrade Exclusion List.

Pushing a major upgrade entails a higher degree of risk as it can impact existing functionality in a subscriber's organization. This is because new components in the upgraded package might not be available to existing users of the package, or could overwrite users' customizations. As the app developer, it's your responsibility to protect users from any adverse impact due to upgrading. We strongly recommend you consider all potential consequences of the upgrade and take appropriate steps to prevent any problems.

When pushing a major upgrade, we recommend that you divide changes in your package into two categories:

1. Enhancements to existing features that users already have access to—Use a post install Apex script to automatically assign the relevant components to existing users. This ensures all current users of the package can continue using it without explicit action by administrators.
2. New features you're introducing—Don't use a post install Apex script to auto-assign components. This ensures your subscribers have the opportunity to decide if and when to use the new features.

Here are some additional guidelines to keep in mind when planning a push upgrade.

- Avoid changes to validation rules, formula fields, and errors thrown from Apex triggers, as they may negatively impact subscribers' integrations.
- Don't make visible changes to a package in a patch. This is because other than a change in the package version number, subscribers aren't notified of push upgrades.
- Test your upgraded package in multiple environments, replicating all relevant features of your customers' organizations, including editions, customizations, other installed packages, and permission sets.
- Schedule push upgrades at your customers' off-peak hours and outside of Salesforce's major release windows, to minimize potential subscriber impact.
- Notify your subscribers in advance about the timing of the upgrade, its potential consequences, and any steps they need to take.

Push Upgrade Best Practices

Push Upgrade is one of the most powerful features we provide to our partners. Pushing an upgrade without proper planning and preparation can result in significant customer satisfaction issues. Here are some best practices to consider.

Plan, Test, and Communicate

- Share an upgrade timeline plan with your customers so they know when you'll upgrade, and how often.
- Plan when you want to push upgrades to your customers' orgs. Keep in mind that most customers don't want changes around their month-end, quarter-end, and year-end or audit cycles. Do your customers have other critical time periods when they don't want any changes to their org? For example, there might be certain times when they don't have staff available to verify changes or perform any required post-installation steps.
- Schedule push upgrades during your customers' off-peak hours, such as late evening and night. Have you considered time zone issues? Do you have customers outside the United States who have different off-peak hours? You can schedule push upgrades to any number of customer organizations at a time. Consider grouping organizations by time zone, if business hours vary widely across your customer base.
- Don't schedule push upgrades close to Salesforce-planned maintenance windows. In most cases, it might be better to wait 3-4 weeks after a major Salesforce release before you push major upgrades.
- Test, test, and test! Since you're pushing changes to the organization instead of the customer pulling in changes, there's a higher bar to ensure the new version of your app works well in all customer configurations.

Stagger Your Push Upgrades

- Don't push changes to all customers at once. It's important to ensure that you have sufficient resources to handle support cases if there are issues. Also, it's important that you discover possible issues before your entire customer base is affected.
- Push to your own test organizations first to confirm that the push happens seamlessly. Log in to your test organization after the push upgrade and test to see that everything works as expected.
- When applicable, push to the sandbox organizations of your customers first before pushing to their production organizations. Give them a week or more to test, validate, and fix in the sandbox environment before you push to their production organizations.
- Push upgrades to small batches of customer production organizations initially. For example, if you have 1,000 customers, push upgrades to 50 or 100 customers at a time, at least the first few times. After you have confidence in the results, you can upgrade customers in larger batches.

Focus on Customer Trust

- You're responsible for ensuring that your customers' organizations aren't adversely affected by your upgrade. Avoid making changes to the package, such as changes to validation rules or formula fields, that might break external integrations made by the customer. If for some reason you do, test and communicate well in advance. Please keep in mind that you can impact customer data, not just metadata, by pushing an upgrade that has bugs.
- Write an Apex test on install to do basic sanity testing to confirm that the upgraded app works as expected.
- If you're enhancing an existing feature, use a post-install script to automatically assign new components to existing users using permission sets.
- If you're adding a new feature, don't auto-assign the feature to existing users. Communicate and work with the admins of the customer org so they can determine who should have access to the new feature, and the timing of the rollout.

Assign Access to New and Changed Features in First- and Second-Generation Managed Packages

Determine how to provide existing non-admin users access to new and changed features. By default, any new components included in the push upgrade package version are assigned only to admins.

If the push upgrade includes:	We recommend you:
New features	Notify admins about the changes the upgrade introduces, and ask them to assign permissions to all users of the package. This approach allows admins to choose when to make the new features available.
Enhancements to existing features	Include a post-install script in the package that assigns permissions to the new components or new fields automatically. This approach ensures that current users of the package can continue using features without interruption. Note: Post-install scripts aren't available to unlocked packages.

Sample Post Install Script for a Push Upgrade for First- and Second-Generation Managed Packages

Automate the assignment of new components to existing users of a package.

Note: Post-install scripts can be used with first and second-generation managed packages only.

For more information on writing a post-install Apex script, see Run Apex on Package Install/Upgrade on page 64.

In this sample script, the package upgrade contains new Visualforce pages and a new permission set that grants access to those pages. The script performs the following actions.

- Gets the Id of the Visualforce pages in the old version of the package
- Gets the permission sets that have access to those pages
- Gets the list of profiles associated with these permission sets
- Gets the list of users who have those profiles assigned
- Assigns the permission set in the new package to those users

```
global class PostInstallClass implements InstallHandler {
    global void onInstall(InstallContext context) {

        //Get the Id of the Visualforce pages
        List<ApexPage> pagesList = [SELECT Id FROM ApexPage WHERE NamespacePrefix =
            'TestPackage' AND Name = 'vfpage1'];

        //Get the permission sets that have access to those pages
        List<SetupEntityAccess> setupEntityAccessList = [SELECT Id,
            ParentId, SetupEntityId, SetupEntityType FROM SetupEntityAccess
            WHERE SetupEntityId IN :pagesList];
        Set<ID> PermissionSetList = new Set<ID> ();

        for (SetupEntityAccess sea : setupEntityAccessList) {
            PermissionSetList.add(sea.ParentId);
        }
        List<PermissionSet> PermissionSetWithProfileIdList =
            [SELECT id, Name, IsOwnedByProfile, Profile.Name,
            ProfileId FROM PermissionSet WHERE IsOwnedByProfile = true
            AND Id IN :PermissionSetList];

        //Get the list of profiles associated with those permission sets
        Set<ID> ProfileList = new Set<ID> ();
        for (PermissionSet per : PermissionSetWithProfileIdList) {
            ProfileList.add(per.ProfileId);
        }

        //Get the list of users who have those profiles assigned
        List<User> UserList = [SELECT id FROM User where ProfileId IN :ProfileList];


        //Assign the permission set in the new package to those users
        List<PermissionSet> PermissionSetToAssignList = [SELECT id, Name
            FROM PermissionSet WHERE Name='TestPermSet' AND
            NamespacePrefix = 'TestPackage'];
        PermissionSet PermissionSetToAssign = PermissionSetToAssignList[0];
        List<PermissionSetAssignment> PermissionSetAssignmentList = new
List<PermissionSetAssignment>();
        for (User us : UserList) {
            PermissionSetAssignment psa = new PermissionSetAssignment();
            psa.PermissionSetId = PermissionSetToAssign.id;
            psa.AssigneeId = us.id;
            PermissionSetAssignmentList.add(psa);
        }
        insert PermissionSetAssignmentList;
    }
}
```

```
// Test for the post install class
@isTest
private class PostInstallClassTest {
    @isTest
    public static void test() {
      PostInstallClass myClass = new PostInstallClass();
      Test.testInstall(myClass, null);
    }
}
```

Scheduling Push Upgrades

USER PERMISSIONS

To push an upgrade:
- Upload AppExchange Packages

After you've created an updated version of your package, you can automatically deploy it to customers using a push upgrade.

