

salesforce

---

# Second-Generation Managed Packaging Developer Guide

Version 62.0, Winter '25



Last updated: January 3, 2025



# CONTENTS

<b>Chapter 1: Second-Generation Managed Packages</b> .....	<b>1</b>
What's a Second-Generation Managed Package? .....	<b>3</b>
Why Switch to Second-Generation Managed Packaging? .....	<b>3</b>
Comparison of First- and Second-Generation Managed Packages .....	<b>5</b>
Set Up Your Development Environment .....	<b>6</b>
Enable Dev Hub and Second-Generation Managed Packaging .....	<b>6</b>
Limited Access License for Package Developers .....	<b>7</b>
Add a Limited Access User to Your Dev Hub Org .....	<b>8</b>
Assign Second-Generation Managed Packaging User Permissions .....	<b>8</b>
Before You Create Second-Generation Managed Packages .....	<b>9</b>
Know Your Orgs for Second-Generation Managed Packages .....	<b>9</b>
Link a Namespace to a Dev Hub Org .....	<b>10</b>
Namespaces for Second-Generation Managed Packages .....	<b>11</b>
Create and Register Your Namespace for Second-Generation Managed Packages .....	<b>11</b>
Key Concepts in Second-Generation Managed Packaging .....	<b>12</b>
How Manageability Rules and Ancestry Impact Upgrades for Second-Generation Managed Packages .....	<b>13</b>
Which Package Types Can Your Package Depend On? .....	<b>14</b>
Scratch Orgs and Package Development .....	<b>15</b>
How Scratch Orgs Fit in the Package Development Workflow .....	<b>16</b>
Scratch Org Definition Files vs Org Shape in Package Development .....	<b>17</b>
When to Use Scratch Org Snapshots in Package Development .....	<b>18</b>
Create a Package Version Based on a Scratch Org Snapshot .....	<b>19</b>
Get Access to Scratch Orgs That Have Agenforce .....	<b>20</b>
Scratch Org Allocations for Salesforce Partners .....	<b>22</b>
Manage Scratch Orgs from the Dev Hub Org .....	<b>22</b>
Supported Scratch Org Editions for Partners .....	<b>23</b>
Workflow for Second-Generation Managed Packages .....	<b>23</b>
Components Available in Second-Generation Managed Packages .....	<b>25</b>
Account Relationship Share Rule .....	<b>40</b>
Action Link Group Template .....	<b>41</b>
Action Plan Template .....	<b>42</b>
Actionable List Definition .....	<b>43</b>
Actionable List Key Performance Indicator Definition .....	<b>44</b>
Activation Platform .....	<b>45</b>
AffinityScoreDefinition .....	<b>46</b>
AI Application .....	<b>47</b>
AI Application Config .....	<b>48</b>
AIUsecaseDefinition .....	<b>49</b>

## Contents

Analytics	50
Apex Class	51
Apex Sharing Reason	53
Apex Trigger	54
Application Subtype Definition	55
AssessmentConfiguration	56
AssessmentQuestion	57
AssessmentQuestionSet	58
Aura Component	59
Batch Calc Job Definition	60
Batch Process Job Definition	61
Benefit Action	62
Bot Template	63
Branding Set	64
Briefcase Definition	65
Building Energy Intensity Record Type Configuration	66
Business Process	67
Business Process Group	68
Business Process Type Definition	69
Care Benefit Verify Settings	70
Care Limit Type	71
Care Request Configuration	72
Care System Field Mapping	73
Channel Layout	74
Chatter Extension	75
Claim Financial Settings	76
Community Template Definition	77
Community Theme Definition	78
Compact Layout	79
Conditional Formatting Ruleset	80
Connected App	81
Contract Type	83
Conversation Channel Definition	84
Conversation Vendor Info	85
CORS Allowlist	86
CSP Trusted Site	87
Custom Application	88
Custom Button or Link	89
Custom Console Components	91
Custom Field on Standard or Custom Object	92
Custom Field on Custom Metadata Type	93
Custom Help Menu Section	94
Custom Index	94
Custom Label	95

## Contents

Custom Metadata Type Records	96
Custom Metadata Type	97
Custom Notification Type	97
Custom Object	98
Custom Object Translation	100
Custom Permission	101
Custom Tab	102
Dashboard	104
DataCalcInsightTemplate	105
Data Connector Ingest API	106
Data Connector S3	107
Data Kit Object Dependency	108
Data Kit Object Template	109
Data Package Kit Definition	110
Data Package Kit Object	111
Data Source	113
Data Source Bundle Definition	114
Data Source Object	115
Data Src Data Model Field Map	116
Data Stream Definition	117
Data Stream Template	118
DataWeaveResource	120
Decision Matrix Definition	121
Decision Matrix Definition Version	122
Decision Table	123
Decision Table Dataset Link	124
Digital Experience	125
Digital Experience Bundle	126
Decision Table	127
Disclosure Definition	128
Disclosure Definition Version	129
Disclosure Type	130
Discovery AI Model	131
Discovery Goal	132
Discovery Story	133
Document	134
Document Generation Setting	135
Eclair GeoData	136
Email Template (Classic)	137
Email Template (Lightning)	137
Embedded Service Config	138
Embedded Service Menu Settings	139
Enablement Measure Definition	140
Enablement Program Definition	141

## Contents

Enablement Program Task Subcategory	143
Entitlement Template	144
ESignature Config	145
ESignature Envelope Config	146
Explainability Action Definition	147
Explainability Action Version	148
Explainability Message Template	149
Expression Set Definition	150
Expression Set Definition Version	151
Expression Set Object Alias	152
Expression Set Message Token	153
External Client App Header	154
External Client App OAuth Settings	155
External Credential	156
External Data Connector	158
External Data Source	159
External Data Transport Field Template	160
External Data Transport Object Template	161
External Document Storage Configuration	162
External Services	163
Feature Parameter Boolean	164
Feature Parameter Date	166
Feature Parameter Integer	167
Field Set	168
Field Source Target Relationship	170
Flow	171
Flow Category	173
Flow Definition	174
Flow Test	175
Folder	176
Fuel Type	177
Fuel Type Sustainability Unit of Measure	178
Fundraising Config	179
Gateway Provider Payment Method Type	180
Gen AI Function	181
Global Picklist	182
Home Page Component	183
Home Page Layout	185
Identity Verification Proc Def	186
Inbound Network Connection	187
IndustriesEinsteinFeatureSettings	188
IntegrationProviderDef	189
LearningAchievementConfig	190
Learning Item Type	191

## Contents

Letterhead	192
Lightning Bolt	193
Lightning Message Channel	194
Lightning Page	194
Lightning Web Component	195
List View	197
Live Chat Sensitive Data Rule	198
Loyalty Program Setup	199
Marketing App Extension	200
MarketingAppExtAction	201
Marketing App Extension Activity	203
Market Segment Definition	204
MktCalculatedInsightsObjectDef	205
MktDataConnection	206
MktDataTranObject	208
Named Credential	209
Object Source Target Map	211
OcrSampleDocument	212
OcrTemplate	213
Outbound Network Connection	214
Page Layout	216
Path Assistant	217
Payment Gateway Provider	218
Permission Set	218
Permission Set Groups	220
Platform Cache	220
Platform Event Channel	221
Platform Event Channel Member	222
Platform Event Subscriber Configuration	223
Pricing Action Parameters	224
Pricing Recipe	225
Process	226
Process Flow Migration	226
Product Attribute Set	227
Product Specification Type	228
Product Specification Record Type	229
Prompts (In-App Guidance)	230
Quick Action	230
Recommendation Strategy	231
Record Action Deployment	232
Record Alert Data Source Expression Set Definition	233
Record Type	234
RedirectWhitelistUrl	236
Referenced Dashboard	237

## Contents

Registered External Service	238
RelationshipGraphDefinition	239
Remote Site Setting	240
Report	241
Report Type	242
ServiceProcess	243
Slack App (Beta)	244
Service Catalog Category	245
Service Catalog Filter Criteria	246
Service Catalog Item Definition	247
Service Catalog Fulfillment Flow	248
Stationary Asset Environmental Source Record Type Configuration	249
Static Resource	250
Streaming App Data Connector	251
Sustainability UOM	252
Sustainability UOM Conversion	253
Timeline Object Definition	254
Timesheet Template	255
Translation	256
UI Object Relation Config	257
User Access Policy	258
Validation Rule	259
Vehicle Asset Emissions Source Record Type Configuration	260
View Definition (Beta)	262
Virtual Visit Config	263
Visualforce Component	264
Visualforce Page	265
Wave Analytic Asset Collection	266
Wave Application	267
Wave Component	268
Wave Dataflow	269
Wave Dashboard	270
Wave Dataset	271
Wave Lens	272
Wave Recipe	273
Wave Template Bundle	274
Wave Xmd	275
Web Store Template	276
Workflow Alert	277
Workflow Field Update	278
Workflow Knowledge Publish	279
Workflow Outbound Message	280
Workflow Rule	281
Workflow Task	282



## Contents

Behavior of Specific Metadata in Second-Generation Managed Packages . . . . .	284
Package Data Cloud Metadata Components . . . . .	285
Protected Components in Managed Packages . . . . .	286
Set Up a Platform Cache Partition with Provider Free Capacity . . . . .	286
Metadata Access in Apex Code . . . . .	287
Permission Sets and Profile Settings in Packages . . . . .	288
Protecting Your Intellectual Property . . . . .	292
Call Salesforce URLs Within a Package . . . . .	293
Namespace-Based Visibility for Apex Classes in Second-Generation Managed Packages . . . . .	295
Work with Services Outside of Salesforce . . . . .	296
Package Connected Apps in Second-Generation Managed Packaging . . . . .	297
Test and Respond to the New Order Save Behavior . . . . .	297
Develop Second-Generation Managed Packages . . . . .	299
Create a Second-Generation Managed Package . . . . .	300
View Package Details for a Second-Generation Managed Package . . . . .	300
Create Versions of a Second-Generation Managed Package . . . . .	301
Guidance for Package Version Numbering . . . . .	303
View Details about a Second-Generation Managed Package Version . . . . .	305
Project Configuration File for a Second-Generation Managed Package . . . . .	307
Get Ready to Promote and Release a Second-Generation Managed Package Version . . . . .	311
Specify a Package Ancestor in the Project File for a Second-Generation Managed Package . . . . .	311
Install and Uninstall Second-Generation Managed Packages . . . . .	313
Use the CLI to Install a Second-Generation Managed Package . . . . .	314
Use a URL to Install a Second-Generation Managed Package . . . . .	315
Install Notifications for Unauthorized Managed Packages . . . . .	316
Upgrade a Second-Generation Managed Package Version . . . . .	316
Resolve Apex Test Failures . . . . .	317
Run Apex on Package Install/Upgrade . . . . .	317
Customize Second-Generation Managed Package Installs and Uninstalls Using Scripts . . . . .	321
Sample Script for Installing Second-Generation Managed Packages with Dependencies . . . . .	322
Uninstall a Second-Generation Managed Package . . . . .	324
Prepare to Distribute Your Second-Generation Managed Package . . . . .	325
Code Coverage for Second-Generation Managed Packages . . . . .	325
Package Installation Key for Second-Generation Managed Packages . . . . .	326
Release a Second-Generation Managed Package . . . . .	326
Share Release Notes and Post-Install Instructions for Second-Generation Managed Packages . . . . .	327
Publishing Your App on AppExchange . . . . .	328
Push a Package Upgrade for Second-Generation Managed Packages . . . . .	329

## Contents

Schedule a Push Upgrade Using SOAP API for First- and Second-Generation Managed Packages	329
Assign Access to New and Changed Features in First- and Second-Generation Managed Packages	330
Sample Post Install Script for a Push Upgrade for First- and Second-Generation Managed Packages	330
Advanced Features for Second-Generation Managed Packages	332
Package Ancestors for Second-Generation Managed Packages	333
Patch Versions for Second-Generation Managed Packages	337
Create Dependencies Between Second-Generation Managed Packages	338
Considerations for Promoting Packages with Dependencies	340
Advanced Project Configuration Parameters for Second-Generation Managed Packages	342
Second-Generation Managed Packaging Keywords	345
Target a Specific Release for Your Second-Generation Managed Packages During Salesforce Release Transitions	346
Use Branches in Second-Generation Managed Packaging	347
Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages	348
Package IDs and Aliases for Second-Generation Managed Packages	349
Avoid Namespace Collisions in Second-Generation Managed Packages	350
Remove Metadata Components from Second-Generation Managed Packages	352
Delete a Second-Generation Managed Package or Package Version	356
Frequently Used Packaging Operations for Second-Generation Managed Packages	357
Transfer a Second-Generation Managed Package to a Different Dev Hub	357
Best Practices for Second-Generation Managed Packages	362
Manage Licenses for Managed Packages	362
Get Started with the License Management App	364
Lead and License Records in the License Management App	367
Modify a License Record	367
Refresh Licenses for a Managed Package	368
Extending the License Management App	368
Move the License Management App to Another Salesforce Org	371
Troubleshoot the License Management App	372
Best Practices for the License Management App	373
Troubleshoot Subscriber Issues	374
Manage Features in Second-Generation Managed Packages	376
Feature Parameter Metadata Types and Custom Objects	377
Set Up Feature Parameters	377
Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features	380
Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters	381
Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs	381
Best Practices for Feature Management	382
Considerations for Feature Management	382

## Contents

Get Started with AppExchange App Analytics . . . . .	382
App Analytics Use Cases . . . . .	384
Enable App Analytics on Your Second-Generation Managed Package . . . . .	387
Download Package Usage Logs, Package Usage Summaries, and Subscriber Snapshots . . . . .	387
Considerations for Custom Interactions . . . . .	388
AppExchange App Analytics Best Practices . . . . .	393
Package Usage Summaries . . . . .	410
Package Usage Logs . . . . .	412
Subscriber Snapshots . . . . .	437
Test Custom Integrations . . . . .	439
AppExchange App Analytics Developer Cookbook . . . . .	440
Gaps Between First-Generation and Second-Generation Managed Packaging . . . . .	460



# CHAPTER 1 Second-Generation Managed Packages

## In this chapter ...

- [What's a Second-Generation Managed Package?](#)
- [Set Up Your Development Environment](#)
- [Before You Create Second-Generation Managed Packages](#)
- [Scratch Orgs and Package Development](#)
- [Workflow for Second-Generation Managed Packages](#)
- [Components Available in Second-Generation Managed Packages](#)
- [Behavior of Specific Metadata in Second-Generation Managed Packages](#)
- [Develop Second-Generation Managed Packages](#)
- [Install and Uninstall Second-Generation Managed Packages](#)
- [Prepare to Distribute Your Second-Generation Managed Package](#)
- [Push a Package Upgrade for Second-Generation Managed Packages](#)
- [Advanced Features for Second-Generation Managed Packages](#)

Second-generation managed packaging (managed 2GP) ushers in a new way for AppExchange partners to develop, distribute, and manage their apps and metadata. You can use managed 2GP packaging to organize your source, build small modular packages, integrate with your version control system, and better utilize your custom Apex code. With version control being the source of truth, there are no packaging or patch orgs. You can execute all packaging operations via Salesforce CLI, or automate them using scripts. Submit second-generation managed packages for security review, and list them on AppExchange.

Use managed 2GP to create new managed packages. You can't currently migrate a first-generation managed package to a second-generation managed package.

Another great way to learn about second-generation managed packages, is by taking the [Second-Generation Managed Packages](#) Trailhead module.



**Note:** Second-generation managed packaging addresses the specific needs of AppExchange partners. If you're a customer or system integrator and you don't plan to distribute a package to multiple customers, unlocked packaging is the preferred tool. You can use unlocked packages to organize your existing metadata, package an app or extension, or package new metadata. See [Unlocked Packages](#) for more information.

## Second-Generation Managed Packages

- [Best Practices for Second-Generation Managed Packages](#)
- [Manage Licenses for Managed Packages](#)
- [Manage Features in Second-Generation Managed Packages](#)
- [Get Started with AppExchange App Analytics](#)
- [Gaps Between First-Generation and Second-Generation Managed Packaging](#)

## What's a Second-Generation Managed Package?

---

If your goal is to build an app and distribute it on AppExchange, you'll 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.

Each second-generation managed package follows a distinct lifecycle. As you develop your app, you add metadata to a package, and create a new package version. While the package is continually evolving, each package version is an immutable artifact.

A package version contains the set of metadata and features associated with the package version at the moment it was created. As you iterate on your package, and add, remove, or change the packaged metadata, you're likely to create many package versions along the way.

You can install a package version in a scratch, sandbox, trial, developer edition, or production org. Your customers can install the package into their org and when you release a new package version, your customers can upgrade to the latest version.

You can repeat the package development cycle any number of times. You can change metadata, create a package version, test the package version, and distribute it to your customers via AppExchange.

### [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.

### [Comparison of First- and Second-Generation Managed Packages](#)

If you're familiar with first-generation managed packages (managed 1GP) and wonder how it's different from second-generation managed packages (managed 2GP), here are some key distinctions.

## 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.



## Comparison of First- and Second-Generation Managed Packages

If you're familiar with first-generation managed packages (managed 1GP) and wonder how it's different from second-generation managed packages (managed 2GP), here are some key distinctions.

Managed 1GP Packages	Managed 2GP Packages
The packaging org is the source of truth for the metadata in your package.	Your version control system is the source of truth (source-driven system) for the metadata in your package.  And unlike managed 1GP, managed 2GP doesn't use packaging or patch orgs.
The packaging org owns the package. The metadata in the package resides in the packaging org.	The Dev Hub owns the package, but the Dev Hub doesn't contain the packaged metadata.  We recommend that you enable Dev Hub in your Partner Business Org (PBO).
A packaging org can own only one managed package.	A Dev Hub can own one or more packages.
The namespace of the managed package is created in the packaging org.	The namespace of a managed package is created in a namespace org and linked to the Dev Hub. And you can associate multiple namespaces to a single Dev Hub.  A namespace is linked to a managed 2GP when you run the <code>sf package create</code> Salesforce CLI command. And you must specify the namespace in the <code>sfdx-project.json</code> file.  See <a href="#">Namespaces for Second-Generation Managed Packages</a> for more details.
A namespace can be associated with only one package.	Multiple packages can use the same namespace.
Global Apex is the only way to share code across packages.	Multiple packages using the same namespace can share code using public Apex classes and methods with the <code>@namespaceAccessible</code> annotation.
Some packaging operations, like package create and package uninstall, can't be automated.	All packaging operations can be automated using Salesforce CLI.
Package versioning is linear.	Package versioning is flexible, and you can abandon unwanted package versions. This flexible versioning allows you to use branches and do parallel package development.
Patch versions can only be created in specialized orgs called patch orgs.	Patch versions are created using Salesforce CLI. The version control system is the source of truth, and there are no patch orgs.

Despite these distinctions, managed 1GP and 2GP packages have many things in common. They share the key packaging concept of associating metadata with a package. And they both allow you to iterate and create package and patch versions, which can be installed and uninstalled in subscriber orgs. Both managed package types enable you to submit a package for AppExchange security review, and list your package on AppExchange. And both managed package types can use the License Management App, Subscriber Support Console, and Feature Management App.

## Set Up Your Development Environment

---

Second-generation managed packaging uses Salesforce DX developer tools. Ensure that you have the required tools and orgs installed and enabled.

You use these tools for managed 2GP package development.

- [Salesforce CLI](#), a rich set of commands to execute different packaging operations like package creation and package install
- A source control system of your choosing
- A Dev Hub org
- [Salesforce Extension for Visual Studio Code](#) (optional), an IDE designed to facilitate the development of Salesforce components

## Use the Dev Hub to Keep Track of Package Development

Your Dev Hub is the designated place to find and manage all your managed 2GP packages, scratch orgs, and namespaces. After you enable the Dev Hub setting on a Salesforce org, that Dev Hub becomes the owner of every managed 2GP package you create.

All Salesforce ISV and OEM partners should designate their Partner Business Org as their Dev Hub org. A Partner Business Org (PBO) is the production org where Salesforce Partners run their business.

### [Enable Dev Hub and Second-Generation Managed Packaging](#)

The Dev Hub lets you create and manage second-generation managed packages and scratch orgs. Your Dev Hub is the designated place to find and manage all your managed 2GP packages, scratch orgs, and namespaces.

### [Limited Access License for Package Developers](#)

The Salesforce Limited Access - Free is designed for users whose role is to build customizations or applications. This license provides access to the Dev Hub, development tools, and environments. In the production org, this license restricts access to standard and custom objects. Partner Business Orgs (PBO) include 100 Salesforce Limited Access - Free user licenses.

### [Add a Limited Access User to Your Dev Hub Org](#)

Provide your developers access to the Dev Hub and Salesforce DX development tools by adding a user with Salesforce Limited Access - Free license and the Limited Access user profile in your Dev Hub org. Then create and assign them a permission set to the required Dev Hub objects.

### [Assign Second-Generation Managed Packaging User Permissions](#)

To create second-generation managed packages and scratch orgs, developers require access to the Dev Hub org. We recommend enabling the Dev Hub in your Partner Business Org (PBO). A Salesforce admin can create a permission set to grant appropriate permissions to the required Dev Hub objects and system permission.

## Enable Dev Hub and Second-Generation Managed Packaging

The Dev Hub lets you create and manage second-generation managed packages and scratch orgs. Your Dev Hub is the designated place to find and manage all your managed 2GP packages, scratch orgs, and namespaces.

After you enable the Dev Hub setting on a Salesforce org, that Dev Hub becomes the owner of every managed 2GP package you create. All Salesforce ISV and OEM partners should designate their Partner Business Org (PBO) as their Dev Hub org.

To enable Dev Hub:

1. Log in to your Partner Business Org.

### EDITIONS

Available in: Salesforce Classic and Lightning Experience

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


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. Select **Enable Dev Hub**. After you enable Dev Hub, you can't disable it.
4. Select **Enable Unlocked Packages and Second-Generation Managed Packages**. After you enable this setting, you can't disable it.

If you choose to use a trial or Developer Edition org as your Dev Hub, consider these factors.

- You're limited to creating 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.
- If a package is associated with a non-production Dev Hub org, and that org expires or becomes inactive, the installed package can't be updated, and new attempts to install the package may fail.
- 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.

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.

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

## Limited Access License for Package Developers

The Salesforce Limited Access - Free is designed for users whose role is to build customizations or applications. This license provides access to the Dev Hub, development tools, and environments. In the production org, this license restricts access to standard and custom objects. Partner Business Orgs (PBO) include 100 Salesforce Limited Access - Free user licenses.

If the Salesforce Limited Access - Free license isn't already enabled in your PBO, log a case with [Salesforce Partner Support](#) to request up to 100 licenses. A Salesforce admin can upgrade a Salesforce Limited Access - Free license to a standard Salesforce license at any time.

Certain developer features aren't available with the Salesforce Limited Access - Free license.

- To provide the ability to create and manage org shapes, assign the Salesforce user license. The Salesforce Limited Access - Free license isn't supported at this time.
- Users with the Salesforce Limited Access - Free license and View All permissions can create scratch orgs using an existing org shape.
- Users with the Salesforce Limited Access - Free license and View All permissions can view scratch org snapshots created by users other than themselves.
- 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.

If your developers need broader access, consider assigning the Salesforce license. For details, see [Standard User Licenses](#) in *Salesforce Help*.

## Add a Limited Access User to Your Dev Hub Org

Provide your developers access to the Dev Hub and Salesforce DX development tools by adding a user with Salesforce Limited Access - Free license and the Limited Access user profile in your Dev Hub org. Then create and assign them a permission set to the required Dev Hub objects.

The Salesforce Limited Access - Free is designed for users whose role is to build customizations or applications. This license provides access to the Dev Hub, development tools, and environments. In the production org, this license restricts access to standard and custom objects.

1. Create a user in your Dev Hub org.
  - a. In Setup, enter `users` in the Quick Find box, then select **Users**.
  - b. Click **New User**.
  - c. Fill out the form.
  - d. Select **Salesforce Limited Access - Free** for User License, and then **Limited Access User** for Profile.
  - e. After filling out the remaining information, click **Save**.
2. Create a permission set that provides your developer users with access to the required Dev Hub objects. For details, see [Create and Assign a Permission Set for Developer Users](#) or [Assign Second-Generation Managed Packaging User Permissions](#).

## Assign Second-Generation Managed Packaging User Permissions

To create second-generation managed packages and scratch orgs, developers require access to the Dev Hub org. We recommend enabling the Dev Hub in your Partner Business Org (PBO). A Salesforce admin can create a permission set to grant appropriate permissions to the required Dev Hub objects and system permission.

To give developers access to the Dev Hub org, create a permission set that contains these required 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 second-generation managed packages and package versions, the permission set must also contain:

- System Permissions > Create and Update Second-Generation Packages

This permission provides access to:

Salesforce CLI Command	Tooling API Object (Create and Edit)
<code>sf package create</code>	Package2
<code>sf package version create</code>	Package2VersionCreateRequest
<code>sf package version update</code>	Package2Version

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.

## Before You Create Second-Generation Managed Packages

---

When you use second-generation managed packaging, to be sure that you set it up correctly, verify the following.

Did you?

- [Enable Dev Hub and Second-Generation Managed Packaging](#) in your Partner Business Org (PBO)
- Install [Salesforce CLI](#)
- [Create and Register Your Namespace for Second-Generation Managed Packages](#)

Developers who work with managed 2GP packages need a user license and permission set that provides access to the Dev Hub org. See [Limited Access License for Package Developers](#) and [Assign Second-Generation Managed Packaging User Permissions](#).

### [Know Your Orgs for Second-Generation Managed Packages](#)

Some of the orgs that you use with second-generation managed packaging (managed 2GP) have a unique purpose.

#### [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.

#### [Namespaces for Second-Generation Managed Packages](#)

A namespace is a 1–15 character alphanumeric identifier that distinguishes your package and its contents from other packages in your customer's org. A namespace is assigned to a second-generation managed package (managed 2GP) at the time that it's created, and can't be changed.

#### [Create and Register Your Namespace for Second-Generation Managed Packages](#)

With second-generation managed packaging (managed 2GP), you can share a single namespace with multiple packages. Since sharing of code is much easier if your package shares the same namespace, we recommend that you use a single namespace for all of your managed 2GP packages.

#### [Key Concepts in Second-Generation Managed Packaging](#)

Let's look at some key high-level concepts in second-generation managed packaging (managed 2GP).

#### [How Manageability Rules and Ancestry Impact Upgrades for Second-Generation Managed Packages](#)

Before you dive in and create your first second-generation managed package (managed 2GP), it's important to understand these concepts, and how they affect each other.

#### [Which Package Types Can Your Package Depend On?](#)

Both second-generation managed packaging (managed 2GP) and unlocked packaging let you easily develop small interdependent packages and share logic between them. If you design your app to rely on small modular packages, both package creation and package installation are faster, and you're less likely to hit limits.

## Know Your Orgs for Second-Generation Managed Packages


Some of the orgs that you use with second-generation managed packaging (managed 2GP) have a unique purpose.