1. Push the upgrade to your own orgs so you can run tests and fix any bugs before upgrading subscribers.
2. When you're ready and have coordinated with your customers on their change management process, push to a small number of customer organizations. Try sandbox organizations first, if possible.
3. After you're comfortable with the initial results, push to your wider customer base, based on your agreements with each customer.
4. Deprecate the previous version of your package in your main development organization. Replace the version on AppExchange if necessary, and update your Trialforce setup.
5. If your upgrade was a patch, after you've successfully distributed the upgrade to subscriber organizations, reintegrate those changes into your main development organization. For more information about combining patches in the main development organization, see Working with Patch Versions on page 74.

Schedule a Push Upgrade Using the UI

Note: Only first-generation managed packages can schedule a push upgrade using the UI. To schedule a push upgrade for unlocked and second-generation managed packages, use the PackagePushRequest in the *Salesforce Object Reference*.

1. Log in to your main development org (not the patch org you used to upload the new version).
2. From Setup, enter `Packages` in the `Quick Find` box, then select **Packages**.
3. Click the name of the managed package whose upgrade you want to push.
4. On the package detail page, click the **Versions** tab, and then click **Push Upgrades**.
5. Click **Schedule Push Upgrades**.
6. Select a package version to push from the **Patch Version** dropdown list.

   Note: Beta versions aren't eligible for push.

7. For the scheduled start date, enter when you want the push upgrade to begin.
8. In the Select Target Organizations section, select the orgs to receive your push upgrade. If an org already received a push upgrade for the selected package version, it doesn't appear in this list. You can select orgs by:
   - Entering a term that filters based on an org's name or ID. Names can match by partial string, but IDs must match exactly.
   - Choosing between production and sandbox orgs from the **Organizations** dropdown list.
   - Choosing orgs that have already installed a particular version.
   - Clicking individual orgs or the **Select All** and **Deselect All** checkboxes.

   This section lists the following information about the org (in alphabetical order):

Field	Description
`Current Version`	The current package version an organization has installed.
`Organization ID`	The ID of the org.
`Organization Name`	The name of the org. To view the upgrade history for the org, click the org name.

Field	Description
`Primary Contact`	The name of the user who installed the package.

9. Click **Schedule**. While a push upgrade is in progress, you can click Abort to stop it.

Schedule a Push Upgrade Using the Enterprise API

1. Authenticate to your main development org (not the patch org you used to upload the new version) according to the tool you're using.

   Note: For unlocked and second-generation managed packages, authenticate to your Dev Hub.

2. Determine the package version you want to upgrade subscribers to by querying the MetadataPackageVersion object.
3. Gather the list of subscriber orgs that are eligible to be upgraded by querying the PackageSubscriber object.

   Note: If you're retrieving more than 2,000 subscribers, use SOAP API `queryMore()` call.

4. Create a PackagePushRequest object. PackagePushRequest objects take a PackageVersionId and, optionally, a ScheduledStartTime parameter to specify when the push begins. If you omit the ScheduledStartTime, the push begins when you set the PackagePushRequest's status to `Pending`.
5. Create a PackagePushJob for each eligible subscriber and associate it with the PackagePushRequest you created in the previous step.
6. Schedule the push upgrade by changing the status of the PackagePushRequest to `Pending`.
7. Check the status of the PackagePushRequest and PackagePushJob objects by querying the `Status` fields. If the status is either Created or Pending, you can abort the push upgrade by changing the status of the PackagePushRequest to Canceled. You can't abort a push upgrade that has a status of Canceled, Succeeded, Failed, or In Progress.

   Note: If you're pushing the upgrade to more than 2,000 subscribers, use the Bulk_API to process the job in batches.

For sample code and more details, see *SOAP API Developer Guide*.

Manage Licenses for Managed Packages

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

Use the License Management App (LMA) to manage leads and licenses for your AppExchange solutions. By integrating the LMA into your sales and marketing processes, you can better engage with prospects, retain existing customers, and grow your ISV business. The LMA is a managed package that is installed in all partner business orgs (PBO) and includes custom objects that track details on packages, package versions, and licenses.

I need to...	Permissions	For details, see...
Configure the LMA	System Admin profile	Get Started with the License Management App on page 91
Bill subscribers or monitor license expiration	Object Permissions: Read	Lead and License Records in the LMA on page 94

I need to...	Permissions	For details, see...
Convert trial subscriptions into paying customers	Object Permissions: Edit	Modify a License Record on page 94
Customize the LMO	Object Permissions: Edit	Extend the LMA on page 95
Provision licenses to a subscriber	Object Permissions: Edit	Modify a License Record on page 94
Support subscribers with technical issues	Various permissions (see Assign Permissions to the Subscriber Support Console on page 93 for details)	Troubleshoot Subscriber Issues

Note: The LMA is available only in English.

The LMA is available to eligible Salesforce partners. For more information on the Partner Program, including eligibility requirements, visit https://partners.salesforce.com.

Get Started with the License Management App
To start managing leads and licenses with the License Management App (LMA), complete these installation and configuration steps.

Lead and License Records in the License Management App
Each time a customer installs your managed package, the License Management App (LMA) creates lead and license records.

Modify a License Record
You can change a customer's access to your offering by modifying a license record using the License Management App (LMA). For example, you can increase or decrease the number of seats included with a license or change the expiration date.

Refresh Licenses for a Managed Package
To sync all license records for a package across all subscriber installations, you refresh the license. Refreshing the license can also resolve discrepancies between the number of licenses in a subscriber's org and the number displayed in the License Management App (LMA). Refreshing is required when you move the LMA to a different org.

Extending the License Management App
The License Management App (LMA) is a managed package that you can customize and extend. In addition to using the LMA to manage leads and licenses, many partners also integrate it into their existing business processes.

Move the License Management App to Another Salesforce Org
You can move an LMA to a different org, but your package and license records don't automatically move with it. You must manually relink your packages and refresh the licenses.

Troubleshoot the License Management App
If you're experiencing issues with the License Management App, review these troubleshooting tips.

Best Practices for the License Management App
Follow these best practices when you use the License Management App (LMA).

Troubleshoot Subscriber Issues
Use the Subscriber Support Console to access information about your subscribers. Subscribers can also grant you login access to troubleshoot issues directly within your app. After you're granted access, you can log in to the subscriber's org and view their configuration and data to troubleshoot and resolve issues.

Get Started with the License Management App

To start managing leads and licenses with the License Management App (LMA), complete these installation and configuration steps.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

Install the License Management App
The License Management App (LMA) is a managed package that is installed in all partner business orgs. The org that the LMA is installed in is called the License Management Org (LMO).

Associate a Package with the License Management App
To receive lead and license records for your package, you connect your License Management Org (LMO), your package, and the Salesforce Partner Console. Your LMO is the Salesforce org where the License Management App (LMA) is installed.

Configure Permissions for the License Management App
Determine who needs access to the License Management App (LMA), and set object permissions. Consider using a permission set to assign user permissions.

Install the License Management App

The License Management App (LMA) is a managed package that is installed in all partner business orgs. The org that the LMA is installed in is called the License Management Org (LMO).

USER PERMISSIONS

To install packages:
- Download AppExchange Packages

We strongly recommend that you use your partner business org (PBO) as your LMO. However, you can choose to install the LMA in another production org. Consider installing the LMA in an org that your company is already using to manage sales, billing, and marketing.

Commercial use of the LMA is prohibited in Developer and Partner Developer Edition orgs. Installing the LMA in a Developer Edition org is allowed only if you're building integrations with the LMA and need an environment only for development and testing purposes. You can install the LMA in Enterprise, Unlimited, or Performance Edition production orgs.

It's not possible to have Slack or the Declarative Lookup Rollup Summary (DLRS) package installed in the same org as the LMA. If the org in which you plan to install the LMA has either Slack or the DLRS package installed, uninstall them before you install the LMA. Alternatively, install the LMA in a different org.

Note: To confirm whether your PBO already has the LMA installed, skip to step 4.

1. To install the LMA in an org other than your PBO, log a case in the Partner Community. After we review the case, you receive an email with an installation URL.
2. Log in to the org where you want to install the LMA, and then go to the installation URL included in the email.
3. Choose which users can access the LMA, and then click **Install**.
4. To confirm that the LMA is installed, open the App Launcher. If the installation was successful, the License Management App appears in the list of available apps.

Associate a Package with the License Management App

To receive lead and license records for your package, you connect your License Management Org (LMO), your package, and the Salesforce Partner Console. Your LMO is the Salesforce org where the License Management App (LMA) is installed.

USER PERMISSIONS

To manage licenses in the Partner Community:
- Manage Listings

A single LMO can manage multiple 1GP and 2GP packages, but a package can be associated with only one LMO.

1. Connect your packaging org (for 1GP) or your Dev Hub org (for 2GP) to the Partner Console.
   a. Log in to the Partner Community, and select the **Publishing** tab.
   b. Click **Technologies** > **Orgs**.
   c. Click **Connect Technology**, and then click **Org**.
   d. Click **Connect Org**.
   e. Log in to the org.

      For 1GP packages, enter the login credentials for the packaging org. Repeat this step for all your 1GP packages.

      For 2GP packages, enter the login credentials for the Dev Hub org. When you connect the Dev Hub org, all the 2GP packages owned by the Dev Hub org are linked to the Partner Console.

2. Select the **Solutions** tab.
3. Locate the package you want to register with the LMO. To register each package you own, repeat this step.
   a. Click the down arrow to expand the list of versions for your package.
   b. Click **Register Package** for the package version you want to register.

      Package versions created after linking to your LMO inherit the association.

   c. To register the package, log in to your LMO.
4. Set the default behavior you want for your package license, and then click **Save**.

After the package is registered, a license is created when customers install it. You can view which packages are registered in the LMA.

Note: Beta package versions don't display in the LMA. Only managed-released package versions (1GP) and promoted package versions (2GP) are visible in the LMA. Unlocked packages aren't supported.

Configure Permissions for the License Management App

Determine who needs access to the License Management App (LMA), and set object permissions. Consider using a permission set to assign user permissions.

Ensure that you:

- Install the LMA.
- Connect your packaging org (for 1GP) or your Dev Hub org (for 2GP) to the AppExchange Partner Console.
- Associate your package with the LMA.

1. Set object permissions for the license, package, and package version custom objects.

Custom Object	Object Permissions
License	To view license records: Assign READ permissions To modify license records: Assign READ and EDIT permissions
Package	To view package records: Assign READ permissions

Custom Object	Object Permissions
	To modify package records: Assign READ and EDIT permissions
Package Version	To view package version records: Assign READ permissions We recommend leaving all package version records as read-only.

2. Set field-level security in user profiles or permission sets.

Custom Object	Field-Level Permissions
License	Make all fields read-only.
Package	Make all fields read-only.
Package Version	Make all fields read-only.

3. Add related lists to page layouts.

To enable...	Add the Licenses related list to the...
License managers to view the licenses associated with a particular lead	Lead page layout
LMA users to view the licenses associated with a particular account	Account page layout
LMA users to view the licenses associated with a particular contact	Contact page layout

Assign Permissions to the Subscriber Support Console
Create a permission set to provide users access to the Subscriber Support Console.

Assign Permissions to the Subscriber Support Console

Create a permission set to provide users access to the Subscriber Support Console.

Note: If you've already assigned these permissions via a profile or another permission set, you can skip this task.

1. From Setup, in the Quick Find box, enter `Permission Sets`, and select **Permission Sets**.
2. Click **New** and enter your permission set information.
3. On the Permission Set Overview page, locate the Apps section, and select **Visualforce Page Access**.
   a. Click **Edit**.

   b. Add **sfLma.LoginToPartnerBT** and **sfLma.SubscriberSupport** to the list of Enabled Visualforce pages, and then click **Save**.

4. On the Permission Set Overview page, locate the System section, and select **System Permissions**. Click **Edit**.
   a. Select **Log in to Subscriber Organization**, and click **Save**.
5. From Setup, in the Quick Find box, enter `Profiles`, and select **Profiles**.
   a. Click **Edit**.
   b. Under Custom App Settings, select **License Management App**.
   c. Under Custom Tab Settings, locate the Subscribers tab and select **Default On**.
   d. Click **Save**.

Lead and License Records in the License Management App

Each time a customer installs your managed package, the License Management App (LMA) creates lead and license records.

The key objects in the LMA are Package, Lead, and License.

- Package—The LMA includes a Package custom object and a Package Version custom object. These objects display details about each 1GP or 2GP package and package version you've listed on AppExchange.
- Lead —The Lead standard object gives you details about who installed your package, such as the installer's name, company, and email address. Lead records created by the LMA are just like the ones you use elsewhere in Salesforce, except the lead source is Package Installation. You can manually convert leads into accounts and contacts. When you convert a lead, the license record links to the converted account or contact.
- License—The License custom object gives you control over how many users in the customer's org can access your package and for how long. Each license record links to a lead record and a package record.

To understand which actions you must take and which actions the LMA handles for you, review this table.

Action	Who Takes This Step
Your package is installed by a new subscriber.	Customer or prospect
A lead record is created with the customer's name, company, and email address.	LMA
A license record is created according to the values you specified when you registered the package.	LMA
The lead record is converted to account and contact records. (Optional)	You (ISV partner)
Account and contact records are associated with the license record.	LMA

Modify a License Record

You can change a customer's access to your offering by modifying a license record using the License Management App (LMA). For example, you can increase or decrease the number of seats included with a license or change the expiration date.

1. In the LMA, locate the license.
2. Click **Modify License**.

   When the LMA is installed, the Edit button doesn't appear on the license page layout, and the Modify License button is included instead. This setup is intentional. Only edit license records on the Modify License page.
3. Update the field values as needed.

Field	Description
`Expiration`	Enter the last day that the customer can access your package, or select **Does not expire**.
`Seats`	Enter the number of licensed seats, or select **Site License** to make your package available to all users in the customer's org. You can allocate up to 99,000,000 seats.
`Status`	Select a value from the dropdown. • **Trial**—Lets the customer try your offering for up to 90 days. After the trial license converts to an active license, it can't return to a trial state. • **Active**—Lets the customer use your package according to the license agreement. • **Suspended**—Prohibits the customer from accessing your offering. Note: When your offering is uninstalled, its status is set to Uninstalled, and the license can't be edited.