### Choose Your Dev Hub Org

Use the Dev Hub org for these purposes.

- As owner of all second-generation managed packages
- To link your namespaces
- To authorize and run your `sf package` Salesforce CLI commands

We recommend that your Partner Business Org is also your Dev Hub org.

 **Note:** The Dev Hub org against which you run the `sf package create` command becomes the owner of the package.

If the Dev Hub org expires or is deleted, packages owned by that Dev Hub:

- Can't be transferred to a different Dev Hub
- Stop working and new package versions can't be created

## Namespace Org

The primary purpose of the namespace org is to acquire a namespace for your managed 2GP package.

After you create a namespace org and specify the namespace in it, open the Dev Hub org and link the namespace org to the Dev Hub org.

## Other Orgs

When you work with managed 2GP packages, you also use these orgs:

- Scratch orgs to develop and test your packages.
- A target or installation org in which you install the package.

SEE ALSO:

[Link a Namespace to a Dev Hub Org](#)

[Scratch Org Allocations for Partners](#)


[Salesforce DX Developer Guide: Scratch Orgs](#)

## 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.

## Namespaces for Second-Generation Managed Packages

A namespace is a 1–15 character alphanumeric identifier that distinguishes your package and its contents from other packages in your customer's org. A namespace is assigned to a second-generation managed package (managed 2GP) at the time that it's created, and can't be changed.

When you specify a package namespace, every component added to a package has the namespace prefixed to the component API name. Let's say you have a custom object called `Insurance_Agent` with the API name, `Insurance_Agent__c`. If you add this component to a package associated with the Acme namespace, the API name becomes `Acme__Insurance_Agent__c`.

 **Important:** 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).

When you work with namespaces, keep these considerations in mind.

- You can develop more than one managed 2GP package and associate the packages with the same namespace. But a single managed 2GP package can't be associated with more than one namespace.
- If you work with more than one namespace, we recommend that you set up one project for each namespace.
- It's beneficial for managed 2GP packages to share the same namespace, but it's not required. Carefully consider your package and namespace strategy. After a namespace is associated with a managed 2GP, the association can't be changed.
- There are scenarios where you may prefer to keep a managed 2GP package isolated from other managed 2GP packages you're developing. For example, if you're developing a product that you intend to sell or spin off, having a unique namespace for that package enables you to transfer the namespace with the package.

SEE ALSO:

[Create and Register Your Namespace for Second-Generation Managed Packages](#)

[Link a Namespace to a Dev Hub Org](#)

[Avoid Namespace Collisions in Second-Generation Managed Packages](#)

## Create and Register Your Namespace for Second-Generation Managed Packages

With second-generation managed packaging (managed 2GP), you can share a single namespace with multiple packages. Since sharing of code is much easier if your package shares the same namespace, we recommend that you use a single namespace for all of your managed 2GP packages.

To create a namespace:

1. Sign up for a new Developer Edition org.
2. In Setup, enter *Package Manager* in the Quick Find box, and select **Package Manager**.
3. In Namespace Settings, click **Edit**.
4. Enter a namespace and select **Check Availability**.
5. (Optional) Select a package to associate with this namespace, or select **None**, then click **Review**.
6. Review your selections, and then click **Save**.

To register a namespace:

1. To link the namespace that you created with your Dev Hub, use Namespace Registry. See [Link a Namespace to a Dev Hub Org](#) for details.
2. In the `sfdx-project.json` file, specify your namespace using the namespace attribute. When you create a new 2GP package, the package is associated with the namespace specified in the `sfdx-project.json` file.

SEE ALSO:

[Namespaces for Second-Generation Managed Packages](#)

[Link a Namespace to a Dev Hub Org](#)

[Avoid Namespace Collisions in Second-Generation Managed Packages](#)

## Key Concepts in Second-Generation Managed Packaging

Let's look at some key high-level concepts in second-generation managed packaging (managed 2GP).

What's the difference between...	Details
An app, a package, and metadata?	<p>An app is a set of features that you're developing for your customers.</p> <p>Metadata is the technical representation of Salesforce features like custom objects, Apex classes, and Lightning pages. An app is composed of a set of metadata.</p> <p>A package is the container for your app's Salesforce metadata. Packages are used to distribute the app that you build. When a package is installed in an org, the app's metadata is deployed to the org.</p>
A package and package version?	<p>Your app, and thus your package, evolves over time. Whenever you change, add, or remove the metadata in your package, you create a new package version. Each package version is an immutable artifact, a static snapshot of your metadata at a specific point in time. So while your package evolves continuously, you take snapshots of it when it's in a stable state in the form of a package version. Technically speaking, when we say "Install a package," we really mean install a specific package version.</p>
A package install and package upgrade?	<p>A package install refers to the first time a version of the package is installed in an org. When a package is installed, the metadata associated with the package is deployed into the org.</p> <p>A package upgrade refers to the installation of a new package version in an org that already has a previous version of the package installed. During a package upgrade, metadata changes are deployed. An upgrade can include deploying new metadata, modifying existing metadata, or deleting or deprecating metadata. At any given point in time, an org can only ever have one version of a package installed in that org.</p>



Is it possible to...	Details
Push a package upgrade?	Yes. Push upgrades enable you to upgrade packages installed in subscriber orgs, without asking customers to install the upgrade themselves. For more details, see <a href="#">Push a Package Upgrade</a> .
Uninstall a package?	Yes. When you uninstall managed 2GP packages, all components in the package and any associated data is deleted from the org. Before uninstalling a package, review these <a href="#">considerations</a> .
Delete a package or package version?	Yes. If you haven't promoted or distributed a specific package or package version, you can delete the package or package version from your Dev Hub org. For more details, see <a href="#">Delete a Managed 2GP Package or Package Version</a> .

## How Manageability Rules and Ancestry Impact Upgrades for Second-Generation Managed Packages

Before you dive in and create your first second-generation managed package (managed 2GP), it's important to understand these concepts, and how they affect each other.

- [Manageability Rules](#)
- [Package Ancestry](#)
- [Package Upgrades](#)

### Manageability Rules

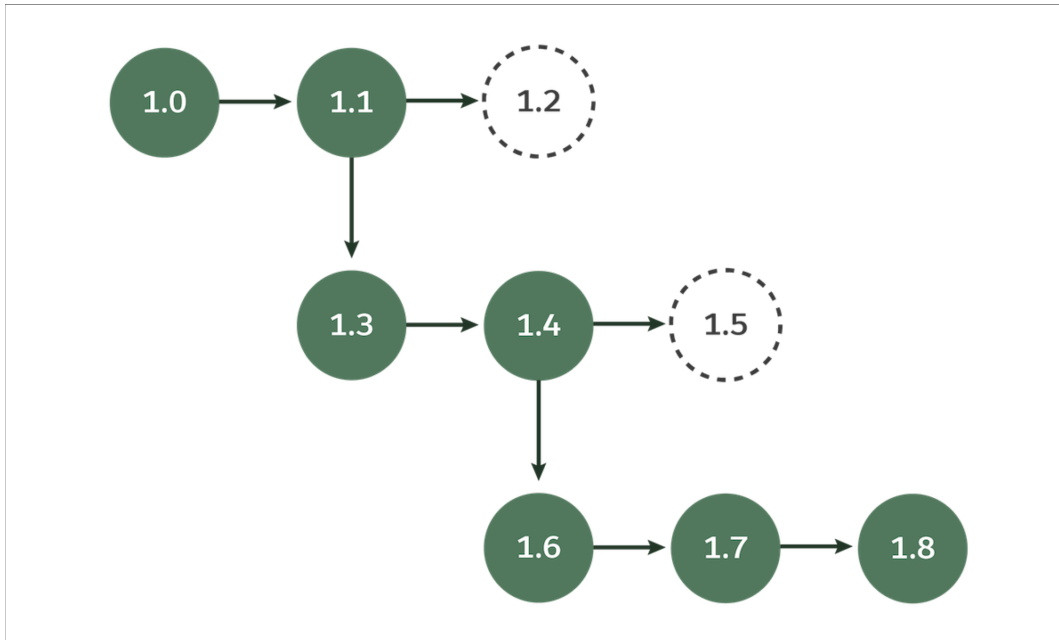
Each metadata component that you include in a managed 2GP package has certain rules that determine its behavior in a subscriber org. Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is installed in a subscriber's org.

Manageability rules apply at both the component level and at the component attribute level. For example, manageability rules determine whether you or the subscriber can delete a custom field, and more specifically whether either of you can edit the Field Label, Default Value, or other attributes of the custom field. For all first- and second-generation managed packages, we enforce manageability rules during package version creation. If you attempt to make a change that would break a manageability rule for one of the metadata components in your package, your package version creation fails.

### Package Ancestry

Second-generation managed packaging offers a flexible linear package versioning model by letting you break your linear versioning and abandon a package version you no longer want to build upon. We call these versioning decisions package ancestry. When you create a package version, you must also specify which package version is the ancestor.

In this quick glance at a package ancestry tree, version 1.2 and 1.5 have been abandoned. To dig deeper into this topic, see [Package Ancestors](#).



### How Manageability Rules and Ancestry Impact Package Upgrades

Both manageability rules and package ancestry impact package upgrades. During package upgrade we enforce the manageability rule for each new and changed component in your package version. Depending on what you changed when you created the new package version, some metadata is added to the org during package upgrade, other metadata is modified or deleted, and some changes aren't applied at all.

For example, page layouts don't get updated during package upgrade, so if you change a page layout, only new customers receive your modified page layout. When existing subscribers upgrade their package, they won't receive that change. Conversely, changes to Apex code or the formula in a formula field are updated during a package upgrade.

Package ancestry determines the package upgrade path. This is a complex topic, and we have topics that go deeper into this subject. At a high level the package version you designate as the ancestor determines whether a subscriber can upgrade to that version. Subscribers can upgrade from one package version to another only if the ancestry tree is followed. To learn more, see [Understanding Package Upgrades with Ancestry](#).

#### SEE ALSO:

[Package Ancestors for Second-Generation Managed Packages](#)

[Understanding Package Upgrades with Ancestry](#)

## Which Package Types Can Your Package Depend On?

Both second-generation managed packaging (managed 2GP) and unlocked packaging let you easily develop small interdependent packages and share logic between them. If you design your app to rely on small modular packages, both package creation and package installation are faster, and you're less likely to hit limits.

To develop small, modular packages, you create dependencies between your packages. A package dependency is when metadata contained in one package depends on metadata contained in another package. These dependencies allow you to extend the functionality of the base package with components and metadata in a separate extension package.

When working with packaging, only certain combinations of packages are supported.

	Can a Managed 1GP Depend on a ...	Can a Managed 2GP Depend on a ...	Can an Unlocked Package Depend on a ...	Can an Unmanaged Package Depend on a ...
Managed 1GP	Yes	Yes	Yes	No
Managed 2GP	No <sup>1</sup>	Yes	Yes	No
Unlocked package	Not recommended	Not recommended	Yes	No
Unmanaged package	Not recommended	Not recommended	Not recommended	No

<sup>1</sup>This dependency isn't supported, 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.

#### SEE ALSO:

[Create Dependencies Between Second-Generation Managed Packages](#)

[Considerations for Promoting Packages with Dependencies](#)

## Scratch Orgs and Package Development

Scratch orgs are temporary Salesforce orgs intended for development and automation. They enable source-driven deployments of Salesforce code and metadata. A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with various features and preferences.

You can use a scratch org to develop the app you want to package, and you can also create scratch orgs to test out your package. Scratch orgs also help with continuous integration (CI) processes to automate package development steps. For example, you could write a script that creates a package version, creates a scratch org, installs the package version into the scratch org, runs Apex tests, and emails the test results to the release manager.

#### EDITIONS

Available in: Lightning Experience

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

## Enable Data Cloud for Scratch Orgs

To use Data Cloud components in scratch orgs or to add these components to a package, Data Cloud for Scratch Orgs must be enabled. Log a case with [Salesforce Partner Support](#) and request that Data Cloud for Scratch Orgs be enabled on your Partner Business Org. Data Cloud for Scratch Orgs is only available to scratch orgs associated with the Dev Hub in your Partner Business Org.

#### [How Scratch Orgs Fit in the Package Development Workflow](#)

Scratch orgs are an essential tool in both developing and testing the app you want to package. Scratch orgs also help with continuous integration (CI) processes to automate package development steps. For example, you could write a script that creates a package version, creates a scratch org, installs the package version into the scratch org, runs Apex tests, and emails the test results to the release manager.

#### [Scratch Org Definition Files vs Org Shape in Package Development](#)

The scratch org definition file is used when you create scratch orgs, and also when you create new package versions. The scratch org definition file is a blueprint for your scratch org and defines the shape of the org you want for your package development work.

### [When to Use Scratch Org Snapshots in Package Development](#)

If the managed 2GP or unlocked package that you're building depends on one or more large packages, it can take a long time for the package version creation CLI command to complete. Let's talk about why that occurs, and how scratch org snapshots can dramatically reduce how long it takes to create a new package version.

### [Create a Package Version Based on a Scratch Org Snapshot](#)

If the dependent package your base package requires is stable, you can reduce the end-to-end package version creation time by creating a scratch org snapshot.

### [Get Access to Scratch Orgs That Have Agentforce](#)

Agentforce is a set of tools to create and customize AI agents that are deeply and securely integrated with customers' data and apps. Agentforce brings together humans with agents to transform the way work gets done. Start your journey with Agentforce by testing it in a scratch org.

### [Scratch Org Allocations for Salesforce Partners](#)

To ensure optimal performance, Salesforce partners are allocated a set number of scratch orgs in their Partner Business Org (PBO). These allocations determine how many scratch orgs you can create daily, and how many can be active at a given point.

### [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.

### [Supported Scratch Org Editions for Partners](#)

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

## How Scratch Orgs Fit in the Package Development Workflow

Scratch orgs are an essential tool in both developing and testing the app you want to package. Scratch orgs also help with continuous integration (CI) processes to automate package development steps. For example, you could write a script that creates a package version, creates a scratch org, installs the package version into the scratch org, runs Apex tests, and emails the test results to the release manager.

### Develop Your Package in a Scratch Org

When developing a package, it's preferable to use a namespaced scratch org. A namespaced scratch org prepends scratch org metadata with the package namespace. This is true for both metadata you create in the scratch org, and any metadata you deploy to the scratch org.

To create a namespaced scratch org, use your Dev Hub org to create the scratch org. Before you create the scratch org:

- Ensure that the namespace you plan to use is already [associated with your Dev Hub org](#).
- Specify the namespace in your `sfdx-project.json` file.
- Create a [scratch org definition file](#) and include any features, settings, or limits that your org needs.

When you create a scratch org both the namespace and ancestry information listed in `sfdx-project.json` file are pulled into the scratch org. The ancestry information, specified as `ancestorId` or `ancestorVersion` in your `sfdx-project.json` file, seeds the scratch org with manageability rules, and later warns you if you attempt to change metadata in a way that's incompatible with the specified ancestor version. This way, you learn of issues during development instead of during the creation of the next package version.

To create a namespaced scratch org that includes ancestor information in the scratch org, run this CLI command.

```
sf org create scratch --target-dev-hub MyHub --definition-file
config/project-scratch-def.json
```

If you don't want the ancestor and manageability rules seeded into the scratch org, include the `--no-ancestors` flag when you create the scratch org.

When you are ready to create a new package version, see [Create Versions of a Second-Generation Managed Package](#).

## Test Your Package in a Scratch Org

When testing your package, create a scratch org that doesn't have a namespace. Use the `--no-namespace` parameter when you create the scratch org.

```
sf org create scratch --definition-file config/project-scratch-def.json --no-namespace --no-ancestors
```

After you create the scratch org, [install the package](#) into the scratch org, and begin testing.

## Enable Data Cloud for Scratch Orgs

To use Data Cloud components in scratch orgs or to add these components to a package, Data Cloud for Scratch Orgs must be enabled. Log a case with [Salesforce Partner Support](#) and request that Data Cloud for Scratch Orgs be enabled on your Partner Business Org. Data Cloud for Scratch Orgs is only available to scratch orgs associated with the Dev Hub in your Partner Business Org.

SEE ALSO:

[Salesforce DX Developer Guide: Create Scratch Orgs](#)

[Salesforce CLI Command Reference: org create scratch](#)

[Salesforce DX Developer Guide: Select the Salesforce Release for a Scratch Org](#)

## Scratch Org Definition Files vs Org Shape in Package Development

The scratch org definition file is used when you create scratch orgs, and also when you create new package versions. The scratch org definition file is a blueprint for your scratch org and defines the shape of the org you want for your package development work.

## Build Your Own Scratch Definition File

If you read [How Scratch Orgs Fit in the Package Development Workflow](#) on page 16 you might recall that the CLI command for creating scratch orgs includes a flag called `--definition-file`.

```
sf org create scratch --target-dev-hub MyHub --definition-file config/project-scratch-def.json
```

In this example, `project-scratch-def.json` is the scratch org definition file. To learn more about what can be specified in this definition file, see [Build Your Own Scratch Org Definition File](#) in the *Salesforce DX Developer Guide*.

Similarly the CLI `--definition-file` flag can be used when creating a new package version.


```
sf package version create --package "Expenser App" --definition-file config/project-scratch-def.json --code-coverage
```

When used in the `package version create` command, the scratch org definition file is used to specify the features, settings, or limits that your package requires.

## When to Use Org Shape

If you're developing managed packages to distribute on AppExchange, we expect that you know what features and settings your packages depends on, and expect you to specify these requirements in a scratch org definition file. But there are scenarios like unlocked packages, or if you're moving from 1GP to 2GP package development, where using [Org Shape for Scratch Orgs](#) can be useful.

During org shape creation, we capture the features, settings, edition, licenses, and limits of the specified source org. By using org shape you don't have to manually list these items in the scratch org definition file.

 **Note:** The source org you use for org shape can't be a sandbox or scratch org.

Later when you create a package version, specify the org ID for the source org in the scratch org definition file.

```
{
  "orgName": "Acme",
  "sourceOrg": "00DB1230400Ifx5"
}
```

For more detailed instructions on enabling and creating org shape, review [Create a Scratch Org Based on an Org Shape](#) in the *Salesforce DX Developer Guide*.

If you're moving from managed 1GP package development to 2GP package development, creating an org shape of your 1GP packaging org could be useful as you begin 2GP package development. Creating an org shape of your 1GP packaging org ensures that the features required for your package metadata are specified.

### SEE ALSO:

[How Scratch Orgs Fit in the Package Development Workflow](#)

[Salesforce DX Developer Guide: Build Your Own Scratch Org Definition File](#)

[Salesforce DX Developer Guide: Create a Scratch Org Based on an Org Shape](#)

[Salesforce DX Developer Guide: Create a Scratch Org Based on an Org Shape](#)

## When to Use Scratch Org Snapshots in Package Development

If the managed 2GP or unlocked package that you're building depends on one or more large packages, it can take a long time for the package version creation CLI command to complete. Let's talk about why that occurs, and how scratch org snapshots can dramatically reduce how long it takes to create a new package version.

When you run the `package version create` CLI command, we create a scratch org behind the scenes. That scratch org serves as a build org where we build your package. In the build org we install the dependent packages you specified, and deploy the package metadata for the package version you're creating. If your dependent packages are large, the package install time for those dependent packages extends the overall package creation time.

If the dependent packages that your base package requires are stable, you can reduce the end-to-end package version creation time by creating a scratch org snapshot, and using that scratch org snapshot during package version creation.

A [scratch org snapshot](#) captures the state of a scratch org's configuration so that you can use the snapshot to create scratch org replicas. A snapshot is a point-in-time copy of a scratch org that includes installed packages, features, limits, licenses, metadata, and data. If you install your dependent packages in the scratch org before you create the snapshot, and you specify the snapshot when you create a new package version, the package build process bypasses these steps. Meaning, we don't install the dependent packages into the build org, we use the snapshot instead. By not installing the dependent packages during package version creation, your package version builds in a fraction of the time.

Keep in mind, the intention of scratch org snapshots in the package development cycle is to shorten the package creation time during development. When you are ready to promote and release a package, you must create a new package version that doesn't rely on a scratch org snapshot. Package versions created using scratch org snapshots can't be promoted to the released state.

SEE ALSO:

[Create a Package Version Based on a Scratch Org Snapshot](#)

[Salesforce DX Developer Guide: Scratch Org Snapshots](#)

## Create a Package Version Based on a Scratch Org Snapshot

If the dependent package your base package requires is stable, you can reduce the end-to-end package version creation time by creating a scratch org snapshot.

If you haven't reviewed [When to Use Scratch Org Snapshots in Package Development](#) on page 18, review that topic before continuing.

There's more than one workflow you can follow when creating a package version based on a scratch org snapshot. You can start by creating a scratch org, you can build your own scratch org definition file, or you can choose to use org shape to create a new scratch org. Whichever path you choose, after the scratch org is created, you install all the dependent packages into it, and then take a snapshot of the scratch org.

Sample Workflow

This workflow uses an org shape to create the initial scratch org where you'll install the stable dependent packages, and then create a scratch org snapshot to create a package version.

1. Create the org shape.

```
sf org create shape --target-org source-org1
```

2. Create a scratch org definition file that indicates the shape's source org.

```
{
  "orgName": "Salesforce",
  "sourceOrg": "00DB1230400Ifx5"
}
```

3. Create a scratch org using the org shape.

```
sf org create scratch --duration-days 30 --no-namespace --no-ancestors --definition-file
config/scratch-def-with-shape-id.json --alias dev1-with-shape
```

If your default Dev Hub org isn't the one that owns the org shape, indicate it on the command line.

4. Install the dependent packages.

```
sf package install --package 04txx --target-org dev1-with-shape
```

5. Create a snapshot of the scratch org.

```
sf org create snapshot --name dhsnapshot --source-org dev1-with-shape --target-dev-hub
dev-hub
```

6. Create a new scratch org definition file and specify the snapshot name, then save the file.

```
{
  "orgName": "Salesforce",
```

```
"snapshot": "dhsnapshot"
}
```

7. Create a package version using the org snapshot. This command is specifying the scratch org definition file that contains the snapshot information in it.

```
sf package version create --package hc-ext1 --code-coverage --installation-key-bypass
--async-validation --definition-file
scratch-def-with-snapshot-id.json
```

## SEE ALSO:

[When to Use Scratch Org Snapshots in Package Development](#)

[Salesforce DX Developer Guide: Create Org Shapes](#)

## Get Access to Scratch Orgs That Have Agentforce

Agentforce is a set of tools to create and customize AI agents that are deeply and securely integrated with customers' data and apps. Agentforce brings together humans with agents to transform the way work gets done. Start your journey with Agentforce by testing it in a scratch org.

If you don't already have a Partner Business Org (PBO), join the [Salesforce Partner Community](#) and [request a PBO](#).

If you're new to creating scratch orgs, follow these steps to complete the one-time Dev Hub setup in your PBO. The Dev Hub is a feature within an org that lets you create and manage scratch orgs, second-generation managed packages (2GP), and namespaces.

- [Enable the Dev Hub and 2GP](#)
- [Create a Developer Edition](#) org using Environment Hub
- [Create a namespace](#) in the Developer Edition org
- [Link that namespace](#) from your PBO. Linking the namespace lets you create 2GP packages that use that namespace.

As of September 16, 2024, all active Partner PBOs can create scratch orgs with Agentforce and Prompt Builder. Agentforce and Prompt Builder can be enabled for scratch orgs that are associated with a Dev Hub in a PBO.

Partners with active PBO orgs	Enabled as of September 16, 2024
New PBO orgs (Trial or Active)	Automatically enabled when created

To create a scratch org with Agentforce and Prompt Builder enabled, use this sample `project-scratch-def.json` file (or simply add the feature and setting shown in this sample to your existing scratch org definition file).

```
{
  "orgName": "GenAI Scratch Org",
  "edition": "Partner Developer",
  "features": ["Einstein1AIPlatform"],
  "settings": {
    "einsteinGptSettings" : {
      "enableEinsteinGptPlatform" : true
    }
  }
}
```



To create a scratch org with the Einstein1AIPlatform feature, the scratch org you create must be a Partner Developer edition.


To create a scratch org, run this Salesforce CLI command. Update the definition-file name, alias, and target-dev-hub alias as needed.

```
sf org create scratch --definition-file config/my-agentforce-project-scratch-def.json
--alias MyScratchOrg --set-default --target-dev-hub MyHub
```

## Scratch Orgs with both Agentforce and Data Cloud

For some use cases such as prompt templates that use RAG, Retrievers, and BYO LLM, a scratch org that has both GenAI and Data Cloud functionality enabled is required.

Only include Data Cloud if it's required. Specifying Data Cloud in a scratch org significantly increases the time it takes for a scratch org creation to complete.

 **Note:** Including Data Cloud in a scratch org has a prerequisite. You must first open a case in the Salesforce Partner Community to request for your PBO Dev Hub org to be granted permission to create Data Cloud scratch orgs. This request is only granted to PBO orgs.

```
{
  "orgName": "GenAI & Data Cloud Scratch Org",
  "edition": "Partner Developer",
  "features": ["CustomerDataPlatform", "CustomerDataPlatformLite", "Einstein1AIPlatform"],

  "settings": {
    "einsteinGptSettings" : {
      "enableEinsteinGptPlatform" : true
    }
    "customerDataPlatformSettings": {
      "enableCustomerDataPlatform": true
    }
  }
}
```

## Set up Agentforce in your Scratch Org

After your scratch org is created, follow these steps to start developing with Agentforce.

- [Enable Agents and Einstein Copilot for Salesforce](#) manually in the scratch org.
- [Assign user permissions](#).
- To use prompt templates with your Agent Actions, [assign prompt template permissions](#).

### SEE ALSO:

[Trailhead: Quick Start: Build Your First Agent with Agentforce](#)

[Salesforce Help: Agentforce: Agents and Copilot](#)

[Salesforce Help: The Building Blocks of Agents and Copilot](#)

[Salesforce Help: Customize Your Agents and Copilot with Topics and Actions](#)

[Salesforce Help: Considerations for Agents and Copilot](#)

[Salesforce Help: AI Project Success](#)

## Scratch Org Allocations for Salesforce Partners

To ensure optimal performance, Salesforce partners are allocated a set number of scratch orgs in their Partner Business Org (PBO). 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.

### Active PBOs

- 150 active
- 300 daily

### Trial PBOs

- 20 active
- 40 daily

## Scratch Org Snapshot Allocations

The number of snapshots you can create (active and daily) is the same as the active scratch org allocation.

## Package Version Creation Limits

The maximum number of package versions you can create per day is equal to the daily allocated scratch orgs. For example, if you're allocated 300 daily scratch orgs, you're also allowed to create 300 package versions per day.

If you specify `--skipvalidation` when creating a package version, the maximum number of package versions you can create using `skip validation` is 500 per day.

## 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.

## 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](#).

## Workflow for Second-Generation Managed Packages

You can create and install a second-generation managed package (managed 2GP) directly from the command line.

Review and complete the steps in [Before You Create Second-Generation Managed Packages](#) before starting this workflow.