4. Click **Save**.

Refresh Licenses for a Managed Package

To sync all license records for a package across all subscriber installations, you refresh the license. Refreshing the license can also resolve discrepancies between the number of licenses in a subscriber's org and the number displayed in the License Management App (LMA). Refreshing is required when you move the LMA to a different org.

Note: For each package, you can refresh licenses only one time per week.

1. From the LMA, select the **Packages** tab.
2. Open the package record.
3. Click **Refresh Licenses**. In Lightning Experience, Refresh Licenses is located in the dropdown menu.

Extending the License Management App

The License Management App (LMA) is a managed package that you can customize and extend. In addition to using the LMA to manage leads and licenses, many partners also integrate it into their existing business processes.

The LMA includes these custom objects:

- License on page 96
- Package on page 96
- Package Version on page 96

You can add custom fields to the objects as long as you don't mark your custom fields as required.

Package and Package Version Object Fields
The License Management App (LMA) includes a Package custom object and a Package Version custom object. These objects display details about each 1GP or 2GP package and package version you've listed on AppExchange.

License Object Fields

Use the License custom object to set limits on how many users in the subscriber's org can use your app and for how long.

Adding Custom Automation to License Management App Objects

Here are some examples of how you can use the License Management App (LMA) to grow your business and retain customers.

Package and Package Version Object Fields

The License Management App (LMA) includes a Package custom object and a Package Version custom object. These objects display details about each 1GP or 2GP package and package version you've listed on AppExchange.

To view details about a package record, from the LMA, select the **Packages** tab, and then select the package name. You can view package versions in the Package Version related list.

Note: The LMA creates the package records, which contain critical information for tracking your licenses and packages. Treat these fields as read-only and ensure that your object permissions protect package records.

Package Custom Object Fields	Description
`Developer Name`	The name of the org that owns the package. For 1GP, the org name is the packaging org. For 2GP, it's the Dev Hub org.
`Developer Org ID`	The 18-character ID of the org that owns the package. For 1GP, the org ID is the packaging org ID. For 2GP, it's the Dev Hub org ID.
`Last License Refresh`	The date when the License Refresh tool was last run.
`Latest Version`	The most recent package version you've released.
`Lead Manager`	The owner of the lead records that the LMA creates when a customer installs your package.
`Next Available Refresh`	The date when the License Refresh tool can be run again.
`Owner`	The LMA owns all package records.
`Package ID`	The 18-character ID that identifies the package. This ID starts with 033.
`Package Name`	The name you specified when you created the package.

Package Version Object Fields	Description
`Package`	The package name and links to the package record's detail page.
`Package Version Name`	The name you specified when you created the package version.
`Release Date`	The date you created this package version.
`Version Number`	The version number in major.minor.patch format. For example, 3.1.0.
`Version ID`	The 18-character ID of this package version.

License Object Fields

Use the License custom object to set limits on how many users in the subscriber's org can use your app and for how long.

The License Management App (LMA) creates a license record every time your package is installed in an org. For example, if a subscriber installs two of your 1GP packages and three of your 2GP packages, you have five license records for that subscriber in your LMA. If you deliver a 2GP app that is composed of multiple packages, a unique license record is created for each package in the app. You can allocate up to 99,000,000 seats per subscriber license.

To view details about a license record, select the **Licenses** tab in the LMA, and then select and open the license record.

License records are automatically created and contain critical information for tracking licenses. Do not directly edit the license record. Instead, use the Modify License on page 94 tool to change the expiration date, license status, and the number of licensed seats.

License Custom Object Fields	Description
`Account`	A lookup field to the account record for a converted lead.
`Contact`	A lookup field to the contact record for a converted lead.
`Created By`	License records are always created by the LMA.
`Expiration Date`	Displays the expiration date or `Does not expire` (default).
`Install Date`	The date the subscriber installed this package version.
`Instance`	The Salesforce instance where the subscriber's org resides.
`Lead`	The lead record that the LMA created when the package was installed. A lead represents the user who owns the license. If you convert the lead into an opportunity, the lead name is retained but the lead record no longer exists.
`License Name`	An auto-generated number that represents an instance of a license. License names are in the format of L-00001, and each new license is incremented by one.
`Licensed Seats`	Displays the number of licenses or `Site License` (default).
`License Status`	The type of license: Active, Suspended, Trial, or Uninstalled.
`License Type`	This is a legacy field and can be ignored.
`Org Edition`	The edition of the subscriber's org.
`Org Expiration Date`	Applies only if the subscriber installs your package in a trial org. Indicates the date when the trial org expires. It isn't related to the package license expiration.
`Org Status`	The status of the subscriber's org: Active, Free, or Trial.
`Owner`	The LMA owns all license records. Don't edit this field.
`Package Version`	A lookup field that links to the package version associated with this license.
`Package Version Number`	The version number in major.minor.patch format. For example, 3.1.0.
`Sandbox`	Indicates whether the license is for a package installed in a sandbox org.
`Subscriber Org ID`	The 15-character ID representing the subscriber's org.
`Used Licenses`	Displays the number of users who have a license to the package. This field is blank if: • A customer uninstalled the package.

License Custom Object Fields	Description
	• `Licensed Seats` is set to Site License.

Adding Custom Automation to License Management App Objects

Here are some examples of how you can use the License Management App (LMA) to grow your business and retain customers.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

Alert Sales Reps Before a License Expires

If you're managing licenses for several packages, it can be difficult to track the various expirations. If a license expires accidentally, you could even lose a customer. To help your customers with renewals, set up a workflow rule to email a sales rep on your team before the license expires.

To automatically email the sales rep, follow these high-level steps.

1. Create an email template for the notification.
2. Create a workflow rule with a filter that specifies enough time before the expiration date to discuss renewal options.
3. Associate the workflow rule with a workflow alert that sends an email to the appropriate team member or sales rep.

Notify Customer-Retention Specialists When an Offering Is Uninstalled

If a customer uninstalls your offering, find out why. By speaking to the customer, you have an opportunity to restore the business relationship or receive feedback that helps you improve your offering.

To notify a customer-retention specialist on your team, follow these high-level steps.

1. Create an email template for the notification.
2. Create a workflow rule with a filter that specifies that the `License Status` equals *`Uninstalled`*.
3. Associate the workflow rule with a workflow alert that sends an email to the retention specialist.

Move the License Management App to Another Salesforce Org

You can move an LMA to a different org, but your package and license records don't automatically move with it. You must manually relink your packages and refresh the licenses.

USER PERMISSIONS

To install packages:
- Download AppExchange Packages

To manage licenses in the Partner Community:
- Manage Listings

It's not possible to have Slack or the Declarative Lookup Rollup Summary (DLRS) package installed in the same org as the LMA. If the org in which you plan to install the LMA has either Slack or the DLRS package installed, uninstall them before you install the LMA. Alternatively, install the LMA in a different org.

1. To remove the association between the LMA and the org where it's currently installed, log a case with the Partner Community.
2. Install the LMA in the new org on page 91.
3. Associate your packages with the new org on page 91.
4. Refresh licenses for your packages on page 95.

Troubleshoot the License Management App

If you're experiencing issues with the License Management App, review these troubleshooting tips.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

Leads and Licenses Aren't Being Created in the License Management App

When a customer installs your package, leads and license records are created. If these records aren't being created, review these configurations in the License Management Org (LMO). If you resolve your issue using one of these recommendations, your missing licenses appear in the LMA within a few days.

Proxy User Has Deactivated Message in the LMA

If you're editing a license and see a "proxy user has deactivated" message, check whether the subscriber org is locked, deleted, or disabled.

Leads and Licenses Aren't Being Created in the License Management App

When a customer installs your package, leads and license records are created. If these records aren't being created, review these configurations in the License Management Org (LMO). If you resolve your issue using one of these recommendations, your missing licenses appear in the LMA within a few days.

Did the customer complete the package installation?

When a customer clicks **Get it Now** on your AppExchange listing, Salesforce counts this selection as an installation. However, the customer can cancel the installation before it's completed, or the installation could have failed. If the installation doesn't finish, a license isn't created.

Is State and Country picklist validation enabled?

To avoid state and country picklist-related lead failures, you have two options. Use the standard picklist integration values, or add duplicate states and countries to your picklists.

Standard picklist integration values

To implement this option, use the Salesforce standard state and country picklists in your org, and leave the integration values as-is. We recommend this option for most partners.

With this option, AppExchange leads propagate to your org with full state and country names, and the names match integration values in the standard picklists.

Add duplicate states and countries to your picklists.

Implement this option if you have a requirement to use the two-letter state or country abbreviations in your org. For example, you display abbreviations in the user interface or use them to integrate with other systems. Add duplicate states and countries to your picklists with different integration values. Set one value to the two-letter state or country abbreviation. Set the other value to the full state or country name. Make only the two-letter abbreviation picklist entries visible.

With this option, AppExchange leads propagate to your org with full state and country names, which match the full name integration values in your org. You also have two-letter integration values to use as needed.

Does the lead or license object have a trigger?

Don't use `before_create` or `before_update` triggers on leads and licenses. Instead, use `after_` triggers, or remove all triggers. If a trigger fails, it can block license creation.

Does the lead or license record have a required custom field?

If yes, remove the requirement. The LMA doesn't populate a required custom field, so it can prevent licenses or leads from being created.

Is the lead manager a valid, active user?
If not, the LMA can't create leads and licenses.

Does the lead or license record have a validation rule?
Validation rules often block the creation of LMA lead or license records because the required field isn't there.

Does the lead or license have a workflow rule?
Workflow rules sometimes prevent leads and licenses from being created. Remove the workflow rule.

Was the lead converted to an account?
When leads are converted to accounts, they're no longer leads.

Are you using standard duplicate rules for leads?
When a customer installs your package, the LMA checks for existing leads and contacts. If an existing contact matches the customer who installed your package, a lead record isn't created. To complete these checks, the LMA applies standard lead duplicate rules and matching rules. If you prefer to have the LMA associate every license with a lead regardless of whether there's an existing contact match, customize the standard duplicate rule for leads and remove the matching rule for contacts.

Proxy User Has Deactivated Message in the LMA

If you're editing a license and see a "proxy user has deactivated" message, check whether the subscriber org is locked, deleted, or disabled.

- If the org has been deleted, delete the corresponding license record.
- If the org is locked or if the package has been uninstalled, license records can't be updated.

Best Practices for the License Management App

Follow these best practices when you use the License Management App (LMA).

- To take advantage of entitlements that are unique to AppExchange partners, use your partner business org as your License Management Org.
- Create a list view filter for leads created by installed packages. The filter helps your team separate subscriber-based leads from leads coming from other sources.
- Use the API to find licensed users. The `isCurrentUserLicensed` method determines if a user has a license to a managed package. For more information, see the *Apex Reference Guide*.
- Treat the LMA custom objects as read-only. Use the Modify License page to edit licenses. Don't attempt to directly or programmatically edit license records.
- The LMA automatically creates package, package version, and license records. Customizations, such as adding required custom fields or creating workflow rules, triggers, or validation rules that require custom fields, can prevent the LMA from working properly.

Troubleshoot Subscriber Issues

Use the Subscriber Support Console to access information about your subscribers. Subscribers can also grant you login access to troubleshoot issues directly within your app. After you're granted access, you can log in to the subscriber's org and view their configuration and data to troubleshoot and resolve issues.

To access the Subscriber Overview page, click the organization's name from the **Subscribers** tab in the LMA.

Note: This feature is available to eligible Salesforce partners. For more information on the Partner Program, including eligibility requirements, see www.salesforce.com/partners.

Request Login Access from Subscribers
To log in to a subscriber org, first request login access from the subscriber.

Log In to Subscriber Orgs
After your subscriber has granted you login access, you can log in to the subscriber org to troubleshoot the issue.

Debug Subscriber Orgs
After logging in to a subscriber's org, you can view logs, obfuscated code in your package, and initiate ISV Customer Debugger sessions.

Request Login Access from Subscribers

To log in to a subscriber org, first request login access from the subscriber.

Ask the subscriber to enable either **Grant Account Login Access** or **Grant Login Access**. If they don't see your company listed, one of the following applies.

- A system admin disabled the ability for non-admins to grant access.
- The user doesn't have a license for the package.
- The package is licensed to the entire org. In this scenario, only an admin with the Manage Users permission can grant access.
- The org setting **Administrators Can Log in as Any User** is enabled.

Note: When the org setting **Administrators Can Log in as Any User** is disabled, login access is granted for a limited amount of time, and the subscriber can revoke access at any time.

Any changes you make while logged in as a subscriber are logged in the subscriber org's audit trail.

Log In to Subscriber Orgs

After your subscriber has granted you login access, you can log in to the subscriber org to troubleshoot the issue.

Available in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To log in to subscriber orgs:
- Log in to Subscriber Org

Note: You can only log in to orgs with a Salesforce Platform or full Salesforce license. You can't log in to subscriber orgs on Government Cloud instances.

Multi-Factor Authentication Required to Log In to a Subscriber Org

Starting in Spring '22, multi-factor authentication (MFA) is required when logging into the License Management Org (LMO). MFA is required only for LMO users who require access to the Subscriber Support Console. This requirement provides subscribers an extra layer of security by verifying the identity of the user accessing their org. You also have more control over which users log in to a subscriber org.

Determine which users require access to the Subscriber Support Console, and then set up multi-factor authentication (MFA) for those users.

Log In to a Subscriber Org

After you've logged in to the LMO using multi-factor authentication (MFA), and your subscriber has granted you login access, you're ready to log in.

1. In the License Management App (LMA), click the **Subscribers** tab.
2. To find a subscriber org, enter a subscriber name or org ID in the search box, and click **Search**.
3. Click the name of the subscriber org.
4. On the Org Details page, click **Login** next to a user's name. You have the same permissions as the user you logged in as.
5. When you're finished troubleshooting, log out of the subscriber org.

Note: Some subscribers require MFA in addition to the MFA required for the LMO. Ask your subscriber if their org requires MFA to log in. If so, your login attempt sends an MFA notification to your subscriber, and your login is blocked until your subscriber responds to the notification. To ensure that your subscriber is available to respond to the MFA notification, consider coordinating a specific login time.

Best Practices for Logging In

- Create an audit trial that indicates when and why a subscriber org login has occurred. You can create an audit trail by logging a case in your LMO before each subscriber org login.
- When you access a subscriber org, you're logged out of your LMO. You can set up a My Domain to not be automatically logged out of your LMO when you log in to a subscriber org. To set up a My Domain subdomain, from Setup, in the Quick Find box, enter `My Domain`, then select **My Domain**.
- Allow only trusted support and engineering personnel to log in to a subscriber's org. Because this feature can include full read/write access to customer data and configurations, it's vital to your reputation to preserve their security.
- Control who has login access by giving the Log in to Subscriber Org user permission to specific support personnel via a profile or permission set. See Assign Permissions to the Subscriber Org Console on page 93.