The basic managed 2GP workflow includes these steps. See specific topics for details about each step.

1. Create a DX project.

```
sf project generate --output-dir expense-manager-workspace --name expenser-app
```

2. Authorize the Dev Hub org.

```
sf org login web --set-default-dev-hub
```

When you perform this step, include the `---set-default-dev-hub` option. You can then omit the Dev Hub username when running subsequent Salesforce CLI commands.



**Tip:** If you define an alias for each org you work with, it's easy to switch between different orgs from the command line. You can authorize different orgs as you iterate through the package development cycle.

3. Create a scratch org and develop the app you want to package. You can use VS Code and the Setup UI in the scratch org to build and retrieve the pieces you want to include in your package. Navigate to the `expenser-app` directory, and then run this command.

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

4. Verify that all package components are in the project directory where you want to create a package. If you're trying out the exact steps and commands in this workflow, you must add at least one piece of metadata before you continue to the next step.

- In the `sfdx-project.json` file, specify a namespace using the `namespace` attribute. For example: `"namespace": "exp-mgr"`  
If you specified a namespace when you created a Salesforce DX project in step one, you can skip this step. Before adding a namespace, make sure that you've linked the [namespace](#) to your Dev Hub org.

- From the Salesforce DX project directory, create the package.

```
sf package create --name "Expense Manager" --path force-app --package-type Managed
```

Your new managed 2GP package has the namespace you specified in the `sfdx-project.json` file.

**Important:** After you create a package, you can't change or add a namespace, or change the Dev Hub the package is associated with.

- Review your `sfdx-project.json` file. The CLI automatically updates the project file to include the package directory and creates an alias based on the package name.

```
{
  "packageDirectories": [
    {
      "path": "force-app",
      "default": true,
      "package": "Expense Manager",
      "versionName": "ver 0.1",
      "versionNumber": "0.1.0.NEXT"
    }
  ],
  "namespace": "exp-mgr",
  "sfdcLoginUrl": "https://login.salesforce.com",
  "sourceApiVersion": "51.0",
  "packageAliases": {
    "Expense Manager": "0Hoxxx"
  }
}
```

Notice the placeholder values for `versionName` and `versionNumber`. You can update these values, or indicate base packages that this package depends on. Your project file displays the namespace you created.

Specify the features and org settings required for the metadata in your package using an external `.json` file, such as the scratch org definition file. You can specify using the `--definition-file` flag with the `sf package version create` command, or list the definition file in your `sfdx-project.json` file. See: [Project Configuration File for a Second-Generation Managed Package](#)

- Create a package version. This example assumes the package metadata is in the `force-app` directory.

```
sf package version create --package "Expense Manager" --code-coverage --installation-key test1234 --wait 10
```

- Install and test the package version in a scratch org. Use a different scratch org from the one you used in step three.

```
sf package install --package "Expense Manager@0.1.0-1" --target-org MyTestOrg1 --installation-key test1234 --wait 10 --publish-wait 10
```

- After the package is installed, open the scratch org to view the package.

```
sf org open --target-org MyTestOrg1
```

Package versions are beta until you promote them to a managed-released state. See: [Release a Second-Generation Managed Package](#).

SEE ALSO:

[Before You Create Second-Generation Managed Packages](#)

[Create and Register Your Namespace for Second-Generation Managed Packages](#)

[Project Configuration File for a Second-Generation Managed Package](#)

[Release a Second-Generation Managed Package](#)

## Components Available in Second-Generation Managed Packages

Each metadata component that you include in a second-generation managed 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.

Before you review the details about the metadata components that can be included in a managed package, be sure you understand the meaning of each manageability rule.

**Table 1: Manageability Rules**

Component Can Be Updated During Package Upgrade	<p><b>If yes:</b> The component can be updated during a package upgrade. The component is first deployed to the subscriber org during the initial package installation, and subsequent package upgrades update the installed component.</p> <p><b>If no:</b> The component can't be updated during package upgrades. Instead, it's only deployed to the subscriber org during the initial package installation, and subsequent package upgrades don't update the component. Components in this category can typically be modified by the admin in the subscriber org.</p>
Subscriber Can Delete Component	<p><b>If yes:</b> The subscriber or installer of the managed package can delete the packaged component from their org. Deleted components aren't reinstalled during a package upgrade.</p> <p><b>If no:</b> The subscriber or installer of the managed package can't delete the packaged component from their org.</p>
Package Developer Can Remove Component	<p><b>If yes:</b> After the package that contains the component is promoted and released, the package developer can choose to remove the component in a future package version.</p> <p>In most cases, removing components from a package version marks the component as deprecated, and doesn't hard delete the component from the subscriber org. These deprecated components can be deleted by the admin of the subscriber org. For details about which managed 2GP components are deprecated, see <a href="#">Remove Metadata Components from Second-Generation Managed Packages</a>.</p> <p>To request access to this feature, log a support case in the <a href="#">Salesforce Partner Community</a>.</p>

	<p><b>If no:</b> After the package that contains the component is promoted and released, the package developer can't remove the component in a future package version.</p>
Component Has IP Protection	<p><b>If yes:</b> To protect the intellectual property of the developer, the component's metadata, such as Apex code or Custom Metadata record information, is hidden in the installed org.</p> <p><b>If no:</b> The component is visible in the subscriber's org.</p>

## Editable Properties After Package Promotion or Installation

Certain properties on metadata components are editable after the managed package is installed.

- **Only Package Developer Can Edit:** The package developer can edit specific component properties. These properties are locked in the subscriber's org. During package upgrade, the changes made by the package developer are applied in the subscriber org. For example, when you update the code in an Apex class or the custom permissions in a permission set, subscribers receive those updates during their package upgrade.
- **Both Subscriber and Package Developer Can Edit:** Both the subscriber and package developer can edit these component properties, but developer changes are only applied to new subscriber installs. This approach prevents a package upgrade from overwriting changes made by the subscriber. For example, the help text on a custom field, and the page layout of a custom object are editable by both the subscriber and package developer. The subscriber can modify the page layout or help text, and trust that their changes won't be overwritten by a future package upgrade.
- **Neither Subscriber or Package Developer Can Edit:** After a package is promoted and released, these component properties are locked and can't be edited by the package developer or the subscriber. For example, the API names of packaged components are locked and can't be edited after the package version is promoted and released.

## Supported Components in Second-Generation Managed Packages

### [Account Relationship Share Rule](#)

Determines which object records are shared, how they're shared, the account relationship type that shares the records, and the level of access granted to the records.

### [Action Link Group Template](#)

Represents the action link group template. Action link templates let you reuse action link definitions and package and distribute action links.

### [Action Plan Template](#)

Represents an instance of an action plan template.

### [Actionable List Definition](#)

Represents the data source definition details associated with an actionable list.

### [Actionable List Key Performance Indicator Definition](#)

Represents the custom key performance indicators that are defined for a specific field in an object.

### [Activation Platform](#)

Represents the ActivationPlatform configuration, such as platform name, delivery schedule, output format, and destination folder.

[AffinityScoreDefinition](#)

Represents the affinity information used in calculations to analyze and categorize contacts for marketing purposes.

[AI Application](#)

Represents an instance of an AI application. For example, Einstein Prediction Builder.

[AI Application Config](#)

Represents additional prediction information related to an AI application.

[AIUsecaseDefinition](#)

Represents a collection of fields in a Salesforce org used to define a machine learning use case and get real-time predictions.

[Analytics](#)

Analytics components include analytics applications, dashboards, dataflows, datasets, lenses, recipes, and user XMD.

[Apex Class](#)

Represents an Apex Class. An Apex class is a template or blueprint from which Apex objects are created. Classes consist of other classes, user-defined methods, variables, exception types, and static initialization code.

[Apex Sharing Reason](#)

Represents an Apex sharing reason, which is used to indicate why sharing was implemented for a custom object.

[Apex Trigger](#)

Represents an Apex trigger. A trigger is Apex code that executes before or after specific data manipulation language (DML) events occur, such as before object records are inserted into the database, or after records have been deleted.

[Application Subtype Definition](#)

Represents a subtype of an application within an application domain.

[AssessmentConfiguration](#)

Represents a configuration for Assessment component. An AssessmentConfiguration entry indicates configuration for user flows such as sending out emails or reminder actions on assessments initiated by the patient.

[AssessmentQuestion](#)

Represents the container object that stores the questions required for an assessment.

[AssessmentQuestionSet](#)

Represents the container object for Assessment Questions.

[Aura Component](#)

Represents an Aura definition bundle. A bundle contains an Aura definition, such as an Aura component, and its related resources, such as a JavaScript controller. The definition can be a component, application, event, interface, or a tokens collection.

[Batch Calc Job Definition](#)

Represents a Data Processing Engine definition.

[Batch Process Job Definition](#)

Represents the details of a Batch Management job definition.

[Benefit Action](#)

Represents details of an action that can be triggered for a benefit.

[Bot Template](#)

Represents the configuration details for a specific Einstein Bot template, including dialogs and variables.

[Branding Set](#)

Represents the definition of a set of branding properties for an Experience Builder site, as defined in the Theme panel in Experience Builder.

### [Briefcase Definition](#)

Represents a briefcase definition. A briefcase makes selected records available for specific users and groups to view when they're offline in the Salesforce Field Service mobile app for iOS and Android.

### [Building Energy Intensity Record Type Configuration](#)

Represents the setup object that contains the mapping between the Building Energy Intensity Record record type and internal enums. You can primarily use this object for calculations across different record types.

### [Business Process](#)

The BusinessProcess metadata type enables you to display different picklist values for users based on their profile.

### [Business Process Group](#)

Represents the surveys used to track customers' experiences across different stages in their lifecycle.

### [Business Process Type Definition](#)

Define the types of business processes that are applied to a rule.

### [Care Benefit Verify Settings](#)

Represents the configuration settings for benefit verification requests.

### [Care Limit Type](#)

Defines the characteristics of limits on benefit provision.

### [Care Request Configuration](#)

Represents the details for a record type such as service request, drug request, or admission request. One or more record types can be associated with a care request.

### [Care System Field Mapping](#)

Represents a mapping from source system fields to Salesforce target entities and attributes.

### [Channel Layout](#)

Represents the metadata associated with a communication channel layout.

### [Chatter Extension](#)

Represents the metadata used to describe a Rich Publisher App that's integrated with the Chatter publisher.

### [Claim Financial Settings](#)

Represents the configuration settings for Insurance Claim Financial Services.

### [Community Template Definition](#)

Represents the definition of an Experience Builder site template.

### [Community Theme Definition](#)

Represents the definition of a theme for an Experience Builder site.

### [Compact Layout](#)

Represents the metadata associated with a compact layout.

### [Conditional Formatting Ruleset](#)

Represents a set of rules that define the style and visibility of conditional field formatting on Dynamic Forms-enabled Lightning page field instances.

### [Connected App](#)

Represents a connected app configuration. A connected app enables an external application to integrate with Salesforce using APIs and standard protocols, such as SAML, OAuth, and OpenID Connect.



### [Contract Type](#)

A contract type is used to group contracts so that they exhibit similar characteristics. For example, the lifecycle states, the people who access, the templates and clauses used.

### [Conversation Channel Definition](#)

Represents the conversation channel definition that's implemented for Interaction Service for Bring Your Own Channel and Bring Your Own Channel for CCaaS messaging channels.

### [Conversation Vendor Info](#)

This setup object connects the partner vendor system to the Service Cloud feature.

### [CORS Allowlist](#)

Represents an origin in the CORS allowlist.

### [CSP Trusted Site](#)

Represents a trusted URL. For each CspTrustedSite component, you can specify Content Security Policy (CSP) directives and permissions policy directives.

### [Custom Application](#)

Represents a custom application.

### [Custom Button or Link](#)

Represents a custom link defined in a home page component.

### [Custom Console Components](#)

Represents a custom console component (Visualforce page) assigned to a CustomApplication that is marked as a Salesforce console. Custom console components extend the capabilities of Salesforce console apps.

### [Custom Field on Standard or Custom Object](#)

Represents the metadata associated with a field. Use this metadata type to create, update, or delete custom field definitions on standard or custom objects.

### [Custom Field on Custom Metadata Type](#)

Represents a custom fields on the custom metadata type.

### [Custom Help Menu Section](#)

Represents the section of the Lightning Experience help menu that the admin added to display custom, org-specific help resources for the org. The custom section contains help resources added by the admin.

### [Custom Index](#)

Represents an index used to increase the speed of queries.

### [Custom Label](#)

The CustomLabels metadata type allows you to create custom labels that can be localized for use in different languages, countries, and currencies.

### [Custom Metadata Type Records](#)

Represents a record of a custom metadata type.

### [Custom Metadata Type](#)

Represents a record of a custom metadata type.

### [Custom Notification Type](#)

Represents the metadata associated with a custom notification type.

### [Custom Object](#)

Represents a custom object that stores data unique to an org or an external object that maps to data stored outside an org.

### [Custom Object Translation](#)

This metadata type allows you to translate custom objects for a variety of languages.

### [Custom Permission](#)

Represents a permission that grants access to a custom feature.

### [Custom Tab](#)

Represents a custom tab. Custom tabs let you display custom object data or other web content in Salesforce.

### [Dashboard](#)

Represents a dashboard. Dashboards are visual representations of data that allow you to see key metrics and performance at a glance.

### [DataCalcInsightTemplate](#)

Represents the object template for data calculations and insights of Data Cloud objects in DataCalcInsightTemplate. These objects are added inside the data kit.

### [Data Connector Ingest API](#)

Represents the connection information specific to Ingestion API.

### [Data Connector S3](#)

Represents the connection information specific to Amazon S3.

### [Data Kit Object Dependency](#)

Represent the object dependencies and relationships between different objects in Data Kit Object Dependency. These objects are added inside the data kit.

### [Data Kit Object Template](#)

Represents the object in Data Kit Object Template. This object template is added inside the data kit.

### [Data Package Kit Definition](#)

Represents the top-level Data Kit container definition. Content objects can be added after the Data Kit is defined.

### [Data Package Kit Object](#)

Represents the object in Data Kit Content Object. These objects are added inside the data kit.

### [Data Source](#)

Used to represent the system where the data was sourced.

### [Data Source Bundle Definition](#)

Represents the bundle of streams that a user adds to a data kit.

### [Data Source Object](#)

Represents the object from where the data was sourced.

### [Data Src Data Model Field Map](#)

Represents the entity that's used to store the design-time bundle-level mappings for the data source fields and data model fields.

### [Data Stream Definition](#)

Contains Data Ingestion information such as Connection, API and File retrieval settings.

### [Data Stream Template](#)

Represents the data stream that a user adds to a data kit.

### [DataWeaveResource](#)

Represents the DataWeaveScriptResource class that is generated for all DataWeave scripts. DataWeave scripts can be directly invoked from Apex.

### [Decision Matrix Definition](#)

Represents a definition of a decision matrix.

[Decision Matrix Definition Version](#)

Represents a definition of a decision matrix version.

[Decision Table](#)

Represents the information about a decision table.

[Decision Table Dataset Link](#)

Represents the information about a dataset link associated with a decision table. In a dataset link, select an object for whose records, the decision table must provide an outcome.

[Digital Experience](#)

Represents a text-based code structure of your organization's workspaces, organized by workspace type, and each workspace's content items.

[Digital Experience Bundle](#)

Represents a text-based code structure of your organization's workspaces, organized by workspace type, and each workspace's content items.

[Decision Table](#)

Represents the information about a decision table.

[Disclosure Definition](#)

Represents information that defines a disclosure type, such as details of the publisher or vendor who created or implemented the report.

[Disclosure Definition Version](#)

Represents the version information about the disclosure definition.

[Disclosure Type](#)

Represents the types of disclosures that are done by an individual or an organization and the associated metadata.

[Discovery AI Model](#)

Represents the metadata associated with a model used in Einstein Discovery.

[Discovery Goal](#)

Represents the metadata associated with an Einstein Discovery prediction definition.

[Discovery Story](#)

Represents the metadata associated with a story used in Einstein Discovery.

[Document](#)

Represents a Document. All documents must be in a document folder, such as sampleFolder/TestDocument.

[Document Generation Setting](#)

Represents an org's settings for automatic document generation from templates.

[Eclair GeoData](#)

Represents an Analytics custom map chart. Custom maps are user-defined maps that are uploaded to Analytics and are used just as standard maps are. Custom maps are accessed in Analytics from the list of maps available with the map chart type.

[Email Template \(Classic\)](#)

Use email templates to increase productivity and ensure consistent messaging. Email templates with merge fields let you quickly send emails that include field data from Salesforce records.

[Email Template \(Lightning\)](#)

Represents a template for an email, mass email, list email, or Sales Engagement email.

### [Embedded Service Config](#)

Represents a setup node for creating an Embedded Service for Web deployment.

### [Embedded Service Menu Settings](#)

Represents a setup node for creating a channel menu deployment. Channel menus list the ways in which customers can contact your business.

### [Enablement Measure Definition](#)

Represents an Enablement measure, which specifies the job-related activity that a user performs to complete a milestone or outcome in an Enablement program. A measure identifies a source object and optional related objects, with optional field filters and filter logic, for tracking the activity.

### [Enablement Program Definition](#)

Represents an Enablement program, which includes exercises and measurable milestones to help users such as sales reps achieve specific outcomes related to your company's revenue goals.

### [Enablement Program Task Subcategory](#)

Represents a custom exercise type that an Enablement admin adds to an Enablement program in Program Builder. A custom exercise type also requires a corresponding EnblProgramTaskDefinition record for Program Builder and corresponding LearningItem and LearningItemType records for when users take the exercise in the Guidance Center.

### [Entitlement Template](#)

Represents an entitlement template. Entitlement templates are predefined terms of customer support that you can quickly add to products.

### [ESignature Config](#)

Using the Electronic Signature Configuration setup, the system admin must define the required configurations to support the e-signature APIs and UI.

### [ESignature Envelope Config](#)

Using the Electronic Signature Envelope Config the system admin can define the default reminders and expiry for the envelopes submitted for eSignature.

### [Explainability Action Definition](#)

Define where the metadata for your Decision Explainer business rules are stored in Public Sector Solutions.

### [Explainability Action Version](#)

Define and store versions of the explainability actions used by your Decision Explainer business rules in Public Sector Solutions.

### [Explainability Message Template](#)

Represents information about the template that contains the decision explanation message for a specified expression set step type.

### [Expression Set Definition](#)

Represents an expression set definition.

### [Expression Set Definition Version](#)

Represents a definition of an expression set version.

### [Expression Set Object Alias](#)

Represents information about the alias of the source object that's used in an expression set.

### [Expression Set Message Token](#)

Represents a token that's used in an explainability message template. The token can be replaced with an expression set version resource that the template is used in. This object is available in API version 59.0 and later.

### [External Client App Header](#)

Represents the header file for an external client application configuration.

[External Client App OAuth Settings](#)

Represents the settings configuration for the external client app's OAuth plugin.

[External Credential](#)

Represents the details of how Salesforce authenticates to the external system.

[External Data Connector](#)

Used to represent the object where the data was sourced.

[External Data Source](#)

Represents the metadata associated with an external data source. Create external data sources to manage connection details for integration with data and content that are stored outside your Salesforce org.

[External Data Transport Field Template](#)

Represents the definition of a Data Cloud schema field.

[External Data Transport Object Template](#)

Represents the definition of a Data Cloud schema object.

[External Document Storage Configuration](#)

Represents configuration, which admin makes in setup to specify the drive, path, and named credential to be used for storing documents on external drives.

[External Services](#)

Represents the External Service configuration for an org.

[Feature Parameter Boolean](#)

Represents a boolean feature parameter in the Feature Management App (FMA). Feature parameters let you drive app behavior and track activation metrics in subscriber orgs that install your package.

[Feature Parameter Date](#)

Represents a date feature parameter in the Feature Management App (FMA). Feature parameters let you drive app behavior and track activation metrics in subscriber orgs that install your package.

[Feature Parameter Integer](#)

Represents an integer feature parameter in the Feature Management App (FMA). Feature parameters let you drive app behavior and track activation metrics in subscriber orgs that install your package.

[Field Set](#)

Represents a field set. A field set is a grouping of fields. For example, you could have a field set that contains fields describing a user's first name, middle name, last name, and business title.

[Field Source Target Relationship](#)

Stores the relationships between a data model object (DMO) and its fields. For example, the Individual.Id field has a one-to-many relationship (1:M) with the ContactPointEmail.PartyId field.

[Flow](#)

Represents the metadata associated with a flow. With Flow, you can create an application that navigates users through a series of pages to query and update records in the database. You can also execute logic and provide branching capability based on user input to build dynamic applications.

[Flow Category](#)

Represents a list of flows that are grouped by category.

[Flow Definition](#)

Represents the flow definition's description and active flow version number.

### [Flow Test](#)

Represents the metadata associated with a flow test. Before you activate a record-triggered flow, you can test it to verify its expected results and identify flow run-time failures.

### [Folder](#)

Represents a folder.

### [Fuel Type](#)

Represents a custom fuel type in an org.

### [Fuel Type Sustainability Unit of Measure](#)

Represents a mapping between the custom fuel types and their corresponding unit of measure (UOM) values defined by a customer in an org.

### [Fundraising Config](#)

Represents a collection of settings to configure the fundraising product.

### [Gateway Provider Payment Method Type](#)

Represents an entity that allows integrators and payment providers to choose an active payment to receive an order's payment data rather than allowing the Salesforce Order Management platform to select a default payment method.

### [Gen AI Function](#)

Represents a copilot action that can be added to Einstein Agents and Copilot.

### [Global Picklist](#)

Represents the metadata for a global picklist value set, which is the set of shared values that custom picklist fields can use. A global value set isn't a field itself. In contrast, the custom picklist fields that are based on a global picklist are of type ValueSet.

### [Home Page Component](#)

Represents the metadata associated with a home page component. You can customize the Home tab in Salesforce Classic to include components such as sidebar links, a company logo, a dashboard snapshot, or custom components that you create. Use to create, update, or delete home page component definitions.

### [Home Page Layout](#)

Represents the metadata associated with a home page layout. You can customize home page layouts and assign the layouts to users based on their user profile.

### [Identity Verification Proc Def](#)

Represents the definition of the identity verification process.

### [Inbound Network Connection](#)

Represents a private connection between a third-party data service and a Salesforce org. The connection is inbound because the callouts are coming into Salesforce.

### [IndustriesEinsteinFeatureSettings](#)

Represents the settings for enabling the Industries Einstein feature.

### [IntegrationProviderDef](#)

Represents an integration definition associated with a service process. Stores data for the Industries: Send Apex Async Request and Industries: Send External Async Request invocable actions.

### [LearningAchievementConfig](#)

Represents the mapping details between a Learning Achievement type and a Learning Achievement record type.

### [Learning Item Type](#)

Represents a custom exercise type that an Enablement user takes in an Enablement program in the Guidance Center. A custom exercise type also requires a corresponding LearningItem record for the Guidance Center and corresponding EnblProgramTaskDefinition and EnblProgramTaskSubCategory records for when admins create a program in Program Builder.

### [Letterhead](#)

Represents formatting options for the letterhead in an email template. A letterhead defines the logo, page color, and text settings for your HTML email templates. Use letterheads to ensure a consistent look and feel in your company's emails.

### [Lightning Bolt](#)

Represents the definition of a Lightning Bolt Solution, which can include custom apps, flow categories, and Experience Builder templates.

### [Lightning Message Channel](#)

Represents the metadata associated with a Lightning Message Channel. A Lightning Message Channel represents a secure channel to communicate across UI technologies, such as Lightning Web Components, Aura Components, and Visualforce.

### [Lightning Page](#)

Represents the metadata associated with a Lightning page. A Lightning page represents a customizable screen made up of regions containing Lightning components.

### [Lightning Web Component](#)

Represents a Lightning web component bundle. A bundle contains Lightning web component resources.

### [List View](#)

List View allows you to see a filtered list of records, such as contacts, accounts, or custom objects.

### [Live Chat Sensitive Data Rule](#)

Represents a rule for masking or deleting data of a specified pattern. Written as a regular expression (regex). Use this object to mask or delete data of specified patterns, such as credit card, social security, or phone and account numbers.

### [Loyalty Program Setup](#)

Represents the configuration of a loyalty program process including its parameters and rules. Program processes determine how new transaction journals are processed. When new transaction journals meet the criteria and conditions for a program process, actions that are set up in the process are triggered for the transaction journals.

### [Marketing App Extension](#)

Represents an integration with a third-party app or service that generates prospect external activity.

### [MarketingAppExtAction](#)

Represents an Action Type, which is an action that you can add to Engagement Studio programs in Account Engagement and execute in a third-party app.

### [Marketing App Extension Activity](#)

Represents an Activity Type, which is a prospect activity that occurs in a third-party app and can be used in Account Engagement automations.

### [Market Segment Definition](#)

Represents the field values for MarketSegmentDefinition. MarketSegmentDefinition is used to store the exportable metadata of a segment, such as segment criteria and other attributes. Developers can create segment definition packages, pass segment definition in the form of data build tool (DBT), and publish it on AppExchange for subscriber organizations to install and instantiate these segments.

### [MktCalculatedInsightsObjectDef](#)

Represents Calculated Insight definition such as expression.

### [MktDataConnection](#)

Represents the connection information of an external connector that can ingest data to Data Cloud, read data from the source, or write data to the source in Data Cloud.

### [MktDataTranObject](#)

An entity that is used to deliver (aka transport) information from the source to a target (target will be called a landing entity). This can be the schema of a file, API, Event, or other means of transporting data, such as SubscriberFile1.csv, or SubscriberCDCEvent.

### [Named Credential](#)

Represents a named credential, which specifies the URL of a callout endpoint and its required authentication parameters in one definition. A named credential can be specified as an endpoint to simplify the setup of authenticated callouts.

### [Object Source Target Map](#)

Contains the object-level mappings between the source and the target objects. The source and target objects can be an MktDataLakeObject or an MktDataModelObject. For example, an Email source object can be mapped to the ContactPointEmail object.

### [OcrSampleDocument](#)

Represents the details of a sample document or a document type that's used as a reference while extracting and mapping information from a customer form.

### [OcrTemplate](#)

Represents the details of the mapping between a form and a Salesforce object using Intelligent Form Reader.

### [Outbound Network Connection](#)

Represents a private connection between a Salesforce org and a third-party data service. The connection is outbound because the callouts are going out of Salesforce.

### [Page Layout](#)

Represents the metadata associated with a page layout.

### [Path Assistant](#)

Represents Path records.

### [Payment Gateway Provider](#)

Represents the metadata associated with a payment gateway provider.

### [Permission Set](#)

Represents a set of permissions that's used to grant more access to one or more users without changing their profile or reassigning profiles. You can use permission sets to grant access but not to deny access.

### [Permission Set Groups](#)

Represents a group of permission sets and the permissions within them. Use permission set groups to organize permissions based on job functions or tasks. Then, you can package the groups as needed.

### [Platform Cache](#)

Represents a partition in the Platform Cache.