Debug Subscriber Orgs

After logging in to a subscriber's org, you can view logs, obfuscated code in your package, and initiate ISV Customer Debugger sessions.

Troubleshoot with Debug Logs

You can debug your code by generating Apex debug logs that contain the output from your managed package. Using this log information, you can troubleshoot issues that are specific to that subscriber.

1. If the user has access, set up a debug log: From Setup, in the Quick Find box, enter `Debug Logs`, and then select **Debug Logs**.
2. Launch the Developer Console.
3. Perform the operation, and view the debug log with your output.

Subscribers are unable to see the logs you set up or generate because they contain your unobfuscated Apex code.

You can also view and edit data contained in protected custom settings from your managed packages when logged in as a user.

Troubleshoot with the ISV Debugger

Each License Management Org can use one free ISV Customer Debugger session at a time. The ISV Customer Debugger is part of the Salesforce Extensions for Visual Studio Code. You can use the ISV Customer Debugger only in sandbox orgs, so you can initiate debugging sessions only from a customer's sandbox.

For details, see the ISV Customer Debugger documentation.

Manage Features in First-Generation Managed Packages

Control how you release features to customers with the Feature Management App (FMA). The FMA extends the functionality of the License Management App (LMA). Use the FMA to manage features as easily as you manage licenses with the LMA.

Here at Salesforce, we sometimes run pilot programs, like the one we ran when we introduced Feature Management. Sometimes we dark-launch features to see how they work in production before sharing them with you. Sometimes we make features available to select orgs for limited-time trials. And sometimes we want to track activation metrics for those features.

With feature parameters, we're extending this functionality to you. Install the FMA in your License Management Org (LMO). The FMA extends the License Management App, and like the LMA, it's a managed package.

Feature Parameter Metadata Types and Custom Objects

Feature parameters are represented as Metadata API types in your packaging org, as records of custom objects in your License Management Org, and as hidden records in your subscriber's org.

Set Up Feature Parameters

Set up the Feature Management App in your License Management Org, define feature parameters, and add them to your package.

Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features

Feature parameters with a data flow direction value of `LMO to Subscriber` are writable at your end and read-only in your subscriber's org. These feature parameters serve as permissions or limits. Use LMO-to-subscriber feature parameters to enable or disable new features or to control how many of a given resource your subscriber can use. Or, enable features for a limited trial period. Assign values to LMO-to-subscriber feature parameters by updating junction object records in your LMO, and then check those values in your code.

Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters

Use subscriber-to-LMO feature parameters to track feature activation in your subscriber's org. Parameter values are assigned on the subscriber's end and then sent to your LMO. To collect the values, update the feature parameters in your subscriber's org using Apex code. Check with your legal team before obtaining activation metrics from your customers. Use activation metrics to collect only aggregated data regarding feature activation.

Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs

Occasionally, you want to include custom permissions or custom objects in a package but not show them to your subscribers. For example, if you're piloting a feature for a few select orgs, and want to hide custom permissions and custom objects related to the pilot feature.

Best Practices for Feature Management

Here are some best practices when working with feature parameters.

Considerations for Feature Management

Keep these considerations in mind when working with feature parameters.

Feature Parameter Metadata Types and Custom Objects

Feature parameters are represented as Metadata API types in your packaging org, as records of custom objects in your License Management Org, and as hidden records in your subscriber's org.

Feature Parameter Fields

Feature parameters are represented as Metadata API types and store boolean, integer, or date values.

The first time a subscriber installs your package, a `FeatureParameter__c` record is created in your License Management Org (LMO) for each feature parameter. The feature parameter records include these fields:

- `FullName__c`
- `DataType__c` (`Boolean`, `Integer`, or `Date`)
- `DataFlowDirection__c`
- `Package__c`
- `IntroducedInPackageVersion__c`
- `Namespace_Prefix__c`

Lifecycle of a Feature Parameter

Set Up the Feature Parameter

Start by defining your feature parameter in the packaging org using the Feature Parameters tab on the Package detail page.

Depending on how you're using the feature parameter, you'll also write code that enables you to check access rights or collect usage information after the parameter is set up.

Subscriber Installs Your Managed Package

When a subscriber installs or upgrades your package in their org, a `FeatureParameter__c` record for each feature parameter is created in the LMO. If these records were created during a previous installation or upgrade, this step is skipped.

During package installation, junction object records are created in both the subscriber org and your LMO. A junction object is a custom object with two master-detail relationships. In this case, the relationships are between `FeatureParameter__c` and `License__c` in the LMO. These records store the value of their associated feature parameter for that subscriber org.

Utilize Your Feature Parameters

Use the junction objects to override the feature parameters' default values or to collect data. Depending on the value of each feature parameter's `DataFlowDirection__c` field, data flows to the subscriber org (from the LMO) or to the LMO (from the subscriber org). That data is stored in the junction object records.

Set Up Feature Parameters

Set up the Feature Management App in your License Management Org, define feature parameters, and add them to your package.

Install and Set Up the Feature Management App in Your License Management Org

Install the FMA in your LMO. Then add the Feature Parameters tab to your default view, and adjust your page layout for licenses to display related lists for your feature parameters.

Create Feature Parameters in Your Packaging Org

Create a feature parameter in your packaging org, and set its type, default value, and data flow direction.

Add Feature Parameters to Your Managed Package

After you've created some feature parameters, you can add them to a managed package as components and reference them in your code. Feature parameters aren't available in unmanaged packages.

Install and Set Up the Feature Management App in Your License Management Org

Install the FMA in your LMO. Then add the Feature Parameters tab to your default view, and adjust your page layout for licenses to display related lists for your feature parameters.

1. To request access to the FMA, log a support case in the Salesforce Partner Community. For product, specify **Partner Programs & Benefits**. For topic, specify **ISV Technology Request**. The FMA extends the License Management App, so be sure to install the LMA before requesting access to the FMA.
2. To install the FMA, follow the instructions in your welcome email.
3. Add the Feature Parameters tab to your default view. For details, see Customize My Tabs in Salesforce Help.
4. Update your page layout for licenses.
   a. Navigate to a license record's detail page.
   b. Click **Edit Layout**.
   c. In the Related Lists section of the License Page Layout Editor, add these lists.
      - Feature Parameter Booleans
      - Feature Parameter Dates
      - Feature Parameter Integers
   d. For each related list, add these columns.
      - Data Flow Direction
      - Feature Parameter Name
      - Full Name
      - Master Label
      - Value

Create Feature Parameters in Your Packaging Org

Create a feature parameter in your packaging org, and set its type, default value, and data flow direction.

1. From Setup, enter `Packages` in the Quick Find box, then select **Packages**.
2. In the Packages section, in the Package Name column, select your managed package.
3. On the Feature Parameters tab, click **New Boolean**, **New Integer**, or **New Date**.

   If the Feature Parameters tab isn't visible, log a case with Salesforce Partner Support.
4. Give your feature parameter a developer name that meets the standard criteria for developer names. The name must be unique in your org. It can contain only alphanumeric characters and underscores, and must begin with a letter. It can't include spaces, end with an underscore, nor contain two consecutive underscores.
5. Give the feature parameter a label.
6. Set a default value for the feature parameter. If you're creating a Feature Parameter Boolean, you see only a checkbox for Default Value. If you want your default value to be `true`, select this checkbox. Integer values can't exceed nine digits.
7. Set a data flow direction. To use this feature parameter to control behavior in your subscriber's org, select **LMO to Subscriber**. To collect activation metrics from your subscriber, select **Subscriber to LMO**. Note: After the feature parameter is included in a promoted and released package version, the data flow direction can't be changed.
8. Click **Save**.

Add Feature Parameters to Your Managed Package

After you've created some feature parameters, you can add them to a managed package as components and reference them in your code. Feature parameters aren't available in unmanaged packages.

A package can include up to 200 feature parameters.

Complete these steps in your packaging org.

1. From Setup, enter `Packages` in the Quick Find box, then select **Packages**.
2. In the Packages section, in the Package Name column, select your managed package.
3. On the Components tab, click **Add**.
4. From the Component Type dropdown, select **Feature Parameter Boolean**, **Feature Parameter Date**, or **Feature Parameter Integer**.
5. Select your feature parameter, and then click **Add to Package**.

Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features

Feature parameters with a data flow direction value of `LMO to Subscriber` are writable at your end and read-only in your subscriber's org. These feature parameters serve as permissions or limits. Use LMO-to-subscriber feature parameters to enable or disable new features or to control how many of a given resource your subscriber can use. Or, enable features for a limited trial period. Assign values to LMO-to-subscriber feature parameters by updating junction object records in your LMO, and then check those values in your code.

Assign Override Values in Your LMO
To override the default value of a feature parameter in a subscriber's org, update the appropriate junction object record in your LMO.

Check LMO-to-Subscriber Values in Your Code
You can reference feature parameters in your code, just like you'd reference any other custom object.

Assign Override Values in Your LMO

To override the default value of a feature parameter in a subscriber's org, update the appropriate junction object record in your LMO.

1. Open the license record for a subscriber's installation of your package.
2. In the related list for Feature Parameter Booleans, Feature Parameter Integers, or Feature Parameter Dates, select the feature parameter whose value you want to update.
3. Click **Edit**.
4. Set a value.
5. Click **Save**.

Check LMO-to-Subscriber Values in Your Code

You can reference feature parameters in your code, just like you'd reference any other custom object.

Use these Apex methods with LMO-to-subscriber feature parameters to check values in your subscriber's org.

- `System.FeatureManagement.checkPackageBooleanValue('`***`YourBooleanFeatureParameter`***`');`
- `System.FeatureManagement.checkPackageDateValue('`***`YourDateFeatureParameter`***`');`

- `System.FeatureManagement.checkPackageIntegerValue('YourIntegerFeatureParameter');`

SEE ALSO:

Apex Developer Guide: FeatureManagement Class

Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters

Use subscriber-to-LMO feature parameters to track feature activation in your subscriber's org. Parameter values are assigned on the subscriber's end and then sent to your LMO. To collect the values, update the feature parameters in your subscriber's org using Apex code. Check with your legal team before obtaining activation metrics from your customers. Use activation metrics to collect only aggregated data regarding feature activation.

- `System.FeatureManagement.setPackageBooleanValue('YourBooleanFeatureParameter', booleanValue);`
- `System.FeatureManagement.setPackageDateValue('YourDateFeatureParameter', datetimeValue);`
- `System.FeatureManagement.setPackageIntegerValue('YourIntegerFeatureParameter', integerValue);`

Warning: The `Value__c` field on subscriber-to-LMO feature parameters is editable in your LMO. But don't change it. The changes don't propagate to your subscriber's org, so your values will be out of sync.

SEE ALSO:

Apex Developer Guide: FeatureManagement Class

Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs

Occasionally, you want to include custom permissions or custom objects in a package but not show them to your subscribers. For example, if you're piloting a feature for a few select orgs, and want to hide custom permissions and custom objects related to the pilot feature.

Note: Check with your company's legal team before releasing hidden functionality.

To hide custom objects when creating your package, set the value of their Visibility field to `Protected`.

To hide custom permissions when creating your package, from Setup, enter *`Custom Permissions`* in the Quick Find box. Select **Custom Permissions** > *`Your Custom Permission`* > **Edit**. Enable **Protected Component**, and then click **Save**. After your package is installed, use the `System.FeatureManagement.changeProtection()` Apex method to hide and unhide custom objects and permissions.

Warning: After you've released unprotected objects to subscribers, you can't change the visibility to `Protected`.

To hide custom permissions in released packages:

- `System.FeatureManagement.changeProtection('YourCustomPermissionName', 'CustomPermission', 'Protected');`

To unhide custom permissions and custom objects in released packages:

- `System.FeatureManagement.changeProtection('`***`YourCustomPermissionName`***`', 'CustomPermission', 'Unprotected');`
- `System.FeatureManagement.changeProtection('`***`YourCustomObjectName__c`***`', 'CustomObject', 'Unprotected');`

SEE ALSO:

Apex Developer Guide: FeatureManagement Class

Best Practices for Feature Management

Here are some best practices when working with feature parameters.

- We recommend that you use this feature set in a test package and a test LMO before using it with your production package. Apply changes to your production package only after fully understanding the product's behavior.
- Create LMO-to-subscriber feature parameters to enable features from your LMO for individual subscriber orgs. Don't use the Apex code in your managed package to modify LMO-to-subscriber feature parameters' values in subscriber orgs. You can't send the modified values back to your LMO, and your records will be out of sync.

  Use LMO-to-subscriber feature parameters as read-only fields to manage app behavior. For example, use LMO-to-subscriber feature parameters to track the maximum number of permitted e-signatures or to make enhanced reporting available.
- Create subscriber-to-LMO feature parameters to manage activation metrics. Set these feature parameters' values in subscriber orgs using the Apex code in your managed package. For example, use subscriber-to-LMO feature parameters to track the number of e-signatures consumed or to check whether a customer has activated enhanced reporting.

Considerations for Feature Management

Keep these considerations in mind when working with feature parameters.

- After a feature parameter is included in a promoted and released package version, we recommend that you only edit the value field located in LMO-to-subscriber junction objects.

  Modifying or deleting other fields or records related to feature parameters, including the data flow direction, may cause the FMA to stop operating correctly.
- Don't use the LMO to create or delete feature parameters.
- When you update LMO-to-subscriber values in your LMO, the values in your subscribers' orgs are updated asynchronously. This process can take several minutes.
- When you publish a push upgrade to your managed package, feature parameters in your LMO and your subscribers' orgs are updated asynchronously. Creating and updating the junction object records can take several minutes.
- When the Apex code in your package updates subscriber-to-LMO values in your subscriber's org, the changes can take up to 24 hours to reach your LMO.

AppExchange App Analytics for First-Generation Managed Packages

AppExchange App Analytics provides usage data about how subscribers interact with your first-generation (1GP) managed packages and packaged components. You can use these details to identify attrition risks, inform feature development decisions, and improve user experience.

Note: AppExchange App Analytics is subject to certain usage restrictions as described in the AppExchange Program Policies. Usage data from Government Cloud and Government Cloud Plus orgs isn't available in App Analytics.

App Analytics is available for managed 1GP packages that passed security review and are registered to a License Management App. Usage data is provided as package usage logs, monthly package usage summaries, or subscriber snapshots. All usage data is available as downloadable comma-separated value (.csv) files. To view the data in dashboard or visualization format, use CRM Analytics or a third-party analytics tool.

In a 24-hour period, you can download a maximum 20 GB of AppExchange App Analytics data.

Enable App Analytics on Your First-Generation Managed Package
Activate AppExchange App Analytics on your first-generation (1GP) managed package to access AppExchange App Analytics package usage logs and subscriber snapshots. Package usage summaries are available by default.