### [Platform Event Channel](#)

Represents a channel that you can subscribe to in order to receive a stream of events.

### [Platform Event Channel Member](#)

Represents an entity selected for Change Data Capture notifications on a standard or custom channel, or a platform event selected on a custom channel.

### [Platform Event Subscriber Configuration](#)

Represents configuration settings for a platform event Apex trigger, including the batch size and the trigger's running user.



### [Pricing Action Parameters](#)

Represents a pricing action associated to a context definition and a pricing procedure.

### [Pricing Recipe](#)

Represents one out of various data models or sets of entities of a particular cloud that'll be consumed by the pricing data store during design and run time.

### [Process](#)

Use Flow instead.

### [Process Flow Migration](#)

Represents a process's migrated criteria and the resulting migrated flow.

### [Product Attribute Set](#)

Represents the ProductAttribute information being used as an attribute such as color\_c, size\_c .

### [Product Specification Type](#)

Represents the type of product specification provided by the user to make the product terminology unique to an industry. A product specification type is associated with a product specification record type.

### [Product Specification Record Type](#)

Represents the relationship between industry-specific product specifications and the product record type.

### [Prompts \(In-App Guidance\)](#)

Represents the metadata related to in-app guidance, which includes prompts and walkthroughs.

### [Quick Action](#)

Represents a specified create or update quick action for an object that then becomes available in the Chatter publisher.

### [Recommendation Strategy](#)

Represents a recommendation strategy. Recommendation strategies are applications, similar to data flows, that determine a set of recommendations to be delivered to the client through data retrieval, branching, and logic operations.

### [Record Action Deployment](#)

Represents configuration settings for the Actions & Recommendations, Action Launcher, and Bulk Action Panel components.

### [Record Alert Data Source Expression Set Definition](#)

Represents information about the data source for a record alert and the association with an expression set definition.

### [Record Type](#)

Represents the metadata associated with a record type. Record types let you offer different business processes, picklist values, and page layouts to different users. Use this metadata type to create, update, or delete record type definitions for a custom object.

### [RedirectWhitelistUrl](#)

Represents a trusted URL that's excluded from redirection restrictions when the redirectionWarning or redirectBlockModeEnabled field on the SessionSettings Metadata type is set to true.

### [Referenced Dashboard](#)

Represents the ReferencedDashboard object in CRM Analytics. A referenced dashboard stores information about an externally referenced dashboard.

### [Registered External Service](#)

Represents a registered external service, which provides an extension or integration.

### [RelationshipGraphDefinition](#)

Represents a definition of a graph that you can configure in your organization to traverse object hierarchies and record details, giving you a glimpse of how your business works.

[Remote Site Setting](#)

Represents a remote site setting.

[Report](#)

Represents a custom report.

[Report Type](#)

Represents the metadata associated with a custom report type. Custom report types allow you to build a framework from which users can create and customize reports.

[ServiceProcess](#)

Represents a process created in Service Process Studio and its associated attributes.

[Slack App \(Beta\)](#)

Represents a Slack app.

[Service Catalog Category](#)

Represents the grouping of individual catalog items in Service Catalog.

[Service Catalog Filter Criteria](#)

Represents an eligibility rule that determines if a Service Catalog user has access to a catalog item.

[Service Catalog Item Definition](#)

Represents the entity associated with a specific, individual service available in the Service Catalog.

[Service Catalog Fulfillment Flow](#)

Represents the flow associated with a specific catalog item in the Service Catalog.

[Stationary Asset Environmental Source Record Type Configuration](#)

Represents the setup object that contains the mapping between the Stationary Asset Environmental Source record type and internal enums. You can primarily use this object for calculations across different record types.

[Static Resource](#)

Represents a static resource file, often a code library in a ZIP file.

[Streaming App Data Connector](#)

Represents the connection information specific to Web and Mobile Connectors.

[Sustainability UOM](#)

Represents information about the additional unit of measure values defined by a customer.

[Sustainability UOM Conversion](#)

Represents information about the unit of measure conversion for the additional fuel types defined by a customer.

[Timeline Object Definition](#)

Represents the container that stores the details of a timeline configuration. You can use this resource with Salesforce objects to see their records' related events in a linear time-sorted view.

[Timesheet Template](#)

Represents a template for creating time sheets in Field Service.

[Translation](#)

Add translations to your managed packages.

[UI Object Relation Config](#)

Represents the admin-created configuration of the object relation UI component.

[User Access Policy](#)

Represents a user access policy.

[Validation Rule](#)

Represents a validation rule, which is used to verify that the data a user enters in a record is valid and can be saved.

[Vehicle Asset Emissions Source Record Type Configuration](#)

Represents the setup object that contains the mapping between the Vehicle Asset Emissions Source record type and internal enums. You can primarily use this object for calculations across different record types.

[View Definition \(Beta\)](#)

Represents a view definition on a Slack app.

[Virtual Visit Config](#)

Represents an external video provider configuration, which relays events from Salesforce to the provider.

[Visualforce Component](#)

Represents a Visualforce component.

[Visualforce Page](#)

Represents a Visualforce page.

[Wave Analytic Asset Collection](#)

A collection of CRM Analytics assets.

[Wave Application](#)

A CRM Analytics application.

[Wave Component](#)

A CRM Analytics dashboard component.

[Wave Dataflow](#)

A CRM Analytics data prep dataflow.

[Wave Dashboard](#)

A CRM Analytics dashboard.

[Wave Dataset](#)

A CRM Analytics dataset.

[Wave Lens](#)

A CRM Analytics lens.

[Wave Recipe](#)

A CRM Analytics data prep recipe.

[Wave Template Bundle](#)

A CRM Analytics template bundle.

[Wave Xmd](#)

The extended metadata for CRM Analytics dataset fields and their formatting for dashboards and lenses.

[Web Store Template](#)

Represents a configuration for creating commerce stores.

[Workflow Alert](#)

WorkflowAlert represents an email alert associated with a workflow rule.

[Workflow Field Update](#)

WorkflowFieldUpdate represents a workflow field update.

[Workflow Knowledge Publish](#)

WorkflowKnowledgePublish represents Salesforce Knowledge article publishing actions and information.

[Workflow Outbound Message](#)

WorkflowOutboundMessage represents an outbound message associated with a workflow rule.

[Workflow Rule](#)

This metadata type represents a workflow rule.

[Workflow Task](#)

This metadata type references an assigned workflow task.

## Account Relationship Share Rule

Determines which object records are shared, how they're shared, the account relationship type that shares the records, and the level of access granted to the records.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Name
- Developer Name
- Description
- Account Relationship Type
- Access Level
- Object Type
- Account to Criteria Field

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: AccountRelationshipShareRule

### Use Case

To share data between external accounts.

### License Requirements

Orgs with Digital Experiences enabled can use this package.

### Documentation

*Salesforce Help:* [Account Relationships and Account Relationship Data Sharing Rules](#)


## Action Link Group Template

Represents the action link group template. Action link templates let you reuse action link definitions and package and distribute action links.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ActionLinkGroupTemplate

Component Type in 1GP Package Manager UI: Action Link Group Template

### Documentation

Salesforce Help: [Action Link Templates](#)


## Action Plan Template

Represents an instance of an action plan template.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ActionPlanTemplate

**Documentation***Salesforce Help:* [Action Plans](#)


## Actionable List Definition

Represents the data source definition details associated with an actionable list.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

**Feature Name**

Metadata Name: ActionableListDefinition

Component Type in 1GP Package Manager UI: ActionableListDefinition

**Documentation**

*Salesforce Help:* [Actionable Segmentation](#)

## Actionable List Key Performance Indicator Definition


Represents the custom key performance indicators that are defined for a specific field in an object.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes, Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

To confirm whether this component is available in managed 1GP, managed 2GP, or both package types, see [Metadata Coverage Report](#).

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- All attributes

#### Both Package Developer and Subscriber Can Edit

- All attributes

#### Neither Package Developer or Subscriber Can Edit

- None

### More Information

#### Feature Name

Metadata Name: ActnblListKeyPrfmIndDef

Component Type in 1GP Package Manager UI: ActnblListKeyPrfmIndDef

#### License Requirements

Actionable Segmentation

#### Documentation

*Salesforce Help:* [Create Custom Key Performance Indicators](#)

*Salesforce Help:* [ActnblListKeyPrfmIndDef](#)




## Activation Platform

Represents the ActivationPlatform configuration, such as platform name, delivery schedule, output format, and destination folder.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- DataConnector
- Description
- LogoUrl
- MasterLabel
- OutputFormat
- RefreshMode
- Type

### Both Package Developer and Subscriber Can Edit

- Enabled (only subscriber editable)
- IncludeSegmentNames (only subscriber editable)

### Neither Package Developer or Subscriber Can Edit

- ID
- OutputGrouping
- PeriodicRefreshFrequency
- RefreshFrequency

## More Information

### Feature Name

Metadata Name: ActivationPlatform

Component Type in 1GP Package Manager UI: ActivationPlatform

### Use Case

Allows ISVs to specify capabilities of their Activation Platform integrations and publish it on AppExchange for subscriber organizations to install and instantiate instances of the platform as a disparate activation target.

### Considerations When Packaging

Some upgrade scenarios are not support:

- Adding a new required field
- Removing a previously supported ID type
- Removing a previously supported optional field or required field
- Changing a previously supported field property from optional to required

Some update scenarios are supported and don't automatically cascade to Activation Target or Activations created before the upgrade installations:

- Adding a new ID type
- Adding of a new optional field
- Adding a new hidden field
- Value change on a previously supported hidden field

To apply updates to future Activation run jobs, the user must edit and resave all Activation Targets created before the upgrade. Developers provide post-install instructions informing the subscriber of this required action anytime a change is made in a new version release.

### License Requirements

Data Cloud enabled orgs can access this package.

### Post Install Steps

An admin from the subscriber org enables the activation platform to start using this platform in Activation creations.

### Documentation

Metadata API Developer Guide: [ActivationPlatform](#)

## AffinityScoreDefinition


Represents the affinity information used in calculations to analyze and categorize contacts for marketing purposes.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- AffinityScoreType
- NumberOfMonths
- NumberOfRanges
- SourceFieldApiNameList
- TargetFieldApiNameList
- ScoreRangeList

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: AffinityScoreDefinition

### Documentation

- *Fundraising Metadata API Types:* [AffinityScoreDefinitions](#)
- *Salesforce Help:* [Set Up RRM Scoring](#)
- *Salesforce Help:* [Scoring Frameworks Help Increase Fundraising Success](#)

## AI Application

Represents an instance of an AI application. For example, Einstein Prediction Builder.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
-----------------	---

Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Type

### Both Package Developer and Subscriber Can Edit

- Status
- ExternalId
- MIExternalId

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: AIApplication

### Considerations When Packaging

AIApplication is the parent entity for all Einstein configuration entities. Packaging of Einstein features must always begin with the selection of one or more AIApplications. To create a package with ML Prediction Definition, select the parent AIApplication (Type = PredictionBuilder). To create a package with ML Recommendation Definition, select the parent AIApplication (Type = RecommendationBuilder). Packaging automatically analyzes the relationships and includes the associated MLPredictionDefinitions, MLRecommendationDefinitions, and MLDataDefinitions necessary to fully define the Einstein configuration.

### Documentation

Metadata API Developer Guide: [AIApplication](#)

Salesforce Help: [Einstein Prediction Builder](#)

Salesforce Help: [Einstein Recommendation Builder](#)

## AI Application Config

Represents additional prediction information related to an AI application.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In: Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)

Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- AIApplicationId

### Both Package Developer and Subscriber Can Edit

- Rank
- IsInsightReasonEnabled
- IsInsightReasonEnabled
- AIScoringMode
- ExternalId

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: AIApplicationConfig

### Considerations When Packaging

AIApplicationConfig is always associated with an AIApplication. Packaging of Einstein features must always begin with the selection of one or more AIApplications. To create a package with AI Application Config, select the parent AIApplication. Packaging automatically analyzes the relationships and includes the associated MLApplicationConfig, MLPredictionDefinition, MLRecommendationDefinitions, and MLDataDefinitions necessary to fully define the Einstein configuration.

### Documentation

*Metadata API Developer Guide:* [AIApplicationConfig](#)

*Salesforce Help:* [Einstein Prediction Builder](#)

*Salesforce Help:* [Einstein Recommendation Builder](#)


## AIUsecaseDefinition

Represents a collection of fields in a Salesforce org used to define a machine learning use case and get real-time predictions.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All the AIUsecaseDefinition fields

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: AIUsecaseDefinition

Component Type in 1GP Package Manager UI: AIUsecaseDefinition

### Use Case

AI Usecase Definition lets you ship data that can be used to set up use cases for which you want to generate real-time predictions. This data includes machine learning models and feature extractors required to generate the real-time predictions.

### License Requirements

This feature is available with the CRM Plus license and the use case-related product's CRM license.

### Documentation

*Industries Common Resources Developer Guide:* [AI Accelerator](#)

*Salesforce Help:* [AI Accelerator](#)


## Analytics

Analytics components include analytics applications, dashboards, dataflows, datasets, lenses, recipes, and user XMD.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes (Analytics Dataflow only). All other analytics components can't be updated.
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (Analytic snapshot only). Supported in managed 2GP packages only. All other analytics components can't be removed.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## More Information

To include analytics components in a managed 2GP package, include [EinsteinAnalyticsPlus](#) in your scratch org definition file.

To enable analytics in a 1GP packaging org, see [Basic CRM Analytics Platform Setup](#) in Salesforce Help.

For more details, see [CRM Analytics Packaging Considerations](#).

## Apex Class


Represents an Apex Class. An Apex class is a template or blueprint from which Apex objects are created. Classes consist of other classes, user-defined methods, variables, exception types, and static initialization code.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	Yes (if not set to global access). Supported in both 1GP and 2GP packages.
Component Has IP Protection	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- API Version
- Code

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: ApexClass

Component Type in 1GP Package Manager UI: Apex Class

### Considerations When Packaging

- Any Apex that is included as part of a package must have at least 75% cumulative test coverage. Each trigger must also have some test coverage. When you upload your package to AppExchange, all tests are run to ensure that they run without errors. In addition, all tests are run when the package is installed in the installer's org. If any test fails, the installer can decide whether to install the package.
- Managed packages receive a unique namespace. This namespace is prepended to your class names, methods, variables, and so on, which helps prevent duplicate names in the installer's org.
- In a single transaction, you can only reference 10 unique namespaces. For example, suppose that you have an object that executes a class in a managed package when the object is updated. Then that class updates a second object, which in turn executes a different class in a different package. Even though the first package didn't access the second package directly, the access occurs in the same transaction. It's therefore included in the number of namespaces accessed in a single transaction.
- If you're exposing any methods as Web services, include detailed documentation so that subscribers can write external code that calls your Web service.
- If an Apex class references a custom label and that label has translations, explicitly package the individual languages desired to include those translations in the package.



- If you reference a custom object's sharing object (such as `MyCustomObject__share`) in Apex, you add a sharing model dependency to your package. Set the default org-wide access level for the custom object to Private so other orgs can install your package successfully.
- The code contained in an Apex class, trigger, 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.
- You can use the `deprecated` annotation in Apex to identify `global` methods, classes, exceptions, enums, interfaces, and variables that can't be referenced in later releases of a managed package. So you can refactor code in managed packages as the requirements evolve. After you create another package version as Managed - Released, new subscribers that install the latest package version can't see the deprecated elements, while the elements continue to function for existing subscribers and API integrations.
- Apex code that refers to Data Categories can't be uploaded.
- Before deleting Visualforce pages or global Visualforce components from your package, remove all references to public Apex classes and public Visualforce components. After removing the references, upgrade your subscribers to an interim package version before you delete the page or global component.

### Usage Limits

The maximum number of class and trigger code units in a deployment of Apex is 7500. For more information, see [Execution Governors and Limits](#) in the *Apex Developer Guide*.

### Documentation

*Second-Generation Managed Packaging Developer Guide:* [Namespace-Based Visibility for Apex Classes in Second-Generation Managed Packages](#)

*First-Generation Managed Packaging Developer Guide:* [About API and Dynamic Apex Access in Packages](#)

*First-Generation Managed Packaging Developer Guide:* [Using Apex in Group and Professional Editions](#)

## Apex Sharing Reason

Represents an Apex sharing reason, which is used to indicate why sharing was implemented for a custom object.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Reason Label

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Reason Name

## More Information

### Feature Name

Metadata Name: SharingReason

Component Type in 1GP Package Manager UI: Apex Sharing Reason

### Considerations When Packaging

Apex sharing reasons can be added directly to a package, but are only available for custom objects.

### Documentation

Metadata API Developer Guide: [SharingReason](#)


## Apex Trigger

Represents an Apex trigger. A trigger is Apex code that executes before or after specific data manipulation language (DML) events occur, such as before object records are inserted into the database, or after records have been deleted.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- API Version
- Code

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: ApexTrigger

Component Type in 1GP Package Manager UI: Apex Trigger

### Documentation

*Apex Developer Guide:* [Triggers](#)


## Application Subtype Definition

Represents a subtype of an application within an application domain.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

[Both Package Developer and Subscriber Can Edit](#)

- Label
- Developer Name
- Description
- Application Usage Type

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

**Feature Name**

Metadata Name: ApplicationSubtypeDefinition

**Documentation***Industries Common Resources Developer Guide:* [AssessmentSubtypeDefinition](#)

## AssessmentConfiguration

Represents a configuration for Assessment component. An AssessmentConfiguration entry indicates configuration for user flows such as sending out emails or reminder actions on assessments initiated by the patient.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in managed 1GP packages only.
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All but DeveloperName

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- DeveloperName

## More Information

### Feature Name

Metadata Name: AssessmentConfiguration

Component Type in 1GP Package Manager UI: AssessmentConfiguration

### Documentation

Health Cloud Developer Guide: [AssessmentConfiguration](#)


## AssessmentQuestion

Represents the container object that stores the questions required for an assessment.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All except DeveloperName

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- DeveloperName

## More Information

### Feature Name

Metadata Name: AssessmentQuestion

**Documentation**

*Industries Common Resources Developer Guide:* [AssessmentQuestion](#)


## AssessmentQuestionSet

Represents the container object for Assessment Questions.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- All except DeveloperName

#### Both Package Developer and Subscriber Can Edit

- None

#### Neither Package Developer or Subscriber Can Edit

- DeveloperName

### More Information

**Feature Name**

Metadata Name: AssessmentQuestionSet

**Documentation**

*Industries Common Resources Developer Guide:* [AssessmentQuestionSet](#)

## Aura Component

Represents an Aura definition bundle. A bundle contains an Aura definition, such as an Aura component, and its related resources, such as a JavaScript controller. The definition can be a component, application, event, interface, or a tokens collection.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

You can build Lightning components using two programming models: the Lightning Web Components model, and the original Aura Components model.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

When a package developer removes an Aura or Lightning web component from a package, the component remains in a subscriber's org after they install the upgraded package. The administrator of the subscriber's org can delete the component, if desired. This behavior is the same for a Lightning web component or an Aura component with a `public` or `global` access value.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- API Version
- Description
- Label
- Markup

#### Both Package Developer and Subscriber Can Edit

- None

#### Neither Package Developer or Subscriber Can Edit

- Name

### More Information

#### Aura Component

Metadata Name: AuraDefinitionBundle

Component Type in 1GP Package Manager UI: Aura Component Bundle

**Documentation**

[Lightning Aura Components Developer Guide](#)

## Batch Calc Job Definition

Represents a Data Processing Engine definition.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes, except templates

## Editable Properties After Package Promotion or Installation

### [Only Package Developer Can Edit](#)

- Entire Data Processing Engine definition

### [Both Package Developer and Subscriber Can Edit](#)

- Label
- Description
- Status

### [Neither Package Developer or Subscriber Can Edit](#)

- API Name
- URL

## More Information

**Feature Name**

Metadata Name: BatchCalcJobDefinition

Component Type in 1GP Package Manager UI: Batch Calculation Job Definition

**Use Case**

Data Processing Engine helps you transform data that's available in your Salesforce org and write back the transformation results as new or updated records. You can transform the data for standard and custom objects using Data Processing Engine definitions.

**License Requirements**

Either Financial Services Cloud, Manufacturing Cloud, Loyalty Management, Net Zero Cloud, or Rebate Management



Data Pipelines

### Documentation

*Salesforce Help:* [Data Processing Engine](#)

## Batch Process Job Definition

Represents the details of a Batch Management job definition.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes, except templates

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- Entire Batch Management job

#### Both Package Developer and Subscriber Can Edit

- Label
- Description
- Status

#### Neither Package Developer or Subscriber Can Edit

- API Name
- URL

### More Information

#### Feature Name

Metadata Name: BatchProcessJobDefinition

Component Type in 1GP Package Manager UI: Batch Process Job Definition

#### Use Case

Automate the processing of records in scheduled flows with Batch Management. With Batch Management, you can process a high volume of standard and custom object records.

#### License Requirements

Either Loyalty Management, Manufacturing Cloud, or Rebate Management

System Administrator Profile

### Documentation

*Salesforce Help:* [Batch Management](#)

## Benefit Action

Represents details of an action that can be triggered for a benefit.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes, except templates

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Entire Benefit Action record

### Both Package Developer and Subscriber Can Edit

- Label
- Description
- Status

### Neither Package Developer or Subscriber Can Edit

- API Name
- URL

## More Information

### Feature Name

Metadata Name: BenefitAction

Component Type in 1GP Package Manager UI: Benefit Action

### Use Case

Benefit Actions are actions that can be triggered for a loyalty program benefit.

### License Requirements

Loyalty Management permission set license

**Documentation***Salesforce Help:* [Benefit Action](#)


## Bot Template

Represents the configuration details for a specific Einstein Bot template, including dialogs and variables.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- Bot Dialog Groups
- Bot Dialogs
- Conversation Context Variables
- Conversation Languages
- Conversation Definition Goals
- Conversation System Dialogs
- Conversation Variables
- Description
- Entry Dialog
- Icon
- Main Menu Dialog
- Label
- MIDomain
- Rich Content Enabled

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

**Feature Name**

Metadata Name: BotTemplate

Component Type in 1GP Package Manager UI: Bot Template

**Documentation**[Salesforce Help: Create an Einstein Bot Template](#)[Salesforce Help: Create a Template from an Einstein Bot](#)[Salesforce Help: Package an Einstein Bot Template](#)[Metadata API Developer Guide: BotTemplate](#)

## Branding Set


Represents the definition of a set of branding properties for an Experience Builder site, as defined in the Theme panel in Experience Builder.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

 **Note:** 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.

[Only Package Developer Can Edit](#)

- brandingSetProperty
- description
- masterLabel
- type

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

**Feature Name**

Metadata Name: BrandingSet

**Relationship to Other Components**

BrandingSet can't be added to a package by itself. BrandingSet is included automatically in a package if it's referenced by another object in the package, such as CommunityThemeDefinition, LightningExperienceTheme, or EmbeddedServiceMenuSettings.

**Documentation**

*Salesforce Help:* [Use Branding Sets in Experience Builder](#)

## Briefcase Definition

Represents a briefcase definition. A briefcase makes selected records available for specific users and groups to view when they're offline in the Salesforce Field Service mobile app for iOS and Android.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Entire briefcase

[Both Package Developer and Subscriber Can Edit](#)

- Active

[Neither Package Developer or Subscriber Can Edit](#)

- Full Name

## More Information

### Feature Name

Metadata Name: BriefcaseDefinition

Component Type in 1GP Package Manager UI: Briefcase Definition

### Considerations When Packaging

As a best practice, package Briefcase Definition with IsActive set to false. If you package Briefcase Definition with IsActive set to true, the package installation fails if installing the package exceeds any limits.

### Usage Limits

All [Briefcase Builder limits](#) apply to a Briefcase Definition package.

### Relationship to Other Components

After you install the package, assign the briefcase to the application that the briefcase's data is for.

### Documentation

*Salesforce Help:* [Briefcase Builder](#)

## Building Energy Intensity Record Type Configuration

Represents the setup object that contains the mapping between the Building Energy Intensity Record record type and internal enums. You can primarily use this object for calculations across different record types.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: BldgEnrgyIntensityCnfg

Component Type in 1GP Package Manager UI: Building Energy Intensity Record Type Configuration

### Use Case

You can use this component to build on top of the current Net Zero Cloud data model and carbon accounting capability to create new stationary asset types for end users.

### License Requirements

- Net Zero Cloud Growth license or Net Zero Cloud Starter license
- Net Zero Cloud Manager permissions set

### Post Install Steps

Enable these org settings:

- Net Zero Cloud
- Manage Carbon Accounting
- Manage Building Energy Intensity

### Documentation

- *Salesforce Help:* [Set Up Record Types for Net Zero Cloud](#)
- *Salesforce Help:* [Benchmark Building Energy Intensity Data](#)


## Business Process

The BusinessProcess metadata type enables you to display different picklist values for users based on their profile.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

- [Only Package Developer Can Edit](#)None
- [Both Package Developer and Subscriber Can Edit](#)All attributes
- [Neither Package Developer or Subscriber Can Edit](#)None

## More Information

### Feature Name

Metadata Name: BusinessProcess

### Use Case

You can use this component to define different picklist values that you associate with record types.

### Relationship to Other Components

Record types of corresponding entities.

### Documentation

*Salesforce Help:* [Tailor Business Processes to Different Users Using Record Types](#)

## Business Process Group

Represents the surveys used to track customers' experiences across different stages in their lifecycle.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### [Only Package Developer Can Edit](#)

- All Business Process Group fields including Business Process Definition and Business Process Feedback.

### [Both Package Developer and Subscriber Can Edit](#)

- None

### [Neither Package Developer or Subscriber Can Edit](#)

- Developer Name
- Customer Satisfaction Metric



## More Information

### Feature Name

Metadata Name: BusinessProcessGroup

Component Type in 1GP Package Manager UI: Business Process Group

### Use Case

Business Process Group lets you ship groupings relevant to survey metrics that are captured as part of any purchase or product lifecycle. For a specific business process group, you can define different stages and associate relevant questions from one or more surveys for reporting purposes.

### License Requirements

This feature is available with the Feedback Management - Growth license.

### Relationship to Other Components

This feature can be used in conjunction with Surveys and Survey Invitation Rules Flow types, and their corresponding dependencies.

### Documentation

*Metadata API Developer Guide:* [BusinessProcessGroup](#)

*Salesforce Help:* [Track Satisfaction Across a Customer's Lifecycle](#)


## Business Process Type Definition

Define the types of business processes that are applied to a rule.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- Label
- Developer Name
- Description
- Application Usage Type

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: BusinessProcessTypeDefinition

## Care Benefit Verify Settings

Represents the configuration settings for benefit verification requests.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- MasterLabel
- ServiceApexClass
- ServiceNamedCredential
- UriPath
- isDefault
- GeneralPlanServiceTypeCode
- ServiceTypeSourceSystem
- OrganizationName
- DefaultNpi
- CodeSetType

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- Name

## More Information

### Feature Name

Metadata Name: CareBenefitVerifySettings

Component Type in 1GP Package Manager UI: Care Benefit Verification Settings

### Use Case

Provides out-of-the-box configuration settings for benefit verification requests in Health Cloud.

### License Requirements

Industries Health Cloud

### Relationship to Other Components

CareBenefitVerifySettings can contain ApexClass as well as NamedCredentials.

### Documentation

*Health Cloud Developer Guide:* [CareBenefitVerifySettings](#)

## Care Limit Type

Defines the characteristics of limits on benefit provision.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- LimitType
- MetricType
- MasterLabel

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- Name

## More Information

### Feature Name

Metadata Name: CareLimitType

Component Type in 1GP Package Manager UI: Care Limit Type

### Use Case

Provide the characteristics of limits on benefit provision in Health Cloud.

### License Requirements

Industries Health Cloud Add On or an org with a Health Cloud Financial Data Platform license

### Documentation

*Health Cloud Developer Guide:* [CareLimitType](#)

## Care Request Configuration

Represents the details for a record type such as service request, drug request, or admission request. One or more record types can be associated with a care request.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- MasterLabel
- CareRequestType
- CareRequestRecordType
- CareRequestRecords
- IsDefaultRecordType

[Both Package Developer and Subscriber Can Edit](#)

- IsActive

Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: CareRequestConfiguration

Component Type in 1GP Package Manager UI: Care Request Configuration

### Use Case

Provides the details for a record type such as a service request, drug request, or admission request in Health Cloud.

### License Requirements

Industries Health Cloud Add On an org with a Health Cloud Utilization Mgmt Platform license

### Relationship to Other Components

Ensure that the record type specified in the Case Record Type field in CareRequestConfiguration is available in the subscriber org. Otherwise, the package must include the record type.

### Documentation

Health Cloud Developer Guide: [CareRequestConfiguration](#)

## Care System Field Mapping

Represents a mapping from source system fields to Salesforce target entities and attributes.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- External ID Field
- Is Active
- Label
- Source System

- Target Object

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- Name

## More Information

### Feature Name

Metadata Name: CareSystemFieldMapping

Component Type in 1GP Package Manager UI: Care System Field Mapping

### Use Case

Provides an out-of-the-box mapping for an external system to Salesforce for the Care Program Enrollment or Remote Monitoring features in Health Cloud.

### License Requirements

Industries Health Cloud

### Documentation

*Health Cloud Developer Guide:* [CareSystemFieldMapping](#)

## Channel Layout

Represents the metadata associated with a communication channel layout.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All attributes

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: ChannelLayout

Component Type in 1GP Package Manager UI: Communication Channel Layout

### Considerations When Packaging

ChannelLayout can only be installed in Salesforce Classic orgs with Knowledge enabled.

ChannelLayout includes the article type \*\_\_kav, which is not supported by Lightning Knowledge.

If you try to install ChannelLayout into an org with Lightning Knowledge enabled, this message is displayed: “When Lightning Knowledge is enabled, you can’t add an article type”.

### License Requirements

Enable Knowledge in Salesforce Classic orgs.

### Documentation

[Salesforce Knowledge Developer Guide: ChannelLayout](#)

## Chatter Extension

Represents the metadata used to describe a Rich Publisher App that’s integrated with the Chatter publisher.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Header Text
- Hover Text
- Icon
- Name

### Both Package Developer and Subscriber Can Edit

- None

[Neither Package Developer or Subscriber Can Edit](#)

- Composition CMP
- Render CMP
- Type

## More Information

**Feature Name**

Metadata Name: ChatterExtension

**Documentation***Metadata API Developer Guide:* [ChatterExtension](#)*Object Reference for the Salesforce Platform:* [ChatterExtension](#)

## Claim Financial Settings

Represents the configuration settings for Insurance Claim Financial Services.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Label

[Both Package Developer and Subscriber Can Edit](#)

- Claim Coverage Pending Authority Status
- Claim Coverage Payment Detail Pending Authority Status
- Claim Pending Authority Status

[Neither Package Developer or Subscriber Can Edit](#)

- None



## More Information

### Feature Name

Metadata Name: ClaimFinancialSettings

### Documentation

*Salesforce Help:* [Claim Financial Settings](#)

## Community Template Definition

Represents the definition of an Experience Builder site template.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: CommunityTemplateDefinition

Component Type in 1GP Package Manager UI: Lightning Community Template

### Use Case

Share or distribute your Experience Builder site templates.

### License Requirements

Customize Application user permission

Create and Set Up Experiences user permission

View Setup and Configuration user permission

### Relationship to Other Components

If you add CommunityTemplateDefinition to a package, you must also add CommunityThemeDefinition to the package.

### Documentation

*Salesforce Help:* [Export a Customized Experience Builder Template for a Lightning Bolt Solution](#)

*Salesforce Help:* [Package and Distribute a Lightning Bolt Solution](#)

## Community Theme Definition

Represents the definition of a theme for an Experience Builder site.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

### More Information

#### Feature Name

Metadata Name: CommunityThemeDefinition

Component Type in 1GP Package Manager UI: Lightning Community Theme

#### Use Case

Share or distribute your Experience Builder site themes.

#### License Requirements

Customize Application user permission

Create and Set Up Experiences user permission

View Setup and Configuration user permission

### Relationship to Other Components

CommunityThemeDefinition must contain a BrandingSet.

CommunityThemeDefinition can be added to a package without a CommunityTemplateDefinition, but CommunityTemplateDefinition must contain a CommunityThemeDefinition to be added to a package.

### Documentation

*Salesforce Help:* [Export a Customized Experience Builder Theme for a Lightning Bolt Solution](#)

*Salesforce Help:* [Package and Distribute a Lightning Bolt Solution](#)


## Compact Layout

Represents the metadata associated with a compact layout.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- All attributes

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: CompactLayout

Component Type in 1GP Package Manager UI: Compact Layout

### Documentation

Metadata API Developer Guide: [CompactLayout](#)

## Conditional Formatting Ruleset

Represents a set of rules that define the style and visibility of conditional field formatting on Dynamic Forms-enabled Lightning page field instances.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Conditional formatting ruleset

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: UiFormatSpecificationSet

Component Type in 1GP Package Manager UI: UI Format Specification Set

### Relationship to Other Components

You can only assign a conditional formatting ruleset to a field on a Dynamic Forms-enabled [Lightning page](#).

**Documentation**

*Salesforce Help:* [Conditional Field Formatting in Lightning App Builder](#)

*Metadata API Developer Guide:* [UiFormatSpecificationSet](#)

## Connected App

Represents a connected app configuration. A connected app enables an external application to integrate with Salesforce using APIs and standard protocols, such as SAML, OAuth, and OpenID Connect.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Access Method
- Canvas App URL
- Callback URL
- Connected App Name
- Contact Email
- Contact Phone
- Description
- Icon URL
- Info URL
- Trusted IP Range
- Locations
- Logo Image URL
- OAuth Scopes

### Both Package Developer and Subscriber Can Edit

- ACS URL
- Entity ID
- IP Relaxation
- Mobile Start URL
- Permitted Users
- Refresh Token Policy
- SAML Attributes
- Service Provider Certificate
- Start URL
- Subject Type

#### Neither Package Developer or Subscriber Can Edit

- API Name
- Created Date/By
- Consumer Key
- Consumer Secret
- Installed By
- Installed Date
- Last Modified Date/By
- Version

## More Information

For details on packaging a connected app in 2GP managed packages, see [Package Connected Apps in Second-Generation Managed Packaging](#) in the *Second-Generation Managed Packaging Developer Guide*.

- Subscribers or installers of a package can't delete a connected app by itself, they can only uninstall the package. When a developer deletes a connected app from a package, the connected app is deleted in the subscriber's org during a package upgrade.
- To publish updates for a connected app that's part of a managed package, you typically push a new managed package version and upgrade subscriber orgs to the new version. But if you update a connected app's PIN Protect settings, it's not necessary to push a new managed package upgrade. After saving changes to PIN Protect settings, these updates are automatically published to subscriber orgs.
- The following connected app settings can't be updated with managed package patches.
  - Mobile App settings
  - Push messaging, including Apple, Android, and Windows push notifications
  - Canvas App settings
  - SAML settings

To update these settings, publish a new package version.

- If you push upgrade a package containing a connected app whose OAuth scope or IP ranges have changed from the previous version, the upgrade fails. This security feature blocks unauthorized users from gaining broad access to a customer org by upgrading an installed package. A customer can still perform a pull upgrade of the same package. This upgrade is allowed because it's with the customer's knowledge and consent.

- For connected apps created before Summer '13, the existing install URL is valid until you package and upload a new version. After you upload a new version of the package with an updated connected app, the install URL no longer works.

SEE ALSO:

[Package Connected Apps in Second-Generation Managed Packaging](#)

## Contract Type

A contract type is used to group contracts so that they exhibit similar characteristics. For example, the lifecycle states, the people who access, the templates and clauses used.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Both Package Developer and Subscriber Can Edit](#)

- Is Default
- Sub Types

[Neither Package Developer or Subscriber Can Edit](#)

- Name

## More Information

### Feature Name

Metadata Name: ContractType

### Use Case

Allows admin users to modify Contract Type properties.

### License Requirements

CLM Admin Permission Set (CLM User PSL).

### Documentation

*Salesforce Contracts Developer Guide:* [ContractType](#)


## Conversation Channel Definition

Represents the conversation channel definition that's implemented for Interaction Service for Bring Your Own Channel and Bring Your Own Channel for CCaaS messaging channels.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- Connected App
- Description
- Label
- Name

#### Both Package Developer and Subscriber Can Edit

- None

#### Neither Package Developer or Subscriber Can Edit

- None

### More Information

#### Feature Name

Metadata Name: ConversationChannelDefinition

Component Type in 1GP Package Manager UI: ConversationChannelDefinition

#### Use Case

To enable and set up Bring Your Own Channel, integrating third-party messaging services with Salesforce.



To enable and set up Bring Your Own Channel for Contact Center as a Service (CCaaS), integrating a third party CCaaS provider with Salesforce.

### License Requirements

Service Cloud license with Digital Engagement add-on license

### Post Install Steps

Set up and enable Bring Your Own Channel or Bring Your Own Channel for CCaaS.

### Relationship to Other Components

Linked to ConversationVendorInfo.

### Documentation

*Salesforce Developer Documentation:* [Bring Your Own Channel](#)

*Salesforce Developer Documentation:* [Bring Your Own Channel for CCaaS](#)

*Salesforce Help:* [Set Up Bring Your Own Channel](#)

*Salesforce Help:* [Set Up Bring Your Own Channel for CCaaS](#)

## Conversation Vendor Info

This setup object connects the partner vendor system to the Service Cloud feature.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ConversationVendorInfo

Component Type in 1GP Package Manager UI: ConversationVendorInfo

### Use Case

Include information about a Service Cloud Voice implementation.

### License Requirements

Enable Service Cloud Voice in your org.

### Documentation

*Service Cloud Voice for Partner Telephony Developer Guide:* [ConversationVendorInfo](#)

*Object Reference for the Salesforce Platform:* [ConversationVendorInfo](#)


## CORS Allowlist

Represents an origin in the CORS allowlist.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Url pattern

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: CorsWhitelistOrigin

Component Type in 1GP Package Manager UI: CORS Allowed Origin List

### Use Case

Customers can add a URL pattern that includes an HTTPS protocol and a domain name. Including a port number is optional. The wildcard character (\*) is supported only for the second-level domain name, for example, `https://*.example.com`.

### Documentation

*Salesforce Help:* [Enable CORS for OAuth Endpoints](#)

*Salesforce Help:* [Configure Salesforce CORS Allowlist](#)

## CSP Trusted Site

Represents a trusted URL. For each CspTrustedSite component, you can specify Content Security Policy (CSP) directives and permissions policy directives.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- context
- description
- endpointUrl
- isActive
- isApplicableToConnectSrc
- isApplicableToFontSrc
- isApplicableToFrameSrc
- isApplicableToImgSrc

- `isApplicableToMediaSrc`
- `isApplicableToStyleSrc`

Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: `CspTrustedSite`

Component Type in 1GP Package Manager UI: `CspTrustedSite`

### Use Case

The Lightning Component framework uses Content Security Policy (CSP) to impose restrictions on content. The main objective of CSP is to help prevent cross-site scripting (XSS) and other code injection attacks. If your package includes sites or pages that load content from an external (non-Salesforce) server or via a WebSocket connection, add the external server as a CSP trusted site. Each CSP trusted site can apply to Experience Cloud sites, Lightning Experience pages, custom Visualforce pages, or all three.

### Considerations When Packaging

When you include the `CspTrustedSite` component in a package, the permissions for the third-party APIs and Websocket connections apply to sites and pages across the org. Because this component modifies security, we don't recommend including `CspTrustedSite` components in packages. Instead, we recommend that you instruct customers to use the CSP Trusted Sites Setup page or the `CSPTrustedSites` metadata API type to add the URLs to their allowlist as part of activating your package. If you choose to include `CspTrustedSite` components in your package, disclose this change prominently in your package documentation to ensure that your customers are aware of the security modification.

You can't load JavaScript resources from a third-party site, even if it's a CSP Trusted Site. To use a JavaScript library from a third-party site, add it to a static resource, and then add the static resource to your component. After the library is loaded from the static resource, you can use it as normal.

CSP isn't enforced by all browsers. For a list of browsers that enforce CSP, see [caniuse.com](http://caniuse.com).

### Usage Limits

`CspTrustedSite` components are available in API version 39.0 and later. Multiple properties and enumeration values are available in later API versions. For details, see `CspTrustedSite` in the *Metadata API Developer Guide*.

For Experience Builder sites, if the HTTP header size is greater than 8 KB, the directives are moved from the CSP header to the `<meta>` tag. To avoid errors from infrastructure limits, ensure that the HTTP header size doesn't exceed 3 KB per context.

### Relationship to Other Components

This component can be used only in conjunction with an Aura or Lightning Web Runtime (LWR) page for an Experience Cloud site, a [Lightning Page](#), or a [Visualforce page](#).

### Documentation

*Salesforce Help:* [Manage CSP Trusted Sites](#)

*Metadata API Developer Guide:* [CspTrustedSites](#)


## Custom Application

Represents a custom application.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Show in Lightning Experience (Salesforce Classic only)
- Selected Items (Lightning Experience only)
- Utility Bar (Lightning Experience only)

### Both Package Developer and Subscriber Can Edit

- All attributes, except App Name and Show in Lightning Experience (Salesforce Classic only)
- All attributes, except Developer Name, Selected Items, and Utility Bar (Lightning Experience only)

### Neither Package Developer or Subscriber Can Edit

- App Name (Salesforce Classic only)
- Developer Name (Lightning Experience only)

## More Information

### Feature Name

Metadata Name: CustomApplication

### Documentation

Metadata API Developer Guide: [CustomApplication](#)


## Custom Button or Link

Represents a custom link defined in a home page component.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Behavior
- Button or Link URL
- Content Source
- Description
- Display Checkboxes
- Label
- Link Encoding

### Both Package Developer and Subscriber Can Edit

- Height
- Resizable
- Show Address Bar
- Show Menu Bar
- Show Scrollbars
- Show Status Bar
- Show Toolbars
- Width
- Window Position

### Neither Package Developer or Subscriber Can Edit

- Display Type

- Name

## More Information

### Feature Name

Metadata Name: WebLink, CustomPageWebLink

### Documentation

*Salesforce Help:* [Custom Buttons and Links](#)

## Custom Console Components

Represents a custom console component (Visualforce page) assigned to a CustomApplication that is marked as a Salesforce console. Custom console components extend the capabilities of Salesforce console apps.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

A package that has a custom console component can only be installed in an org with the Service Cloud license or Sales Console permission enabled.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

To confirm whether this component is available in managed 1GP, managed 2GP, or both package types, see [Metadata Coverage Report](#).



**Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: CustomApplicationComponent

Component Type in 1GP Package Manager UI: Custom Console Component

### Documentation

*Metadata API Developer Guide:* [CustomApplicationComponent](#)

*Salesforce Help:* [Create Console Components in Salesforce Classic](#)


## Custom Field on Standard or Custom Object

Represents the metadata associated with a field. Use this metadata type to create, update, or delete custom field definitions on standard or custom objects.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Auto-Number Display Format
- Decimal Places
- Description
- Default Value
- Field Label
- Formula
- Length



- Lookup Filter
- Related List Label
- Required
- Roll-Up Summary Filter Criteria

Both Package Developer and Subscriber Can Edit

- Chatter Feed Tracking
- Help Text
- Mask Type
- Mask Character
- Sharing Setting
- Sort Picklist Values
- Track Field History

Neither Package Developer or Subscriber Can Edit

- Child Relationship Name
- Data Type
- External ID
- Field Name
- Roll-Up Summary Field
- Roll-Up Summary Object
- Roll-Up Summary Type
- Unique

### More Information

- Developers can add required and universally required custom fields to managed packages as long as they have default values.
- Auto-number type fields and required fields can't be added after the object is included in a Managed - Released package.
- Subscriber orgs can't install roll-up summary fields that summarize detail fields set to *protected*.

## Custom Field on Custom Metadata Type

Represents a custom fields on the custom metadata type.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Custom Help Menu Section

Represents the section of the Lightning Experience help menu that the admin added to display custom, org-specific help resources for the org. The custom section contains help resources added by the admin.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## More Information

### Feature Name

Metadata Name: CustomHelpMenuSection

### Documentation

Metadata API Developer Guide: [CustomHelpMenuSection](#)

## Custom Index

Represents an index used to increase the speed of queries.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No. It can only be removed if the associated custom field is removed.

Component Has IP Protection	No
-----------------------------	----

## More Information

### Feature Name

Metadata Name: CustomIndex

Component Type in 1GP Package Manager UI: Custom Index

### Considerations When Packaging

Subscribers can remove the custom index using Metadata API only.

### Documentation

*Best Practices for Deployments with Large Data Volumes:* [Indexes](#)

## Custom Label

The CustomLabels metadata type allows you to create custom labels that can be localized for use in different languages, countries, and currencies.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No



**Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Category
- Short Description
- Value

### Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: CustomLabels

### Considerations When Packaging

If a label is translated, the language must be explicitly included in the package for the translations to be included in the package. Subscribers can override the default translation for a custom label.

This component can be marked as protected. For more details, see [Protected Components](#) in the *First-Generation Managed Packaging Developer Guide*.

### Documentation

Metadata API Developer Guide: [CustomLabels](#)


## Custom Metadata Type Records

Represents a record of a custom metadata type.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in managed 1GP if protected, and managed 2GP whether protected or not.
Component Has IP Protection	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## More Information

### Feature Name

Metadata Name: CustomObject

This component can be marked as protected. For more details, see [Protected Components](#) in the *First-Generation Managed Packaging Developer Guide*.

### Usage Limits

Deprecated custom metadata type records count against the subscriber's org limit. When removing custom metadata type records from a second-generation managed package, encourage subscribers to delete the deprecated records from their org. If the subscriber org reaches their org limit for custom metadata type records, package upgrades that include new custom metadata type records fail. For details see [Custom Metadata and Allocations and Usage Calculations](#) in *Salesforce Help*.

### Documentation

*Salesforce Help*: [Package Custom Metadata Types and Records](#)

## Custom Metadata Type

Represents a record of a custom metadata type.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes

### More Information

Second-generation managed packages (2GP) include the fields and records for custom metadata types that you add. You can't add fields directly to an existing package after the package version is promoted. If you create multiple packages that share a namespace, then layouts and records can be in separate packages. Custom fields on the custom metadata type must be in the same package.

You can add fields to a custom metadata type by publishing an extension to the existing package, creating an entity relationship field, and mapping the field to the custom metadata type in your extension. See [Add Custom Metadata Type Fields to Existing Packages](#).

This component can be marked as protected. For more details, see [Protected Components](#) in the *First-Generation Managed Packaging Developer Guide*.


## Custom Notification Type

Represents the metadata associated with a custom notification type.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Desktop, Mobile

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: CustomNotificationType

Component Type in 1GP Package Manager UI: Custom Notification Type

### License Requirements

Database.com editions don't have permission to access this component.

### Usage Limits

You can package up to 500 custom notification types, but keep in mind that subscriber orgs are limited to a total of 500 custom notification types. The subscriber org limit is shared across namespaces.

A subscriber org can execute up to 10,000 notification actions per hour.

### Documentation

*Salesforce Help:* [Create and Send Custom Desktop or Mobile Notifications](#)

*Salesforce Help:* [Considerations for Processes that Send Custom Notifications](#)


## Custom Object

Represents a custom object that stores data unique to an org or an external object that maps to data stored outside an org.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Label
- Plural Label
- Record Name
- Record Name Display Format
- Starts with a Vowel Sound

### Both Package Developer and Subscriber Can Edit

- Allow Activities
- Allow Reports
- Available for Customer Portal
- Context-Sensitive Help Setting
- Default Sharing Model
- Development Status
- Enable Divisions
- Enhanced Lookup
- Grant Access Using Hierarchy
- Search Layouts
- Track Field History

### Neither Package Developer or Subscriber Can Edit

- Object Name
- Record Name Data Type

## More Information

### Feature Name

Metadata Name: CustomObject

Component Type in 1GP Package Manager UI: Custom Object

### Considerations When Packaging

If a developer enables the `Allow Reports` or `Allow Activities` attributes on a packaged custom object, the subscriber's org also has these features enabled during a package upgrade. After it's enabled in a Managed - Released package, the developer and the subscriber can't disable these attributes.

Standard button and link overrides are also packageable.

In your extension package, if you want to access history information for custom objects contained in the base package, work with the base package owner to:

1. Enable history tracking in the release org of the base package.
2. Create a new version of the base package.
3. Install the new version of the base package in the release org of the extension package to access the history tracking info.

As a best practice, don't enable history tracking for custom objects contained in the base package directly in the extension package's release org. Doing so can result in an error when you install the package and when you create patch orgs for the extension package.

This component can be marked as protected. For more details, see [Protected Components](#) and [Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs](#) in the *First-Generation Managed Packaging Developer Guide*.

### Documentation

Metadata API Developer Guide: [CustomObject](#)

## Custom Object Translation

This metadata type allows you to translate custom objects for a variety of languages.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No



## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes except Description of WorkflowTask, Help of CustomField, PicklistValueTranslation, and MasterLabel of LayoutSection.

### Both Package Developer and Subscriber Can Edit

- Description of WorkflowTask
- Help of CustomField
- PicklistValueTranslation
- MasterLabel of LayoutSection

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: CustomObjectTranslation

### Relationship to Other Components

When you create a first-generation managed package and add the [Translation](#) component, the Custom Object Translation component is automatically added to your package.

When you create a second-generation managed package, you must add Custom Object Translation to your package, even if you've already added the Translation component.

### Documentation

Metadata API Developer Guide: [CustomObjectTranslation](#)


## Custom Permission

Represents a permission that grants access to a custom feature.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Connected App
- Description
- Label
- Name

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: CustomPermission

Component Type in 1GP Package Manager UI: Custom Permission

### Considerations When Packaging

If you deploy a change set with a custom permission that includes a connected app, the connected app must already be installed in the destination org.

This component can be marked as protected. For more details, see [Protected Components](#) and [Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs](#) in the *First-Generation Managed Packaging Developer Guide*.

### Documentation

*Metadata API Developer Guide:* [CustomPermission](#)

## Custom Tab


Represents a custom tab. Custom tabs let you display custom object data or other web content in Salesforce.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Encoding
- Has Sidebar
- Height
- Label
- S-control
- Splash Page Custom Link
- Type
- URL
- Width

### Both Package Developer and Subscriber Can Edit

- Tab Style

### Neither Package Developer or Subscriber Can Edit

- Tab Name

## More Information

### Feature Name

Metadata Name: CustomTab

### Considerations When Packaging

- The tab style for a custom tab must be unique within your app. However, it doesn't have to be unique within the org where it's installed. A custom tab style doesn't conflict with an existing custom tab in the installer's environment.
- To provide custom tab names in different languages, from Setup, in the Quick Find box, enter *Rename Tabs and Labels*, then select **Rename Tabs and Labels**.

### Documentation

*Metadata API Developer Guide:* [CustomTab](#)


## Dashboard

Represents a dashboard. Dashboards are visual representations of data that allow you to see key metrics and performance at a glance.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- None

#### Both Package Developer and Subscriber Can Edit

- All attributes except Dashboard Unique Name

#### Neither Package Developer or Subscriber Can Edit

- Dashboard Unique Name

### More Information

#### Feature Name

Metadata Name: Dashboard

Type in 1GP Package Manager UI: Dashboard

#### Considerations When Packaging

Developers of managed packages must consider the implications of introducing dashboard components that reference reports released in a previous version of the package. If the subscriber deleted the report or moved the report to a personal folder, the dashboard component referencing the report is dropped during the installation. Also, if the subscriber has modified the report, the report results can impact what displays in the dashboard component. As a best practice, release a dashboard and the related reports in the same version.

**Documentation**

Metadata API Developer Guide: [Dashboard](#)


## DataCalcInsightTemplate

Represents the object template for data calculations and insights of Data Cloud objects in DataCalcInsightTemplate. These objects are added inside the data kit.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes. Supported in 1GP packages only.
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

### More Information

**Feature Name**

Metadata Name: DataCalcInsightTemplate

Component Type in 1GP Package Manager UI: Calculated Insight Template

**Use Case**

DataCalcInsightTemplate represents the data calculations and insights for objects of a Data Cloud schema field the user includes in a data kit.

**Considerations When Packaging**

A Data Cloud feature is always packaged via a data kit. A calculated insight template is added to a package when you add a data calculation and insight into a data kit, and package that data kit. You can't directly add this component to a package.

**License Requirements**

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

**Post Install Steps**

After you install a package that contains a data kit, you must manually deploy the features from the installed data kit.

**Documentation**

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)

## Data Connector Ingest API

Represents the connection information specific to Ingestion API.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- None

**Both Package Developer and Subscriber Can Edit**

- None

**Neither Package Developer or Subscriber Can Edit**

- DeveloperName

### More Information

**Feature Name**

Metadata Name: DataConnectorIngestApi

Component Type in 1GP Package Manager UI: Adding DataStreamDefinition brings in DataConnectorIngestApi for Ingestion API DataStreams.

**Use Case**

This component is part of the Ingestion API Data stream metadata that is packaged and installed in subscriber.

**License Requirements**

You need Customer 360 Audiences Corporate (cdpPsl) licenses on both package developer org and subscriber org.

**Post Install Steps**

User has to create DataStream via ui-api or using the Data Cloud App.

**Documentation**

*Salesforce Help:* [Ingestion API](#)

## Data Connector S3

Represents the connection information specific to Amazon S3.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- None

#### Both Package Developer and Subscriber Can Edit

- Delimiter
- FileNameWildcard
- ImportFromDirectory
- S3AccessKey
- S3BucketName
- S3SecretKey

#### Neither Package Developer or Subscriber Can Edit

- DeveloperName

## More Information

### Feature Name

Metadata Name: DataConnectorS3

### Use Case

This includes the bucket details for the S3 connector in Data Cloud.

### Considerations When Packaging

To package this component, first add it to a data kit. For more information about data kits, see [Data Kits](#) in *Salesforce Help*.

Credentials are encrypted and need "IsDevInternal" permission for the encryption service.

### License Requirements

You need Customer 360 Audiences Corporate (cdpPsl) licenses on both package developer org and subscriber org.

### Post Install Steps

User has to create DataStream via ui-api or using the Data Cloud App.

### Documentation

*Salesforce Help*: [Data Connector S3](#)


## Data Kit Object Dependency

Represent the object dependencies and relationships between different objects in Data Kit Object Dependency. These objects are added inside the data kit.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes. Supported in 1GP packages only.
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None



[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: DataKitObjectDependency

Component Type in 1GP Package Manager UI: Data Kit Object Dependency

### Use Case

DataKitObjectDependency represents the relationship of objects of a Data Cloud schema field the user includes in a data kit.

### Considerations When Packaging

A Data Cloud feature is always packaged via a data kit. You add the external data transport field template to a data kit and then add that data kit to a package. You can't directly add this component to a package.

### License Requirements

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

### Post Install Steps

After you install a package that contains a data kit, you must manually deploy the features from the installed data kit.

### Documentation

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)

## Data Kit Object Template

Represents the object in Data Kit Object Template. This object template is added inside the data kit.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes. Supported in 1GP packages only.
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No



**Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: DataKitObjectTemplate

Component Type in 1GP Package Manager UI: Data Kit Object Dependency

### Use Case

DataKitObjectTemplate represents the objects the user includes in a data kit.

### Considerations When Packaging

A Data Cloud feature is always packaged via a data kit. You add the external data transport field template to a data kit, and then add that data kit to a package. You can't directly add this component to a package.

### License Requirements

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

### Post Install Steps

After you install a package that contains a data kit, you must manually deploy the features from the installed data kit.

### Documentation

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)

## Data Package Kit Definition


Represents the top-level Data Kit container definition. Content objects can be added after the Data Kit is defined.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes. Supported in 1GP packages only.
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- description
- developerName
- isDeployed
- isEnabled
- masterLabel
- versionNumber

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: DataPackageKitDefinition

Component Type in 1GP Package Manager UI: Data Package Kit Definition

### Use Case

Represents the top-level data kit container definition. Content objects can be added after the data kit is defined.

### License Requirements

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

### Post Install Steps

After you install a package that contains a data kit, you must manually deploy features from the installed data kit.

### Documentation

*Metadata API Developer Guide:* [DataPackageKitDefinition](#)

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)


## Data Package Kit Object

Represents the object in Data Kit Content Object. These objects are added inside the data kit.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes (supported only in 1GP packages)
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (supported only in 1GP packages)
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- parentDataPackageKitDefinitionName
- referenceObjectName
- referenceObjectType

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: DataPackageKitObject

Component Type in 1GP Package Manager UI: Data Package Kit Object

### Use Case

Represents an object in a data kit.

### License Requirements

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

### Post Install Steps

After you install a package that contains a data kit, you must manually deploy features from the installed data kit.

### Documentation

Metadata API Developer Guide: [DataPackageKitObject](#)

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)

## Data Source

Used to represent the system where the data was sourced.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- None

#### Both Package Developer and Subscriber Can Edit

- DataSourceStatus
- ExternalRecordIdentifier
- LastDataChangeStatusDateTime
- LastDataChangeStatusErrorCode

#### Neither Package Developer or Subscriber Can Edit

- DeveloperName

### More Information

#### Feature Name

Metadata Name: DataSource

#### Use Case

DataSource gives the lineage information of the datastream.

#### License Requirements

You need Customer 360 Audiences Corporate (cdpPsl) licenses on both package developer org and subscriber org.

#### Post Install Steps

Create DataSource using ui-api or the Data Cloud App.

**Relationship to Other Components**

This isn't a top-level entity. Add `DataStreamDefinition` or `DataKitDefinition` to pick up `DataSource`.

**Documentation**

*Salesforce Help:* [Connection Tasks in Data Cloud](#)


## Data Source Bundle Definition

Represents the bundle of streams that a user adds to a data kit.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes (supported only in 1GP packages)
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (supported only in 1GP packages)
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- `dataPlatform`
- `isMultiDeploymentSupported`
- `masterLabel`

**Both Package Developer and Subscriber Can Edit**

- None

**Neither Package Developer or Subscriber Can Edit**

- None

### More Information

**Feature Name**

Metadata Name: `DataSourceBundleDefinition`

Component Type in 1GP Package Manager UI: Data Source Bundle Definition

### Use Case

Represents the data stream data sources that a user adds to a data kit.

### Considerations When Packaging

Any Data Cloud feature is always packaged via a data kit. A data source bundle definition is added to a package when you add a data stream to a data kit and package that data kit. You can't directly add this component to a package.

### License Requirements

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

### Post Install Steps

After you install a package that contains a data kit, you must manually deploy features from the installed data kit.

### Documentation

*Metadata API Developer Guide:* [DataSourceBundleDefinition](#)

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)

## Data Source Object

Represents the object from where the data was sourced.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- DataObjectType
- DataSource
- ExternalRecordId

## More Information

### Feature Name

Metadata Name: DataSourceObject

### Use Case

DataSourceObject contains specific information about the source of the data like filename, table names.

### Considerations When Packaging

DataSourceObject pulls in child DataSourceField entity records when packaged with DataKitDefinition.

### License Requirements

Customer 360 Audiences Corporate (cdpPsl) licenses must be available on both package developer org and subscriber org.

### Post Install Steps

Create a DataStream via ui-api or using the Data Cloud App.

### Relationship to Other Components

This isn't a top-level entity. Add DataStreamDefinition or DataKitDefinition to pick up DataSourceObject.

### Documentation

*Salesforce Help:* [Connection Tasks in Data Cloud](#)


## Data Src Data Model Field Map

Represents the entity that's used to store the design-time bundle-level mappings for the data source fields and data model fields.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes (supported only in 1GP packages)
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (supported only in 1GP packages)
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- masterLabel



- sourceField
- targetField
- versionNumber

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: DataSrcDataModelFieldMap

Component Type in 1GP Package Manager UI: Data Source Data Model Field Mapping

### Use Case

Represents the entity that contains design-time bundle-level mappings for the data source fields and data model fields.

### Considerations When Packaging

Any Data Cloud feature is always packaged via a data kit. Data model field mappings are added to a package when you add a data stream and any associated mappings to a data kit and package that data kit. You can't directly add this component to a package.

### License Requirements

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

### Post Install Steps

After you install a package that contains a data kit, you must manually deploy features from the installed data kit.

### Documentation

*Metadata API Developer Guide:* [DataSrcDataModelFieldMap](#)

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)

## Data Stream Definition

Contains Data Ingestion information such as Connection, API and File retrieval settings.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- AreHeadersIncludedInTheFiles
- BulkIngest
- Description
- IsLimitedToNewFiles
- IsMissingFileFailure

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- DataConnectionGCS
- DataConnectorType
- DataExtractField
- DataExtractMethod
- DataExtractField
- DataPlatformDataSetBundle
- FileNameWildcard
- MktDataLakeObject
- MktDataTranObject

## More Information

### Feature Name

Metadata Name: DataStreamDefinition

Component Type in 1GP Package Manager UI: DataStreamDefinition

### Use Case

DataStreamDefinition is the starting point for packaging a Datastream and its mappings.

### Considerations When Packaging

Data Cloud admin user can install or upgrade the package. Admin User or Data Aware Specialist User can create Datastreams out of the installed package.

### License Requirements

Customer 360 Audiences Corporate (cdpPsl) licenses must be available on both package developer org and subscriber org. CDP Admin User can install, upgrade, or uninstall the package.

### Post Install Steps

Create the DataStream via ui-api or using the Data Cloud App.

### Documentation

Metadata API Developer Guide: [DataStreamDefinition](#)


## Data Stream Template

Represents the data stream that a user adds to a data kit.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes (supported only in 1GP packages)
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (supported only in 1GP packages)
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- dataImportRefreshFrequency
- dataSourceBundleDefinition
- dataSourceObject
- objectCategory
- refreshFrequency
- refreshHours
- refreshMode

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: DataStreamTemplate

Component Type in 1GP Package Manager UI: Data Stream Template

### Use Case

Represents the data stream that a user adds to a data kit.

**Considerations When Packaging**

Any Data Cloud feature is always packaged via a data kit. A data stream template is added to a package when you add a data stream to a data kit and package that data kit. You can't directly add this component to a package.

**License Requirements**

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

**Post Install Steps**

After you install a package that contains a data kit, you must manually deploy features from the installed data kit.

**Documentation**

*Metadata API Developer Guide:* [DataStreamTemplate](#)

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)


## DataWeaveResource

Represents the DataWeaveScriptResource class that is generated for all DataWeave scripts. DataWeave scripts can be directly invoked from Apex.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (if not set to <code>global</code> access).
Component Has IP Protection	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- API Version
- DataWeave Script

**Both Package Developer and Subscriber Can Edit**

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

**Feature Name**

Metadata Name: DataWeaveResource

Component Type in 1GP Package Manager UI: DataWeaveResource

**Use Case**

Include MuleSoft DataWeave scripts to read and parse data from one format, transform it, and export it in a different format directly from Apex.

**Considerations When Packaging**

There's a maximum of 50 DataWeave scripts per org.

**Documentation**

*Apex Developer Guide:* [DataWeave in Apex](#).


## Decision Matrix Definition

Represents a definition of a decision matrix.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes. Only if the component is inactive.
Subscriber Can Delete Component From Org	Yes. Only if the component is inactive.
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Type
- GroupKey
- SubGroupKey

[Both Package Developer and Subscriber Can Edit](#)

- versions

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: DecisionMatrixDefinition

Component Type in 1GP Package Manager UI: Decision Matrix Definition

### Use Case

Decision matrices are lookup tables that match input values to a matrix row and return the row's output values. Expression sets and various digital procedures can call decision matrices. Decision matrices accept JSON input from, and return JSON output to the digital processes that call the matrices. Decision matrices are useful for implementing complex rules in a systematic, readable manner.

### Documentation

*Industries Common Resources Developer Guide:* [Decision Matrix Definition](#)

*Salesforce Help:* [Decision Matrices](#)

*Salesforce Help:* [Decision Matrix Migration Considerations](#)


## Decision Matrix Definition Version

Represents a definition of a decision matrix version.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes. Only if the component is inactive.
Subscriber Can Delete Component From Org	Yes. Only if the component is inactive.
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- columns

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: DecisionMatrixDefinitionVersion

Component Type in 1GP Package Manager UI: Decision Matrix Definition Version

### Post Install Steps

After migrating a decision matrix version, upload the row data to the active version manually. The row data isn't migrated as part of the migration.

### Relationship to Other Components

A DecisionMatrixDefinitionVersion is a child of DecisionMatrixDefinition, and can't exist without the parent DecisionMatrixDefinition.

### Documentation

*Industries Common Resources Developer Guide:* [Decision Matrix Definition](#)

*Salesforce Help:* [Decision Matrices](#)

*Salesforce Help:* [Decision Matrix Migration Considerations](#)

## Decision Table

Represents the information about a decision table.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes, except templates

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Decision Table

[Both Package Developer and Subscriber Can Edit](#)

- Label
- Description
- Status

[Neither Package Developer or Subscriber Can Edit](#)

- API Name
- URL

## More Information

**Feature Name**

Metadata Name: DecisionTable

Component Type in 1GP Package Manager UI: Decision Table

**Use Case**

Decision tables read business rules and decide the outcome for records in your Salesforce org or for the values that you specify.

**License Requirements**

Either Loyalty Management or Rebate Management

**Documentation***Salesforce Help:* [Decision Tables](#)

## Decision Table Dataset Link

Represents the information about a dataset link associated with a decision table. In a dataset link, select an object for whose records, the decision table must provide an outcome.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes, except templates

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Dataset Link record

[Both Package Developer and Subscriber Can Edit](#)



- Label
- Description
- Status

[Neither Package Developer or Subscriber Can Edit](#)

- API Name
- URL

## More Information

### Feature Name

Metadata Name: DecisionTableDatasetLink

### Use Case

In a dataset link, you can map the decision table's input fields with fields of different standard or custom objects.

### License Requirements

Either Loyalty Management or Rebate Management

### Documentation

*Salesforce Help:* [Add Dataset Links to a Decision Table](#)

## Digital Experience

Represents a text-based code structure of your organization's workspaces, organized by workspace type, and each workspace's content items.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Content Title
- Content Body
- Content Folder

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

**Feature Name**

Metadata Name: DigitalExperience

**Use Case**

To move Digital Experience metadata Content from one org to another

**Post Install Steps**

After the package is installed, publish the site to make it available to customers.

**Documentation***Salesforce Help:* [CMS Content](#)

## Digital Experience Bundle

Represents a text-based code structure of your organization's workspaces, organized by workspace type, and each workspace's content items.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Labels
- Description
- Content

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: DigitalExperienceBundle

### Use Case

Share or distribute the content of a marketing workspace in Marketing Cloud, including landing pages, forms, and emails (and their associated images and branding).

### Considerations When Packaging

You can package only marketing workspaces. Packaging for general workspaces is unsupported. For example, let's say you share a general workspace with a marketing workspace. If content in the marketing workspace references an image that's shared from the general workspace, you can't successfully package the marketing workspace.

Enhanced LWR sites are also unsupported.

### Post Install Steps

After the package is installed, publish the workspace content to make it available to customers.

### Documentation

*Salesforce Help:* [Marketing Cloud](#)

*Metadata API Developer Guide:* [DigitalExperienceBundle](#)

## Decision Table

Represents the information about a decision table.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes, except templates

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Decision Table

### Both Package Developer and Subscriber Can Edit

- Label
- Description
- Status

[Neither Package Developer or Subscriber Can Edit](#)

- API Name
- URL

## More Information

**Feature Name**

Metadata Name: DecisionTable

Component Type in 1GP Package Manager UI: Decision Table

**Use Case**

Decision tables read business rules and decide the outcome for records in your Salesforce org or for the values that you specify.

**License Requirements**

Either Loyalty Management or Rebate Management

**Documentation***Salesforce Help:* [Decision Tables](#)

## Disclosure Definition

Represents information that defines a disclosure type, such as details of the publisher or vendor who created or implemented the report.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All attributes

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: DisclosureDefinition

Component Type in 1GP Package Manager UI: Disclosure Definition

### Use Case

You can use this component to define a disclosure type, such as details of the publisher or vendor who created or implemented the report.

### License Requirements

- Net Zero Cloud Growth license
- Disclosure and Compliance Hub permission set license
- Disclosure and Compliance Hub User permission set

### Post Install Steps

Enable these org settings:

- Manage Disclosure and Compliance Hub

### Documentation

- *Salesforce Help:* [Disclosure and Compliance Hub](#)
- *Salesforce Help:* [Generate Disclosures Using Disclosure and Compliance Hub](#)
- *Metadata API Developer Guide:* [DisclosureDefinition](#)

## Disclosure Definition Version

Represents the version information about the disclosure definition.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- DisclosureDefinition
- Description

- IsActive
- VersionNumber
- OmniScriptCnfgApiName
- IsCurrentVersion
- DisclosureDefCurrVer

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: DisclosureDefinitionVersion

Component Type in 1GP Package Manager UI: Disclosure Definition Version

### Use Case

You can use this component to define the version information about the disclosure definition.

### License Requirements

- Net Zero Cloud Growth license
- Disclosure and Compliance Hub permission set license
- Disclosure and Compliance Hub User permission set

### Post Install Steps

Enable these org settings:

- [Manage Disclosure and Compliance Hub](#)

### Documentation

- *Salesforce Help:* [Disclosure and Compliance Hub](#)
- *Salesforce Help:* [Generate Disclosures Using Disclosure and Compliance Hub](#)
- *Metadata API Developer Guide:* [DisclosureDefinitionVersion](#)

## Disclosure Type

Represents the types of disclosures that are done by an individual or an organization and the associated metadata.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes

Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: DisclosureType

Component Type in 1GP Package Manager UI: Disclosure Type

### Use Case

You can use this component to create types of disclosures that are done by an individual or an organization.

### License Requirements

- Net Zero Cloud Growth license
- Disclosure and Compliance Hub permission set license
- Disclosure and Compliance Hub User permission set

### Post Install Steps

Enable these org settings:

- Manage Disclosure and Compliance Hub

### Documentation

- *Salesforce Help*: [Disclosure and Compliance Hub](#)
- *Salesforce Help*: [Generate Disclosures Using Disclosure and Compliance Hub](#)
- *Metadata API Developer Guide*: [DisclosureType](#)


## Discovery AI Model

Represents the metadata associated with a model used in Einstein Discovery.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- All attributes except Discovery AI Model Unique Name

### Neither Package Developer or Subscriber Can Edit

- Discovery AI Model Unique Name

## More Information

### Feature Name

Metadata Name: DiscoveryAIModel

### Documentation

*Metadata API Developer Guide:* [DiscoveryAIModel](#)

## Discovery Goal

Represents the metadata associated with an Einstein Discovery prediction definition.


## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No



Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- All attributes except Discovery Goal Unique Name

### Neither Package Developer or Subscriber Can Edit

- Discovery Goal Unique Name

## More Information

### Feature Name

Metadata Name: DiscoveryGoal

### Documentation

*Metadata API Developer Guide:* [DiscoveryGoal](#)


## Discovery Story

Represents the metadata associated with a story used in Einstein Discovery.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- All attributes except Discovery Story Unique Name

[Neither Package Developer or Subscriber Can Edit](#)

- Discovery Story Unique Name

## More Information

### Feature Name

Metadata Name: DiscoveryStory

### Documentation

Metadata API Developer Guide: [DiscoveryStory](#)


## Document

Represents a Document. All documents must be in a document folder, such as sampleFolder/TestDocument.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## More Information

### Feature Name

Metadata Name: Document

Component Type in 1GP Package Manager UI: Document

### Documentation

*Metadata API Developer Guide:* [Document](#)

## Document Generation Setting

Represents an org's settings for automatic document generation from templates.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Both Package Developer and Subscriber Can Edit](#)

- Document Template Library Name
- Generation Mechanism
- Guest Access Named Credential
- Label
- Preview Type

[Neither Package Developer or Subscriber Can Edit](#)

- API Name

## More Information

### Feature Name

Metadata Name: DocumentGenerationSetting

**Use Case**

Allows admin users to modify document generation properties.

**License Requirements**

DocGen Designer (Permission Set License)

**Documentation**

*Metadata API Developer Guide:* [DocumentGenerationSetting](#)


## Eclair GeoData

Represents an Analytics custom map chart. Custom maps are user-defined maps that are uploaded to Analytics and are used just as standard maps are. Custom maps are accessed in Analytics from the list of maps available with the map chart type.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- None

**Both Package Developer and Subscriber Can Edit**

- All attributes except Eclair GeoData Unique Name

**Neither Package Developer or Subscriber Can Edit**

- Eclair GeoData Unique Name

## More Information

**Feature Name**

Metadata Name: EclairGeoData

**Documentation**

Metadata API Developer Guide: [EclairGeoData](#)

## Email Template (Classic)

Use email templates to increase productivity and ensure consistent messaging. Email templates with merge fields let you quickly send emails that include field data from Salesforce records.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- None

**Both Package Developer and Subscriber Can Edit**

- All attributes except Email Template Name

**Neither Package Developer or Subscriber Can Edit**

- Email Template Name

## Email Template (Lightning)

Represents a template for an email, mass email, list email, or Sales Engagement email.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
-----------------	---

Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only. However, 1GP packages created in Email Template Builder can't be removed.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None


### Neither Package Developer or Subscriber Can Edit

- All attributes

## More Information

These packaging considerations apply to Lightning email templates, including email templates created in Email Template Builder.

- For email templates created in Email Template Builder before the Spring '21 release, 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.
- Enhanced email template folders have these behaviors:
  - If a package includes an enhanced email template folder, the target organization must have enhanced folders enabled for the deploy to succeed.
  - If an email template is in a subfolder, adding the root folder to a package doesn't automatically add the email template to the package. If the email template is in the root folder, it's automatically added to the package.
  - You can't package an email template in the default public and private folders.
- For merge fields based on custom fields that are used in the Recipients prefix (for leads and contacts), we add references to those merge fields. If the custom field is renamed, the reference in the template isn't updated. Edit the custom merge field to use the new field name and update the reference.

 **Note:** An email template created in Email Template Builder can't be edited after it's downloaded. To edit the template, clone it. When upgrading a package that has Email Template Builder email templates, only the associated FlexiPage is updated. After downloading the new version of the template, clone it to see the changes.

## Embedded Service Config

Represents a setup node for creating an Embedded Service for Web deployment.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: EmbeddedServiceConfig

### Documentation

*Metadata API Developer Guide:* [EmbeddedServiceConfig](#)

*Salesforce Help:* [Embedded Chat](#)

## Embedded Service Menu Settings

Represents a setup node for creating a channel menu deployment. Channel menus list the ways in which customers can contact your business.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: EmbeddedServiceMenuSettings

### Documentation

*Metadata API Developer Guide:* [EmbeddedServiceMenuSettings](#)

*Salesforce Help:* [Channel Menu Setup](#)


## Enablement Measure Definition

Represents an Enablement measure, which specifies the job-related activity that a user performs to complete a milestone or outcome in an Enablement program. A measure identifies a source object and optional related objects, with optional field filters and filter logic, for tracking the activity.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).



## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All but Status and DeveloperName

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- DeveloperName

## More Information

### Feature Name

Metadata Name: EnablementMeasureDefinition

### Use Case

Include this component in a package with a program if the program has outcomes or milestones.

### Considerations When Packaging

See [Considerations for Packaging Enablement Programs and Dependencies](#).

### License Requirements

Enablement add-on license and the Enablement permission set license are required. For Partner Enablement programs in supported Experience Cloud sites, a [supported Partner Relationship Management \(PRM\) add-on license](#) is also required.

### Usage Limits

See [Enablement Limits](#).

### Relationship to Other Components

An Enablement measure is used within an Enablement program. Package the Enablement Measure Definition component with the Enablement Program Definition component. Or, package the Enablement Measure Definition component separately. Each measure references a source object and optional related objects.

### Documentation

- *Salesforce Help:* [Sales Programs and Partner Tracks with Enablement](#)
- *Metadata API Developer Guide:* [EnablementMeasureDefinition](#)
- *Sales Programs and Partner Tracks with Enablement Developer Guide:* [Create a Managed Package for Enablement Programs, Measures, and Content](#)

## Enablement Program Definition

Represents an Enablement program, which includes exercises and measurable milestones to help users such as sales reps achieve specific outcomes related to your company's revenue goals.


## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:

Second-Generation Managed Packages (2GP)

Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- All but DeveloperName

### Neither Package Developer or Subscriber Can Edit

- DeveloperName

## More Information

### Feature Name

Metadata Name: EnablementProgramDefinition

### Use Case

Include this component in a package when you want to move a program from one org to another.

### Considerations When Packaging

See [Considerations for Packaging Enablement Programs and Dependencies](#).

### License Requirements

Enablement add-on license and the Enablement permission set license are required. For Partner Enablement programs in supported Experience Cloud sites, a [supported Partner Relationship Management \(PRM\) add-on license](#) is also required.

### Usage Limits

See [Enablement Limits](#).

### Relationship to Other Components

An Enablement program can contain other items that are related to other packageable components. Package the Enablement Program Definition component with other appropriate components.

- Exercises that reference Digital Experiences content. Package the Digital Experience component.
- Exercises that reference assessment surveys. Package the Flow component.
- Custom exercise types that reference user-defined content. Package the Learning Item Type and Enablement Program Task Subcategory components.
- Measures that track job-related activity using specific objects. Package the Enablement Measure Definition component.

**Documentation**

- [Salesforce Help: Sales Programs and Partner Tracks with Enablement](#)
- [Metadata API Developer Guide: EnablementMeasureDefinition](#)
- [Sales Programs and Partner Tracks with Enablement Developer Guide: Create a Managed Package for Enablement Programs, Measures, and Content](#)


## Enablement Program Task Subcategory

Represents a custom exercise type that an Enablement admin adds to an Enablement program in Program Builder. A custom exercise type also requires a corresponding EnblProgramTaskDefinition record for Program Builder and corresponding LearningItem and LearningItemType records for when users take the exercise in the Guidance Center.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- All but DeveloperName

**Both Package Developer and Subscriber Can Edit**

- None

**Neither Package Developer or Subscriber Can Edit**

- DeveloperName

### More Information

**Feature Name**

Metadata Name: EnblProgramTaskSubCategory

**Use Case**

Include this component in a package with a program if the program has a custom exercise type.

**Considerations When Packaging**

See [Considerations for Packaging Enablement Programs and Dependencies](#).

**License Requirements**

Enablement add-on license and the Enablement permission set license are required.

 **Important:** Custom exercises aren't compatible with Partner Enablement programs.

**Usage Limits**

See [Enablement Limits](#).

**Relationship to Other Components**

The Enablement Program Task Subcategory component requires a corresponding Learning Item Type component. Both components are used with custom exercise types in Enablement programs. Package both of these components with an Enablement Program Definition component.

**Documentation**

- *Salesforce Help: Sales Programs and Partner Tracks with Enablement*
- *Metadata API Developer Guide: [EnblProgramTaskSubCategory](#)*
- *Metadata API Developer Guide: [LearningItemType](#)*
- *Object Reference for the Salesforce Platform: [EnblProgramTaskDefinition](#)*
- *Object Reference for the Salesforce Platform: [LearningItem](#)*
- *Sales Programs and Partner Tracks with Enablement Developer Guide: [Create a Managed Package for Enablement Programs, Measures, and Content](#)*
- *Sales Programs and Partner Tracks with Enablement Developer Guide: [Implement Custom Exercise Types for Enablement Programs](#)*

## Entitlement Template

Represents an entitlement template. Entitlement templates are predefined terms of customer support that you can quickly add to products.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: EntitlementTemplate

### Documentation

Metadata API Developer Guide: [EntitlementTemplate](#)

Salesforce Help: [Set Up an Entitlement Template](#)

## ESignature Config

Using the Electronic Signature Configuration setup, the system admin must define the required configurations to support the e-signature APIs and UI.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Both Package Developer and Subscriber Can Edit

- Config Type
- Config Value
- Description
- Group Type
- Vendor

[Neither Package Developer or Subscriber Can Edit](#)

- DeveloperName
- MasterLabel

## More Information

**Feature Name**

Metadata Name: ESignatureConfig

**Use Case**

Allows users to get the electronic signatures on their documents.

**License Requirements**

DocGen Designer (Permission Set License)

## ESignature Envelope Config

Using the Electronic Signature Envelope Config the system admin can define the default reminders and expiry for the envelopes submitted for eSignature.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Both Package Developer and Subscriber Can Edit](#)

- Expiration Enabled
- Expiration Period
- Expiration Warning Period
- First Reminder Period
- Reminder Enabled
- Reminder Interval Period
- Target Object Name
- Vendor
- Vendor Account Identifier

- Vendor Default Notification Enabled

[Neither Package Developer or Subscriber Can Edit](#)

- DeveloperName
- MasterLabel

## More Information

### Feature Name

Metadata Name: ESignatureEnvelopeConfig

### Use Case

Allows users to get the electronic signatures and notifications on their documents.

### License Requirements

DocGen Designer (Permission Set License)

### Documentation

*Metadata API Developer Guide:* [ESignatureEnvelopeConfig](#)


## Explainability Action Definition

Define where the metadata for your Decision Explorer business rules are stored in Public Sector Solutions.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- Label

- Description
- Developer Name
- Business Process Type
- Application Type
- Action Log Schema Type
- Application Subtype

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: ExplainabilityActionDefinition


## Explainability Action Version

Define and store versions of the explainability actions used by your Decision Explainer business rules in Public Sector Solutions.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- Label
- Active
- Description



- Explainability Action Definition

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: ExplainabilityActionVersion


## Explainability Message Template

Represents information about the template that contains the decision explanation message for a specified expression set step type.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- Label
- Message
- Name
- Result Type
- Default
- Expression Set Step Type

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: ExplainabilityMsgTemplate

### Documentation

*Industries Common Resources Developer Guide:* [ExplainabilityMsgTemplate](#)

*Salesforce Help:* [Create Explainability Message Templates](#)


## Expression Set Definition

Represents an expression set definition.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes. Only if the component doesn't contain any active versions.
Subscriber Can Delete Component From Org	Yes. Only if the component doesn't contain any active versions.
Package Developer Can Remove Component From Package	Yes. Only if the component doesn't contain any active versions.

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- versions

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ExpressionSetDefinition

Component Type in 1GP Package Manager UI: ExpressionSet Definition

**Relationship to Other Components**

To use this component, any expression set version dependencies such as decision matrices, decision tables, object field aliases, and subexpressions must be present in the target org.

**Documentation**

*Industries Common Resources Developer Guide:* [Expression Set Definition](#)

*Salesforce Help:* [Expression Set Migration Considerations](#)


## Expression Set Definition Version

Represents a definition of an expression set version.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes. Only if the component is in an inactive state.
Subscriber Can Delete Component From Org	Yes. Only if the component is in an inactive state.
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- None

**Both Package Developer and Subscriber Can Edit**

- variables
- steps

**Neither Package Developer or Subscriber Can Edit**

- None

### More Information

**Feature Name**

Metadata Name: ExpressionSetDefinitionVersion

Component Type in 1GP Package Manager UI: Expression Set Definition Version

### Relationship to Other Components

This component can be used only if the ExpressionSetDefinition to which this ExpressionSetDefinitionVersion component belongs is present in the target org.

To use this component, any expression set version dependencies such as decision matrices, decision tables, object field aliases, and subexpressions must be present in the target org.

### Documentation

*Industries Common Resources Developer Guide:* [Expression Set Definition Version](#)

*Salesforce Help:* [Expression Set Migration Considerations](#)

## Expression Set Object Alias

Represents information about the alias of the source object that's used in an expression set.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- mappings.sourceFieldName
- mappings.fieldAlias

#### Both Package Developer and Subscriber Can Edit

- None

#### Neither Package Developer or Subscriber Can Edit

- objectApiName
- usageType
- dataType

### More Information

#### Feature Name

Metadata Name: ExpressionSetObjectAlias

Component Type: Expression Set Object Alias

**Use Case**

Expression set object aliases allow you to use object fields as variables in expression sets. Aliases are relevant and user-friendly names that are created for underlying source object fields. Field aliases are grouped under an object alias.

**Documentation**

*Industries Common Resources Developer Guide:* [ExpressionSetObjectAlias](#)

*Salesforce Help:* [Object Variables in Expression Sets](#)


## Expression Set Message Token

Represents a token that's used in an explainability message template. The token can be replaced with an expression set version resource that the template is used in. This object is available in API version 59.0 and later.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- Master Label
- Developer Name
- Description

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ExpressionSetMessageToken

Component Type in 1GP Package Manager UI: ExpressionSetMessageToken

### Documentation

*Industries Common Resources Developer Guide:* [ExpressionSetMessageToken](#)

*Salesforce Help:* [Create Expression Set Message Tokens](#)

## External Client App Header

Represents the header file for an external client application configuration.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All properties

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: ExternalClientApplication

### Considerations When Packaging

Unlike most metadata, External Client Apps can't be created via the Setup menu in a scratch org. ISVs who intend to package External Client Apps in a managed 2GP should instead define the External Client App in their PBO (Partner Business Org) Dev Hub. The External Client App can then be retrieved via Salesforce CLI and deployed into a scratch org, or packaged and installed into a scratch org for testing. See [Configure Packageable External Client Apps](#) for more information.

**Relationship to Other Components**

ExternalClientApplication is the header file for an external client app. This defines the basic configurations of the external client app, including whether the external client app can be packaged or if it is developed for local use only.

ExtlClntAppGlobalOAuthSettings includes sensitive information for the External Client Apps OAuth plugin, like OAuth consumer credentials that can't be packaged or added to source control. ExtlClntAppOAuthSettings includes packageable configurations. All settings are determined by the developer and can't be edited by the admin. Admin-controlled configurations are called policies and are included in ExtlClntAppOAuthConfigurablePolicies.

**Documentation**

*Salesforce Help:* [External Client Apps](#)

*Salesforce Help:* [Configure Packageable External Client Apps](#)


## External Client App OAuth Settings

Represents the settings configuration for the external client app's OAuth plugin.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

### Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- All properties

**Both Package Developer and Subscriber Can Edit**

- None

**Neither Package Developer or Subscriber Can Edit**

- None

### More Information

**Feature Name**

Metadata Name: ExtlClntAppOAuthSettings

### Considerations When Packaging

Unlike most metadata, External Client Apps can't be created via the Setup menu in a scratch org. ISVs who intend to package External Client Apps in a managed 2GP should instead define the External Client App in their PBO (Partner Business Org) Dev Hub. The External Client App can then be retrieved via Salesforce CLI and deployed into a scratch org, or packaged and installed into a scratch org for testing. See [Configure Packageable External Client Apps](#) for more information.

### Relationship to Other Components

External Client App plugins like the OAuth plugin include two types of configurations: settings and policies. All settings are determined by the external client app developer and can't be edited by the admin for the subscriber org. Admin-controlled configurations are called policies.

ExtlClntAppOAuthSettings contains all of the packageable configurations for the External Client Apps OAuth plugin. Sensitive information, like OAuth consumer credentials that can't be packaged or added to source control, are stored in the ExtlClntAppGlobalOAuthSettings. Policies are saved in ExtlClntAppOAuthConfigurablePolicies, which is not packaged but is generated with default values at runtime.

### Documentation

*Salesforce Help:* [External Client Apps](#)


## External Credential

Represents the details of how Salesforce authenticates to the external system.

## Component Manageability Rules


Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

 **Note:** In addition to these properties, the Description, ParameterGroup, ParameterName, ParameterValue, and SequenceNumber properties have the same editability as the ExternalCredentialParameters they're included in.

[Only Package Developer Can Edit](#)



- Label
- AuthenticationProtocol
- ExternalCredentialParameters
  - AuthProtocolVariant

#### Both Package Developer and Subscriber Can Edit

- Description
- ExternalCredentialParameters
  - AuthHeader
  - AuthProvider (only subscriber editable in 2GP)
  - AuthProviderUrl
  - AuthProviderUrlQueryParameter
  - AuthParameter
  - AwsStsPrincipal (only for external credentials that use AWS Signature v4 authentication with STS)
  - Description
  - JwtBodyClaim
  - JwtHeaderClaim
  - NamedPrincipal
  - PerUserPrincipal
  - SequenceNumber
  - SigningCertificate (only subscriber editable in 2GP)

#### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: `ExternalCredential`

### Considerations When Packaging

Though named and external credentials are represented by metadata, the standard Metadata API can't fully expose the definition of a credential and render sensitive information like tokens in plain text. This means that packaged named credentials don't include the access tokens or certificates that are needed to perform authenticated callouts. You can create the external credential's principal or populate its tokens or certificates in the UI or via the Connect API.

In managed 1GP packages, 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.

In managed 2GP packages, if an external credential uses an authentication provider to capture the details of the authorization endpoint, you can't include the reference to the authentication provider in the package. If the external credential references an authentication provider, you must recreate the authentication provider in the subscriber org and add it to the external credential.

### Post Install Steps

After installing an external credential from a managed or unmanaged package, you must:

- Create the external credential's principal or populate its tokens or certificates in the UI or via the Connect API.
- Give permission sets and profiles access to the principals of the external credential. See [Enable External Credential Principals](#).
- Reauthenticate to the external system.
  - For a Named Principal, the admin must go to **Setup > Named Credential > External Credential** to authenticate.
  - For a Per User Principal, each user must go to **My Personal Information > External Credential** to authenticate.

### Relationship to Other Components

ExternalCredential can be added to a package without a NamedCredential, but NamedCredential must be packaged with an ExternalCredential.

The named credential defines a callout endpoint and an HTTP transport protocol, while the external credential represents the details of how Salesforce authenticates to an external system via an authentication protocol. Each named credential must be mapped to at least one external credential.

### Documentation

*Salesforce Help:* [Named Credentials](#)

*Named Credentials Developer Guide:* [Named Credentials Packaging Guide](#)

*Metadata API Developer Guide:* [ExternalCredential](#)

## External Data Connector

Used to represent the object where the data was sourced.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- DataConConfiguration
- DataConnectionStatus

- DataConnectorType
- DataPlatform
- ExternalRecordId

## More Information

### Feature Name

Metadata Name: ExternalDataConnector

Component Type in 1GP Package Manager UI: Adding DataStreamDefinition or DataKitDefinition brings ExternalDataConnector for S3 data streams.

### Use Case

This component holds reference to Source Data Connector Metadata.

### License Requirements

You need Customer 360 Audiences Corporate (cdpPsl) licenses on both package developer org and subscriber org.

### Post Install Steps

User has to create DataStream via ui-api or using the Data Cloud App.

### Relationship to Other Components

This isn't a top-level entity. Add DataStreamDefinition or DataKitDefinition to pick up this entity.

## External Data Source

Represents the metadata associated with an external data source. Create external data sources to manage connection details for integration with data and content that are stored outside your Salesforce org.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Type

### Both Package Developer and Subscriber Can Edit

- Auth Provider

- Certificate
- Custom Configuration
- Endpoint
- Identity Type
- OAuth Scope
- Password
- Protocol
- Username

[Neither Package Developer or Subscriber Can Edit](#)

- Name

## More Information

### Feature Name

Metadata Name: ExternalDataSource

Component Type in 1GP Package Manager UI: External Data Source

### Considerations When Packaging

- After installing an external data source from a managed or unmanaged package, the subscriber must reauthenticate to the external system.
  - For password authentication, the subscriber must reenter the password in the external data source definition.
  - For OAuth, the subscriber must update the callback URL in the client configuration for the authentication provider, then reauthenticate by selecting `Start Authentication Flow on Save` on the external data source.
- Certificates aren't packageable. If you package an external data source that specifies a certificate, make sure that the subscriber org has a valid certificate with the same name.

### Documentation

Metadata API Developer Guide: [ExternalDataSource](#)

## External Data Transport Field Template

Represents the definition of a Data Cloud schema field.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes (supported only in 1GP packages)
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (supported only in 1GP packages)

Component Has IP Protection

No

---

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- DataSourceField
- ExternalDataTranField
- ExternalName
- IsDataRequired

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ExtDataTranFieldTemplate

Component Type in 1GP Package Manager UI: External Data Transport Field Template

### Use Case

ExtDataTranFieldTemplate represents the definition of a Data Cloud schema field the user includes in a data kit.

### Considerations When Packaging

A Data Cloud feature is always packaged via a data kit. You add the external data transport field template to a data kit and then add that data kit to a package. You can't directly add this component to a package.

### License Requirements

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

### Post Install Steps

After you install a package that contains a data kit, you must manually deploy the features from the installed data kit.

### Documentation

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)

## External Data Transport Object Template

Represents the definition of a Data Cloud schema object.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes (supported only in 1GP packages)
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (supported only in 1GP packages)
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- DataSourceObject
- ExternalDataTranObject
- ExternalName

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ExtDataTranObjectTemplate

Component Type in 1GP Package Manager UI: External Data Transport Object Template

### Use Case

ExtDataTranObjectTemplate represents the definition of a Data Cloud schema object the user includes in a data kit.

### Considerations When Packaging

A Data Cloud feature is always packaged via a data kit. You add the external data transport object template to a data kit and then add that data kit to a package. You can't directly add this component to a package.

### License Requirements

For more information, see [Data Cloud Standard Permission Sets](#) in Salesforce Help.

### Post Install Steps

After you install a package that contains a data kit, you must manually deploy the features from the installed data kit.

### Documentation

*Data Cloud Developer Guide:* [Packages and Data Kits](#)

*Salesforce Help:* [Packaging in Data Cloud](#)

## External Document Storage Configuration

Represents configuration, which admin makes in setup to specify the drive, path, and named credential to be used for storing documents on external drives.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- Target Object
- Record Type
- External Document Storage Identifier
- Document Path
- Named Credential
- Storage Drive Type

### Neither Package Developer or Subscriber Can Edit

- DeveloperName
- MasterLabel

## More Information

### Feature Name

Metadata Name: ExternalDocStorageConfig

### Use Case

Represents the configuration that the admin makes in Setup to specify the drive, path, and named credential to be used for storing the documents on external drives.

### License Requirements

Microsoft Word 365

### Documentation

*Salesforce Help:* [Configure External Document Storage for Contracts](#)


## External Services

Represents the External Service configuration for an org.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## More Information

### Feature Name

Metadata Name: ExternalServiceRegistration

Component Type in 1GP Package Manager UI: ExternalServiceRegistration

### Considerations When Packaging

Package developers must add named credential components to the External Services registration package. A subscriber can also create a named credential in Salesforce. However, the subscriber must use the same name as the named credential specified in the External Services registration that references it.

Create named credentials manually or with Apex. Be sure to add the named credential to a package so that subscriber orgs can install it. When a subscriber org installs a named credential, it can use the Apex callouts generated by the External Services registration process.

### Documentation

*Salesforce Help:* [External Services](#)

## Feature Parameter Boolean

Represents a boolean feature parameter in the Feature Management App (FMA). Feature parameters let you drive app behavior and track activation metrics in subscriber orgs that install your package.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.



Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No. See note.
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

Note: Feature parameters with a data flow direction set as LMO-to-Subscriber, can be updated in the LMO (License Management Org). Feature parameters with a data flow direction set as Subscriber-to-LMO can be updated using Apex in the subscriber org. Neither of these changes require a package upgrade.

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Master Label
- Value (When Data Flow Direction is set to LMO to Subscriber)

### Both Package Developer and Subscriber Can Edit

- Value (When Data Flow Direction is set to Subscriber to LMO)

### Neither Package Developer or Subscriber Can Edit

- Full Name
- Data Type
- Data Flow Direction

## More Information

### Feature Name

Metadata Name: FeatureParameterBoolean

Component Type in 1GP Package Manager UI: Feature Parameter Boolean

### Use Case

Use LMO-to-Subscriber feature parameters to enable and disable your app's features, or use Subscriber-to-LMO feature parameters to track customer preferences and activation metrics.

### Considerations When Packaging

Feature parameters are an extension of the License Management App (LMA), and because beta package versions can't be registered with the LMA, there are aspects of feature parameters that can't be tested using a beta package version. If you use the default value, you can test LMO-to-Subscriber values in beta package versions. You can't test any Subscriber-to-LMO feature parameter values in a beta managed package version.

### Usage Limits

A package can include up to 200 feature parameters.

### Documentation

Metadata API Developer Guide: [FeatureParameterBoolean](#)

[Create Feature Parameters for Your Second-Generation Managed Package](#)

[Create Feature Parameters in Your First-Generation Packaging Org](#)

*Apex Reference Guide:* [FeatureManagement Class](#)

## Feature Parameter Date

Represents a date feature parameter in the Feature Management App (FMA). Feature parameters let you drive app behavior and track activation metrics in subscriber orgs that install your package.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No. See note.
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

Note: Feature parameters with a data flow direction set as LMO-to-Subscriber, can be updated in the LMO (License Management Org). Feature parameters with a data flow direction set as Subscriber-to-LMO can be updated using Apex in the subscriber org. Neither of these changes require a package upgrade.

## Editable Properties After Package Promotion or Installation

### [Only Package Developer Can Edit](#)

- Master Label
- Value (When Data Flow Direction is set to LMO to Subscriber)

### [Both Package Developer and Subscriber Can Edit](#)

- Value (When Data Flow Direction is set to Subscriber to LMO)

### [Neither Package Developer or Subscriber Can Edit](#)

- Full Name
- Data Type
- Data Flow Direction

## More Information

### Feature Name

Metadata Name: FeatureParameterDate

Component Type in 1GP Package Manager UI: Feature Parameter Date

**Use Case**

Use LMO-to-Subscriber feature parameters to enable and disable your app's features, or use Subscriber-to-LMO feature parameters to track customer preferences and activation metrics.

**Considerations When Packaging**

Feature parameters are an extension of the License Management App (LMA), and because beta package versions can't be registered with the LMA, there are aspects of feature parameters that can't be tested using a beta package version. If you use the default value, you can test LMO-to-Subscriber values in beta package versions. You can't test any Subscriber-to-LMO feature parameter values in a beta managed package version.

**Usage Limits**

A package can include up to 200 feature parameters.

**Documentation**

*Metadata API Developer Guide:* [FeatureParameterDate](#)

[Create Feature Parameters for Your Second-Generation Managed Package](#)

[Create Feature Parameters in Your First-Generation Packaging Org](#)

*Apex Reference Guide:* [FeatureManagement Class](#)

## Feature Parameter Integer

Represents an integer feature parameter in the Feature Management App (FMA). Feature parameters let you drive app behavior and track activation metrics in subscriber orgs that install your package.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No. See note.
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

Note: Feature parameters with a data flow direction set as LMO-to-Subscriber, can be updated in the LMO (License Management Org). Feature parameters with a data flow direction set as Subscriber-to-LMO can be updated using Apex in the subscriber org. Neither of these changes require a package upgrade.

## Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- Master Label
- Value (When Data Flow Direction is set to LMO to Subscriber)

[Both Package Developer and Subscriber Can Edit](#)

- Value (When Data Flow Direction is set to `Subscriber to LMO`)

[Neither Package Developer or Subscriber Can Edit](#)

- Full Name
- Data Type
- Data Flow Direction

## More Information

### Feature Name

Metadata Name: `FeatureParameterInteger`

Component Type in 1GP Package Manager UI: Feature Parameter Integer

### Use Case

Use LMO-to-Subscriber feature parameters to enable and disable your app's features, or use Subscriber-to-LMO feature parameters to track customer preferences and activation metrics.

### Considerations When Packaging

Feature parameters are an extension of the License Management App (LMA), and because beta package versions can't be registered with the LMA, there are aspects of feature parameters that can't be tested using a beta package version. If you use the default value, you can test LMO-to-Subscriber values in beta package versions. You can't test any Subscriber-to-LMO feature parameter values in a beta managed package version.

### Usage Limits

A package can include up to 200 feature parameters.

### Documentation

*Metadata API Developer Guide:* [FeatureParameterInteger](#)

[Create Feature Parameters for Your Second-Generation Managed Package](#)

[Create Feature Parameters in Your First-Generation Packaging Org](#)

*Apex Reference Guide:* [FeatureManagement Class](#)

## Field Set


Represents a field set. A field set is a grouping of fields. For example, you could have a field set that contains fields describing a user's first name, middle name, last name, and business title.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Label
- Available fields

### Both Package Developer and Subscriber Can Edit

- Selected fields (only subscriber editable)

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name


Metadata Name: FieldSet

Component Type in 1GP Package Manager UI: Field Set

### Considerations When Packaging

Field sets in installed packages perform different merge behaviors during a package upgrade:

If a package developer:	Then in the package upgrade:
Changes a field from <b>Unavailable</b> to <b>Available for the Field Set</b> or <b>In the Field Set</b>	The modified field is placed at the end of the upgraded field set in whichever column it was added to.
Adds a field	The new field is placed at the end of the upgraded field set in whichever column it was added to.
Changes a field from <b>Available for the Field Set</b> or <b>In the Field Set</b> to <b>Unavailable</b>	The field is removed from the upgraded field set.
Changes a field from <b>In the Field Set</b> to <b>Available for the Field Set</b> (or vice versa)	The change isn't reflected in the upgraded field set.

 **Note:** Subscribers aren't notified of changes to their installed field sets. The developer must notify users of changes to released field sets through the package release notes or other documentation. Merging has the potential to remove fields in your field set.

When a field set is installed, a subscriber can add or remove any field.

### Documentation

*Metadata API Developer Guide:* [FieldSet](#)


## Field Source Target Relationship

Stores the relationships between a data model object (DMO) and its fields. For example, the Individual.Id field has a one-to-many relationship (1:M) with the ContactPointEmail.PartyId field.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- CreationType
- DeveloperName
- MasterLabel
- RelationshipCardinality
- SourceField
- TargetField

### Both Package Developer and Subscriber Can Edit

- LastDataChangeStatusDateTime
- LastDataChangeStatusErrorCode

- Status

Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: FieldSrcTrgtRelationship

Component Type in 1GP Package Manager UI: Field Source Target Relationship

### License Requirements

Data Cloud must be provisioned.

### Documentation

Metadata API Developer Guide: [FieldSrcTrgtRelationship](#)


## Flow

Represents the metadata associated with a flow. With Flow, you can create an application that navigates users through a series of pages to query and update records in the database. You can also execute logic and provide branching capability based on user input to build dynamic applications.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	Yes, except a flow that is a template or overridable.

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- Entire flow

Both Package Developer and Subscriber Can Edit

- Flow Label

- Description
- Status

[Neither Package Developer or Subscriber Can Edit](#)

- Flow API Name
- URL

## More Information

### Feature Name

Metadata Name: Flow

### Use Case

To repeat a business process automatically such as creating an account when some criteria are met or sending an email every week, build a flow to save time and resources

### Considerations When Packaging

- When you upload a package or package version, the active flow version is included. If the flow has no active version, the latest version is packaged.
- To update a managed package with a different flow version, activate that version and upload the package again. Or deactivate all versions of the flow, make sure the latest flow version is the one to distribute, and then upload the package.
- In a packaging org, you can't delete a flow after you upload it to a released or beta first-generation managed package. You can only delete a flow version from a packaging org after you upload it to a released or beta first-generation managed package, if:
  - Salesforce Customer Support activated the Managed Component Deletion permission.
  - The flow version is not the most recently packaged version of the flow.
  - The flow version is not active.
  - The flow version is not the only version.
- You can't delete a flow from an installed package. To remove a packaged flow from your org, deactivate it and then uninstall the package.
- If you have multiple versions of a flow installed from multiple unmanaged packages, you can't remove only one version by uninstalling its package. Uninstalling a package—managed or unmanaged—that contains a single version of the flow removes the entire flow, including all versions.
- You can't include flows in package patches.
- An active flow in a package is active after it's installed. The previous active version of the flow in the destination org is deactivated in favor of the newly installed version. Any in-progress flows based on the now-deactivated version continue to run without interruption but reflect the previous version of the flow. The same behavior is true even if the destination org deactivated the flow. Future active versions of the flow that are packaged activate the flow during package upgrade.
- Upgrading a managed package in your org installs a new flow version only if there's a newer flow version from the developer. After several upgrades, you can end up with multiple flow versions.
- A package version can contain only one flow version per flow. If you install a managed package version that contains a flow, only the active flow version is deployed. If the flow has no active version, the latest version is deployed.
- If you install a flow from an unmanaged package that has the same name but a different version number as a flow in your org, the newly installed flow becomes the latest version of the existing flow. However, if the packaged flow has the same name and version number as a flow already in your org, the package install fails. You can't overwrite a flow.



- A flow can be modified if it's deployed from an org without a namespace to an org with a namespace. If the flow references data like a field on an object, the reference in the flow is modified to prepend the namespace to the field. Because the flow is modified, a flow version is created in the subscriber org.
- Flow Builder can't open a flow that is installed from a managed package, unless the flow is a template or overridable.
- You can't create a package that contains flows invoked by both managed and unmanaged package pages. As a workaround, create two packages, one for each type of component. For example, suppose that you want to package a customizable flow invoked by a managed package page. Create one unmanaged package with the flow that users can customize. Then create another managed package with the Visualforce page referencing the flow (including namespace) from the first package.
- When you translate a flow from a managed package, the flow's Master Definition Name doesn't appear on the Translate page or the Override page. To update the translation for the Master Definition Name, edit the flow label and then update the translation from the Translate page.
- If any of the following elements are used in a flow, packageable components that they reference aren't included in the package automatically. To deploy the package successfully, manually add those referenced components to the package.
  - Post to Chatter
  - Send Email
  - Submit for Approval
- If a flow references a Lightning component that depends on a CSP Trusted Site, the trusted site isn't included in the package or change set automatically.

### Usage Limits

*Salesforce Help:* [General Flow Limits](#)

### Relationship to Other Components

The associated Flow Definition component is required for managed 1GP packages.

### Documentation

*Metadata API Developer Guide:* [Flow](#)

*Salesforce Help:* [Packaging Considerations for Flows](#)

*Salesforce Help:* [Considerations for Deploying Flows with Packages](#)

## Flow Category

Represents a list of flows that are grouped by category.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- label
- description

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: FlowCategory

### Use Case

To reuse flow-based automated processes, group the flows into a flow category, and then add one or more flow categories to a Lightning Bolt Solution.

### License Requirements

Customize Application user permission

View Setup and Configuration user permission

### Relationship to Other Components

You can use FlowCategory only as part of a Lightning Bolt Solution.

### Documentation

*Salesforce Help:* [Add Flows to a Lightning Bolt Solution](#)

*Salesforce Help:* [Package and Distribute a Lightning Bolt Solution](#)

## Flow Definition

Represents the flow definition's description and active flow version number.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- Active Version Number
- Description
- Master Label

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: Flow Definition

Component Type in 1GP Package Manager UI: Flow Definition

### Use Case

Include this component when you use managed 1GP to package flows.

### Considerations When Packaging

[Considerations for Deploying Flows with Packages](#)

### Relationship to Other Components

The associated Flow component is required for managed 1GP packages.

### Documentation

*Metadata API Developer Guide:* [Flow Definition](#)

*Salesforce Help:* [Flow Builder](#)

## Flow Test

Represents the metadata associated with a flow test. Before you activate a record-triggered flow, you can test it to verify its expected results and identify flow run-time failures.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation\

### Only Package Developer Can Edit

- All properties

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- API Name

## More Information

### Feature Name

Metadata Name: FlowTest

Component Type in 1GP Package Manager UI: FlowTest

### Use Case

Include this component when you use managed 1GP to package flow tests.

### Usage Limits

*Salesforce Help:* [Considerations for Testing Flows](#)

### Relationship to Other Components

The associated Flow component is required for managed 1GP packages.

### Documentation

*Metadata API Developer Guide:* [Flow Test](#)

*Salesforce Help:* [Testing Your Flow](#)

## Folder

Represents a folder.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- All attributes except Folder Unique Name

### Neither Package Developer or Subscriber Can Edit

- Folder Unique Name

## More Information

- Five different folder metadata types can be packaged:
  - DashboardFolder
  - DocumentFolder
  - EmailFolder (available for Salesforce Classic email templates only)
  - EmailTemplateFolder
  - ReportFolder
- Components that Salesforce stores in folders, such as documents, can't be added to packages when stored in personal and unfiled folders. Put documents, reports, and other components that Salesforce stores in folders in one of your publicly accessible folders.
- Components such as documents, email templates, reports, or dashboards are stored in new folders in the installer's org using the publisher's folder names. Give these folders names that indicate they're part of the package.
- If a new report, dashboard, document, or email template is installed during an upgrade, and the folder containing the component was deleted by the subscriber, the folder is re-created. Any components in the folder that were previously deleted aren't restored.
- The name of a component contained in a folder must be unique across all folders of the same component type, excluding personal folders. Components contained in a personal folder must be unique within the personal folder only.

### Documentation

*Metadata API Developer Guide:* [Folder](#)

## Fuel Type

Represents a custom fuel type in an org.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No

Component Has IP Protection	No
-----------------------------	----

---

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All attributes

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: FuelType

Component Type in 1GP Package Manager UI: Fuel Type

### License Requirements

- Net Zero Cloud Growth license or Net Zero Cloud Starter license
- Net Zero Cloud Manager permissions set

### Post Install Steps

Enable these org settings:

- Net Zero Cloud
- Manage Carbon Accounting

### Documentation

- [Salesforce Help: Create a Custom Fuel Type](#)

## Fuel Type Sustainability Unit of Measure

Represents a mapping between the custom fuel types and their corresponding unit of measure (UOM) values defined by a customer in an org.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No

Component Has IP Protection	No
-----------------------------	----

---

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All attributes

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: FuelTypeSustnUom

Component Type in 1GP Package Manager UI: Fuel Type Sustainability Unit of Measure

### License Requirements

- Net Zero Cloud Growth license or Net Zero Cloud Starter license
- Net Zero Cloud Manager permissions set

### Post Install Steps

Enable these org settings:

- Net Zero Cloud
- Manage Carbon Accounting

### Documentation

- [Salesforce Help: Associate a Custom Fuel Type with a Unit of Measure](#)

## Fundraising Config

Represents a collection of settings to configure the fundraising product.


## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.

Component Has IP Protection

No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- LapsedUnpaidTrxnCount
- HouseholdSoftCreditRole
- IsHshldSoftCrAutoCrea
- InstallmentExtDayCount
- DonorMatchingMethod
- FailedTransactionCount
- ShouldCreateRcrSchdTrxn
- ShouldClosePaidRcrCmt

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: FundraisingConfig

### License Requirements

Fundraising Access (Permission Set License)

### Documentation

Metadata API Developer Guide: [FundraisingConfig](#)

## Gateway Provider Payment Method Type

Represents an entity that allows integrators and payment providers to choose an active payment to receive an order's payment data rather than allowing the Salesforce Order Management platform to select a default payment method.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.



Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- All fields

## More Information

### Feature Name

Metadata Name: GatewayProviderPaymentMethodType

### License Requirements

Salesforce Order Management, B2B Commerce, or B2C Commerce (for B2B2C Commerce) licenses are required. These licenses enable the Payment Platform org permission required to use payments objects.

### Documentation

*Salesforce Help:* [Processing Payments with Payment Gateways](#)

## Gen AI Function

Represents a copilot action that can be added to Einstein Agents and Copilot.


## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes

Component Has IP Protection

No (However, actions can incorporate flows or Apex code that do have IP protection.)

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- MasterLabel
- IsConfirmationRequired
- Description

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: [GenAiFunction](#)

Component Type in 1GP Package Manager UI: GenAiFunction

### Use Case

Provide actions that customers can add to their own agents and copilots.

### Considerations When Packaging

When creating an Agent Action of type Apex, the Apex class, invocable Apex method, and any invocable Apex variables must all be marked as `global`. If any of these are public or private, the Apex method won't appear in the list of options to add to the Agent Action, and won't be invoked by an Agent at runtime.

### Documentation

*Salesforce Help:* [Einstein Copilot](#)

## Global Picklist

Represents the metadata for a global picklist value set, which is the set of shared values that custom picklist fields can use. A global value set isn't a field itself. In contrast, the custom picklist fields that are based on a global picklist are of type ValueSet.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## More Information

### Feature Name

Metadata Name: Global Value Set

Component Type in 1GP Package Manager UI: Global Value Set

### Considerations When Packaging

When explicitly referencing a picklist value in code, keep in mind that picklist values for a custom field can be renamed, added, edited, or deleted by subscribers.

Picklist field values can be added or deleted in the developer's org. Changes to standard picklists can't be packaged and deployed to subscriber orgs, and picklist values deleted by the developer are still available in the subscriber's org. If there are differences between the package and the target org, or if there are dependencies on new values from features such as PathAssistant, the deploy fails. To change values in subscriber orgs, you must manually add or modify the values in the target subscriber org.

Updating picklist values in unlocked packages isn't supported. Manually add or modify the values in the target subscriber org.

Package upgrades retain dependent picklist values that are saved in a managed custom field.

Global value sets can be added to developer and subscriber orgs. Global value sets have these behaviors during a package upgrade.

- Label and API names for field values don't change in subscriber orgs.
- New field values aren't added to the subscriber orgs.
- Active and inactive value settings in subscriber orgs don't change.
- Default values in subscriber orgs don't change.
- Global value set label names change if the package upgrade includes a global value set label change.

### Documentation

*Salesforce Help:* [Create a Global Picklist Value Set](#)

*Salesforce Help:* [Make Your Custom Picklist Field Values Global](#)

## Home Page Component

Represents the metadata associated with a home page component. You can customize the Home tab in Salesforce Classic to include components such as sidebar links, a company logo, a dashboard snapshot, or custom components that you create. Use to create, update, or delete home page component definitions.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Body
- Component Position

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name
- Type

## More Information

### Feature Name

Metadata Name: HomePageComponent

Component Type in 1GP Package Manager UI: Home Page Component

### Relationship to Other Components

When you package a custom home page layout, all the custom home page components included on the page layout are automatically added. Standard components such as Messages & Alerts aren't included in the package and don't overwrite the installer's Messages & Alerts. To include a message in your custom home page layout, create an HTML Area type custom Home tab component containing your message. From Setup, in the Quick Find box, enter *Home Page Components*, then select **Home Page Components**. Then add the message to your custom home page layout.

### Documentation

Metadata API Developer Guide: [HomePageComponent](#)

## Home Page Layout

Represents the metadata associated with a home page layout. You can customize home page layouts and assign the layouts to users based on their user profile.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- None

#### Both Package Developer and Subscriber Can Edit

- All attributes except Layout Name

#### Neither Package Developer or Subscriber Can Edit

- Layout Name

### More Information

#### Feature Name

Metadata Name: HomePageLayout

Component Type in 1GP Package Manager UI: Home Page Layout

#### Considerations When Packaging

After they're installed, your custom home page layouts are listed with all the subscriber's home page layouts. Distinguish them by including the name of your app in the page layout name.

#### Documentation

Metadata API Developer Guide: [HomePageLayout](#)

## Identity Verification Proc Def

Represents the definition of the identity verification process.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- MasterLabel
- SearchLayoutType

#### Both Package Developer and Subscriber Can Edit

- None

#### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: IdentityVerificationProcDef

Component Type in 1GP Package Manager UI: Identity Verification Process Definition

### Use Case

Links the configuration for Identity Verification to a flow.

### License Requirements

Industries Health Cloud, Industries Sales Excellence, and Industries Service Excellence licenses.

Actionable Segmentation Engagement, Industries Sales Excellence, Industry Service Excellence or Health Cloud Platform Permission set license is required to use this metadata type.

### Relationship to Other Components

An Identity Verification Process Field record looks up to an Identity Verification Process Details record, which in turn looks up to an Identity Verification Process Definition record.

### Documentation

*Health Cloud Developer Guide:* [IdentityVerificationProcDef](#)

## Inbound Network Connection

Represents a private connection between a third-party data service and a Salesforce org. The connection is inbound because the callouts are coming into Salesforce.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

You can only delete connections that are in an unprovisioned state.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

 **Note:** 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.

#### Only Package Developer Can Edit

- AWS VPC Endpoint ID
- Connection Type
- Developer Name
- Description
- Link ID
- Master Label
- Region
- Source IP Ranges

#### Both Package Developer and Subscriber Can Edit

- Status

#### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: InboundNetworkConnection

Component Type in 1GP Package Manager UI: Inbound Network Connection

### Considerations When Packaging

- Packaged connections are installed as unprovisioned. Alert subscribers about how to provision connections after package installation.
- If a developer changes the Region of a packaged connection that is subscriber-provisioned, the upgrade fails for the subscriber. Alert subscribers about tearing down the connection before updating the Region field. As a best practice, avoid changing the Region of a packaged connection unless necessary.

### License Requirements

This feature is available with the Private Connect license.

### Documentation

*Salesforce Help:* [Secure Cross-Cloud Integrations with Private Connect](#)

*Salesforce Help:* [Establish an Inbound Connection with AWS](#)

## IndustriesEinsteinFeatureSettings

Represents the settings for enabling the Industries Einstein feature.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None



## More Information

### Feature Name

Metadata Name: IndustriesEinsteinFeatureSettings

### Documentation

*Salesforce Help:* [Intelligent Document Reader](#)

*Salesforce Help:* [Intelligent Form Reader](#)

## IntegrationProviderDef

Represents an integration definition associated with a service process. Stores data for the Industries: Send Apex Async Request and Industries: Send External Async Request invocable actions.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All other fields

### Both Package Developer and Subscriber Can Edit

- StringValue
- IntegerValue
- DateTimeValue
- DateValue
- PercentageValue
- DoubleValue
- IsTrueOrFalseValue

### Neither Package Developer or Subscriber Can Edit

- FullName

## More Information

### IntegrationProviderDef

Metadata Name: IntegrationProviderDef

Component Type in 1GP Package Manager UI: IntegrationProviderDef

### Documentation

IntegrationProviderDef in *Metadata API Developer Guide*.

## LearningAchievementConfig

Represents the mapping details between a Learning Achievement type and a Learning Achievement record type.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### [Only Package Developer Can Edit](#)

- All but DeveloperName

### [Both Package Developer and Subscriber Can Edit](#)

- None

### [Neither Package Developer or Subscriber Can Edit](#)

- DeveloperName

## More Information

### Feature Name

Metadata Name: LearningAchievementConfig

### Documentation

Education Cloud Developer Guide


## Learning Item Type

Represents a custom exercise type that an Enablement user takes in an Enablement program in the Guidance Center. A custom exercise type also requires a corresponding LearningItem record for the Guidance Center and corresponding EnblProgramTaskDefinition and EnblProgramTaskSubCategory records for when admins create a program in Program Builder.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All but DeveloperName

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- DeveloperName

## More Information

### Feature Name

Metadata Name: LearningItemType

### Use Case

Include this component in a package with a program if the program has a custom exercise type.

### Considerations When Packaging

See [Considerations for Packaging Enablement Programs and Dependencies](#).

### License Requirements

Enablement add-on license and the Enablement permission set license are required.

 **Important:** Custom exercises aren't compatible with Partner Enablement programs.

### Usage Limits

See [Enablement Limits](#).

### Relationship to Other Components

The Learning Item Type component requires a corresponding Enablement Program Task Subcategory component. Both components are used with custom exercise types in Enablement programs. Package both of these components with an Enablement Program Definition component.

### Documentation

- *Salesforce Help: Sales Programs and Partner Tracks with Enablement*
- *Metadata API Developer Guide: [EnblProgramTaskSubCategory](#)*
- *Metadata API Developer Guide: [LearningItemType](#)*
- *Object Reference for the Salesforce Platform: [EnblProgramTaskDefinition](#)*
- *Object Reference for the Salesforce Platform: [LearningItem](#)*
- *Sales Programs and Partner Tracks with Enablement Developer Guide: [Create a Managed Package for Enablement Programs, Measures, and Content](#)*
- *Sales Programs and Partner Tracks with Enablement Developer Guide: [Implement Custom Exercise Types for Enablement Programs](#)*

## Letterhead

Represents formatting options for the letterhead in an email template. A letterhead defines the logo, page color, and text settings for your HTML email templates. Use letterheads to ensure a consistent look and feel in your company's emails.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- All attributes except Letterhead Name

[Neither Package Developer or Subscriber Can Edit](#)

- Letterhead Name

## More Information

**Feature Name**

Metadata Name: Letterhead

**Documentation***Metadata API Developer Guide:* [Letterhead](#)


## Lightning Bolt

Represents the definition of a Lightning Bolt Solution, which can include custom apps, flow categories, and Experience Builder templates.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## More Information

**Feature Name**

Metadata Name: LightningBolt

Component Type in 1GP Package Manager UI: Lightning Bolt

**Documentation***Metadata API Developer Guide:* [LightningBolt](#)

## Lightning Message Channel

Represents the metadata associated with a Lightning Message Channel. A Lightning Message Channel represents a secure channel to communicate across UI technologies, such as Lightning Web Components, Aura Components, and Visualforce.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### More Information

#### Feature Name

Metadata Name: LightningMessageChannel

Component Type in 1GP Package Manager UI: Lightning Message Channel

#### Considerations When Packaging

To pass the [AppExchange Security Review](#), the `isExposed` attribute must be set to `false`.

#### Documentation

Metadata API Developer Guide: [Lightning Message Channel](#)

Lightning Web Components Developer Guide: [Create a Message Channel](#)

## Lightning Page


Represents the metadata associated with a Lightning page. A Lightning page represents a customizable screen made up of regions containing Lightning components.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
-----------------	---

Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Lightning page

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: FlexiPage

### Documentation

*Metadata API Developer Guide:* [Flexipage](#)

## Lightning Web Component

Represents a Lightning web component bundle. A bundle contains Lightning web component resources.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

You can build Lightning components using two programming models: the Lightning Web Components model, and the original Aura Components model.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

When a package developer removes an Aura or Lightning web component from a package, the component remains in a subscriber's org after they install the upgraded package. The administrator of the subscriber's org can delete the component, if desired. This behavior is the same for a Lightning web component or an Aura component with a `public` or `global` access value.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- API Version
- Description
- `isExposed` (can only change from `false` to `true`)
- Label
- Markup
- Targets
- `targetConfigs`

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Lightning Web Component

Metadata Name: `LightningComponentBundle`

Component Type in 1GP Package Manager UI: Lightning Web Component Bundle

### Considerations When Packaging

Licensing Considerations:

Lightning Web Components don't automatically enforce managed package licensing. Lightning Web Components in a managed package can be seen and used by users who don't have active licenses for that managed package. These Lightning Web Components can also be seen and used after a trial of that managed package expires.

AppExchange partners are responsible for enforcing package licensing in their Lightning Web Components. We recommend using an Apex controller that calls either the `UserInfo.isCurrentUserLicensed(namespace)` or `UserInfo.isCurrentUserLicensedForPackage(packageID)` methods, and only rendering the component if `true` is returned.

Considerations When Using `isExposed`:

If `isExposed` is `false`, the package developer can remove configuration targets and a public (`@api`) property from a component. The component isn't available to other namespaces or to Salesforce builders like Lightning App Builder and Experience Builder.



If `isExposed` is true and the component is in a published managed package, the package developer can't remove configuration targets or a public (`@api`) property from a component. This restriction is enforced even if the target or public property was added after the most recent publication of the package.

If `isExposed` is true, the component is available to other namespaces, including namespaces outside of a published managed package.

If `isExposed` is true and a `Targets` value is also provided, the component is available to Salesforce builders such as Lightning App Builder and Experience Builder.

When you delete a Lightning Web Component with an `isExposed` value of true, we recommend a two-stage process to ensure that the deleted component has no dependencies on the other items in the package. See [Remove Components from Second-Generation Managed Packages](#) for details.

### Documentation

[Lightning Web Components Developer Guide](#)

*Lightning Web Components Developer Guide: [Add Components to Managed Packages](#)*

*Lightning Web Components Developer Guide: [Delete Components from Managed Packages](#)*

## List View

ListView allows you to see a filtered list of records, such as contacts, accounts, or custom objects.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- All attributes except View Unique Name

### Neither Package Developer or Subscriber Can Edit

- View Unique Name


## More Information

### Feature Name

Metadata Name: ListView

Component Type in 1GP Package Manager UI: List View

### Considerations When Packaging

If a subscriber removes a packaged listview from their production org, that listview is deprecated, but not deleted. If that subscriber org later creates a sandbox org, and upgrades the package in the sandbox org, the removed listview persists in the sandbox org. To remove the listview from the sandbox, package subscribers can click  and select **Delete**.

### Relationship to Other Components

List views associated with queues can't be included in a managed package or an unlocked package.

### Documentation

Metadata API Developer Guide: [ListView](#)

## Live Chat Sensitive Data Rule

Represents a rule for masking or deleting data of a specified pattern. Written as a regular expression (regex). Use this object to mask or delete data of specified patterns, such as credit card, social security, or phone and account numbers.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes, Supported in 1GP Packages only
Component Has IP Protection	No



**Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: LiveChatSensitiveDataRule

Component Type in 1GP Package Manager UI: Sensitive Data Rules

### Documentation

Metadata API Developer Guide: [LiveChatSensitiveDataRule](#)

## Loyalty Program Setup

Represents the configuration of a loyalty program process including its parameters and rules. Program processes determine how new transaction journals are processed. When new transaction journals meet the criteria and conditions for a program process, actions that are set up in the process are triggered for the transaction journals.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes, except templates

## Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- Loyalty Program Process records

Both Package Developer and Subscriber Can Edit

- Label
- Description
- Status

Neither Package Developer or Subscriber Can Edit

- API Name
- URL

## More Information

### Feature Name

Metadata Name: LoyaltyProgramSetup

Component Type in 1GP Package Manager UI: Loyalty Program Setup

### Use Case

Promotion setup allows loyalty program managers to create loyalty program processes.

### License Requirements

Loyalty Management permission set license

### Documentation

*Salesforce Help:* [Create Processes with Promotion Setup](#)

## Marketing App Extension

Represents an integration with a third-party app or service that generates prospect external activity.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- DeveloperName
- MasterLabel
- Description

### Both Package Developer and Subscriber Can Edit

- IsActive

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: MarketingAppExtension

Component Type in 1GP Package Manager UI: Marketing App Extension

### Use Case

Partners and ISVs can provide integrations with third-parties so Account Engagement customers can enhance their automations.

### Considerations When Packaging

Marketing app extensions require an associated action type component to function. The related component activity type isn't supported for packaging.

### License Requirements

This feature is available in Plus, Advanced, or Premium editions of Account Engagement. To work with marketing app extensions, users must be a Salesforce admin or have the [required permissions to access Marketing Setup](#).

### Usage Limits

The number of active extensions, activities, and actions the end user can have at one time depends on their edition of Account Engagement.

- Plus—10 active extensions, with 10 active activities and 10 active actions per active extension
- Advanced—20 active extensions, with 20 active activities and 20 active actions per active extension
- Premium—30 active extensions, with 30 active activities and 30 active actions per active extension

For more on limits, see [Considerations for Working with Marketing App Extensions](#).

### Post Install Steps

To receive data, the extension must be activated for automations and have a business unit assignment.

### Relationship to Other Components

The extension requires an associated action type component to function.

### Documentation

This component is part of Account Engagement's extensibility feature set.

- *Salesforce Help:* [Automate Data Sharing with Third-Party Apps](#)
- *Developer Guide:* [Work with Extensibility Features](#)

## MarketingAppExtAction

Represents an Action Type, which is an action that you can add to Engagement Studio programs in Account Engagement and execute in a third-party app.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No

Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- ActionName
- ApiName
- Description
- ActionSchema
- ActionSelector
- ActionParams
- Version

### Both Package Developer and Subscriber Can Edit

- IsActive

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: MarketingAppExtAction

Component Type in 1GP Package Manager UI: Marketing App Extension

### Use Case

Partners and ISVs can provide integrations with third-parties so Account Engagement customers can automate actions or tasks in external applications.

### Considerations When Packaging

This component is included when the parent component MarketingAppExtension is added to a package. The related component marketingAppExtActivity isn't supported for packaging.

### License Requirements

This feature is available in Plus, Advanced, or Premium editions of Account Engagement. To work with marketing app extensions and related components, users must be a Salesforce admin or have the [required permissions to access Marketing Setup](#).

### Usage Limits

The number of active extensions, activities, and actions the end user can have at one time depends on their edition of Account Engagement.

- Plus—10 active extensions, with 10 active activities and 10 active actions per active extension
- Advanced—20 active extensions, with 20 active activities and 20 active actions per active extension
- Premium—30 active extensions, with 30 active activities and 30 active actions per active extension

For more on limits, see [Considerations for Working with Marketing App Extensions](#).

**Post Install Steps**

To receive data, the action and its related extension must be activated for automations and the extension must have a business unit assignment.

**Relationship to Other Components**

This component is a child of the MarketingAppExtension component.

**Documentation**

This component is part of Account Engagement's extensibility feature set.

- *Salesforce Help:* [Automate Data Sharing with Third-Party Apps](#)
- *Developer Guide:* [Work with Extensibility Features](#)

## Marketing App Extension Activity

Represents an Activity Type, which is a prospect activity that occurs in a third-party app and can be used in Account Engagement automations.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- MasterLabel
- Description

**Both Package Developer and Subscriber Can Edit**

- IsActive

**Neither Package Developer or Subscriber Can Edit**

- DeveloperName
- EndpointUrl
- MarketingAppExtension

### More Information

**Feature Name**

Metadata Name: MarketingAppExtActivity

Component Type in 1GP Package Manager UI: Marketing App Extension

### Use Case

Partners and ISVs can use Activities to submit external prospect engagement data to Marketing Cloud Account Engagement.

### Considerations When Packaging

This component is included when the parent component [MarketingAppExtension](#) on page 200 is added to a package. The related component `MarketingAppExtActivity` isn't supported for packaging.

### License Requirements

This feature is available in Plus, Advanced, or Premium editions of Account Engagement. To work with marketing app extensions and related components, users must be a Salesforce admin or have the [required permissions to access Marketing Setup](#).

### Usage Limits

The number of active extensions, activities, and actions the end user can have at one time depends on their edition of Account Engagement.

- Plus—10 active activities per active extension
- Advanced—20 active activities per active extension
- Premium—30 active activities per active extension

For more information, see [Considerations for Working with Marketing App Extensions](#).

### Post Install Steps

To receive data, the activity and its related extension must be activated for automations.

### Relationship to Other Components

This component is a child of the [MarketingAppExtension](#) on page 200 component. Activities interact with Marketing Cloud Account Engagement features that support external activities. For more information, see [Capture External Prospect Activity](#).

### Documentation

This component is part of Account Engagement's extensibility feature set.

- *Salesforce Help:* [Automate Data Sharing with Third-Party Apps](#)
- *Developer Guide:* [Work with Extensibility Features](#)

## Market Segment Definition


Represents the field values for `MarketSegmentDefinition`. `MarketSegmentDefinition` is used to store the exportable metadata of a segment, such as segment criteria and other attributes. Developers can create segment definition packages, pass segment definition in the form of data build tool (DBT), and publish it on AppExchange for subscriber organizations to install and instantiate these segments.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No



 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Yes, applicable for all properties.

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: MarketSegmentDefinition

Component Type in 1GP Package Manager UI: Market Segment Definition

## MktCalculatedInsightsObjectDef

Represents Calculated Insight definition such as expression.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- BuilderExpression
- CalculatedInsightCreationType
- Description
- Expression

- Label

Both Package Developer and Subscriber Can Edit

- CalculatedInsightObjectDefinitionStatus
- Description

Neither Package Developer or Subscriber Can Edit

- DeveloperName

## More Information

### Feature Name

Metadata Name: MktCalcInsightObjectDef

Component Type in 1GP Package Manager UI: MktCalcInsightObjectDef.

### Use Case

Defines CDP calculated insight for easy creation on subscriber organizations.

### Considerations When Packaging

To package this component, first add it to a data kit. For more information about data kits, see [Data Kits](#) in *Salesforce Help*.

### License Requirements

You need Customer 360 Audiences Corporate (cdpPsl) licenses on both package developer org and subscriber org.

### Post Install Steps

User has to go to the **Calculated Insights** object home in Customer Data Platform, click **New action** and select **Create from a Package**.

### Relationship to Other Components

Calculated Insight Component is tied to the Data Model Object component. The Calculated Insight component must have Data Model Object dependencies available on the subscriber organization that are used in the Calculated Insight.

### Documentation

*Metadata API Developer Guide:* [MktCalcInsightObjectDef](#)

## MktDataConnection

Represents the connection information of an external connector that can ingest data to Data Cloud, read data from the source, or write data to the source in Data Cloud.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No

Component Has IP Protection

No

---

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- masterLabel
- Parameters
  - paramName
  - value
- Credentials
  - credentialName
  - value

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: MktDataConnection

Component Type in 1GP Package Manager UI: Data Connection

### Use Case

To reuse connection parameters.

### Considerations When Packaging

Connection credentials are excluded from the package. Available parameters are defined in Connector Metadata which is exposed from Connect API.

### License Requirements

Data Cloud must be provisioned. For more information, see [Data Cloud: Access and Provisioning](#).

### Usage Limits

The number of connections per connector type can be up to 200.

### Post Install Steps

After you create the connection, it will be in INACTIVE state, you must manually activate the connection.

### Relationship to Other Components

Must be used with Data Stream and Activation.

### Documentation

Salesforce Help: [Third-Party Data Cloud Connectors](#)

## MktDataTranObject

An entity that is used to deliver (aka transport) information from the source to a target (target will be called a landing entity). This can be the schema of a file, API, Event, or other means of transporting data, such as SubscriberFile1.csv, or SubscriberCDCEvent.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No



**Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- CreationType
- DataSource
- DataSourceObject
- DeveloperName
- ObjectCategory
- Status

#### Both Package Developer and Subscriber Can Edit

- DataConnector

#### Neither Package Developer or Subscriber Can Edit

- None

### More Information

#### Feature Name

Metadata Name: MktDataTranObject

Component Type in 1GP Package Manager UI: It's not a top-level component, it can only be spidered in when customer selects some other component. You won't be able to add this component directly to the package.

**License Requirements**

Data Cloud must be provisioned.

**Documentation**

*Metadata API Developer Guide:* [MktDataTranObject](#)

## Named Credential


Represents a named credential, which specifies the URL of a callout endpoint and its required authentication parameters in one definition. A named credential can be specified as an endpoint to simplify the