SEE ALSO:

Get Started with AppExchange App Analytics

Enable App Analytics on Your First-Generation Managed Package

Activate AppExchange App Analytics on your first-generation (1GP) managed package to access AppExchange App Analytics package usage logs and subscriber snapshots. Package usage summaries are available by default.

1. Log in to your packaging org.
2. Click the **gear icon** , and select **Setup**.
3. In the Quick Find box, enter `package`, and select **Package Manager**.
4. Find your package, and click **Edit**.
5. Check **Enable AppExchange App Analytics**.
6. Save your work.

For full documentation on available App Analytics data and query best practices, read Get Started with AppExchange App Analytics in the *Second-Generation Managed Packaging Developer Guide*.

EDITIONS

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions.

USER PERMISSIONS

To access packages and package versions:
- Read on Packages, Package Versions

To request and retrieve AppExchange App Analytics data:
- Create, Read, Edit, Delete, View All, and Modify All on the AppAnalyticsQueryRequest object

Developing and Distributing Unmanaged Packages

Unmanaged packages can be used for distributing open-source projects to developers, or as a one-time drop of applications that require customization after installation.

After the components are installed from an unmanaged package, they can be edited in the org they're installed in. The developer who creates and uploads an unmanaged package has no control over the installed components, and can't change or upgrade them.

As a best practice, install an unmanaged package only if the org used to upload the package still exists. If that org is deleted, you may not be able to install the unmanaged package.

Don't use unmanaged packages for sandbox to production migration. Instead, use the Salesforce Extensions for Visual Studio Code or the Ant Migration Tool. If you're using Enterprise, Unlimited, or Performance Edition, see Change Sets.

Create and Upload an Unmanaged Package

Use the following procedure to upload an unmanaged package through the UI. You can also upload a package using the Tooling API. For sample code and more details, see the PackageUploadRequest object in the Tooling API Developer Guide.

Components Available in Unmanaged Packages

Not all components can be packaged for distribution.

Convert Unmanaged Packages to Managed

Create and Upload an Unmanaged Package

Use the following procedure to upload an unmanaged package through the UI. You can also upload a package using the Tooling API. For sample code and more details, see the PackageUploadRequest object in the Tooling API Developer Guide.

1. Create the package:
   a. From Setup, enter `Package Manager` in the `Quick Find` box, then select **Package Manager**.
   b. Click **New**.
   c. Fill in the details of the package.
   d. Click **Save**.
2. On the Components tab, click **Add**.
3. From the Component Type dropdown list, choose a component.
4. Select the component you want to add.
5. Click **Add To Package**.
6. Repeat these steps until you've added all the components you want in your package.
7. Click **Upload**.

You will receive an email that includes an installation link when your package has been uploaded successfully. Wait a few moments before clicking the installation link or distributing it to others, as it might take a few minutes for it to become active.

Considerations for Uninstalling Unmanaged Packages

If your unmanaged package has dependencies on metadata in another package, remove any dependencies before attempting to uninstall either package.

If you're working in a sandbox org, you must first remove the package dependencies in your production org.

1. Locate the unmanaged package in your production org and remove the dependencies to the package you plan to uninstall.
2. Create or refresh your sandbox org.
3. In your sandbox org, you can now uninstall the package that your unmanaged package previously depended on.

Components Available in Unmanaged Packages

Not all components can be packaged for distribution.

Packaged Explicitly or Implicitly

Components can be added either explicitly or implicitly. Explicit components must be included directly in the package, while implicit components are automatically added. For example, if you create a custom field on a standard object, you must explicitly add the custom field to your package. However, if you create a custom object and add a custom field to it, the field is implicitly added to the package when you add the custom object.

- **Explicitly**: The component must be manually added to the package.
- **Implicitly**: The component is automatically added to the package when another dependent component, usually a custom object, is added.

Automatic Renaming

Salesforce can resolve naming conflicts automatically on install.

- **No**: If a naming conflict occurs the install is blocked.
- **Yes**: If a naming conflict occurs Salesforce can optionally change the name of the component being installed.

Component	Packaged Explicitly or Implicitly	Automatic Renaming
Apex Class	Explicitly	No
Apex Sharing Reason	Implicitly On an extension: Explicitly	No
Apex Sharing Recalculation	Implicitly	No
Apex Trigger	On a standard or extension object: Explicitly On an object in the package: Implicitly	No
Application	Explicitly	No
Custom Button or Link	On a standard object: Explicitly On a custom object: Implicitly	No
Custom Field	On a standard object: Explicitly On a custom object: Implicitly	No
Custom Label	Implicitly	No
Custom Object	Explicitly	No
Custom Permission	Implicitly With required custom permissions: Explicitly	No
Custom Report Type	Explicitly	No
Custom Setting	Explicitly	No
Dashboard	Explicitly In a folder: Implicitly	Yes
Document* (10 MB limit)	Explicitly In a folder: Implicitly	Yes
Email Template (Classic)	Explicitly In a folder: Implicitly	Yes

Component	Packaged Explicitly or Implicitly	Automatic Renaming
External Data Source	Explicitly Referenced by an external object: Implicitly Assigned by a permission set: Implicitly	No
Flow Definition	Implicitly	No
Folder	Explicitly	Yes
Home Page Component	Explicitly	No
Home Page Layout	Explicitly	No
Inbound Network Connection	Explicitly	No
Letterhead	Explicitly	Yes
Lightning Application	Explicitly	No
Lightning Component	Explicitly	No
Lightning Event	Explicitly	No
Lightning Interface	Explicitly	No
Lightning Page	Explicitly	No
List View	On a standard object: Explicitly On a custom object: Implicitly	Yes
Named Credential	Explicitly	No
Outbound Network Connection	Explicitly	No
Page Layout	On a standard object: Explicitly On a custom object: Implicitly	No
Record Type	On a standard object: Explicitly On a custom object: Implicitly	No
Report	Explicitly In a folder: Implicitly	Yes
Reporting Snapshot	Explicitly	Yes
S-Control* (10 MB limit)	Explicitly	No
Static Resource	Explicitly	No
Tab	Explicitly	No
Translation	Explicitly	No

Component	Packaged Explicitly or Implicitly	Automatic Renaming
Validation Rule	On a standard object: Explicitly On a custom object: Implicitly	No
Visualforce Component	Explicitly	No
Visualforce Page	Explicitly	No
Workflow Email Alert	Explicitly	No
Workflow Field Update	Explicitly	No
Workflow Outbound Message	Explicitly	No
Workflow Rule	Explicitly	No
Workflow Task	Explicitly	No

*The combined size of S-Controls and documents must be less than 10 MB.

Convert Unmanaged Packages to Managed

Before you convert an existing package to managed, alert any current installers that they must save their data:

1. Export all the data from the previous, unmanaged version of the package.
2. Uninstall the unmanaged package.
3. Install the new managed version of the package.
4. Import all the exported data into the new managed package.

   Note: Note to installers: if you have made customizations to an installation of an unmanaged package, make a list of these customizations before uninstalling since you may want to implement them again.

To convert an unmanaged package into a managed package:

1. Register a namespace.
2. From Setup, enter `Package Manager` in the `Quick Find` box, then select **Package Manager**.
3. Edit the package that you want to make managed, then select **Managed**.

EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Developer** Edition

Package uploads and installs are available in **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To configure namespace settings:
- Customize Application

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages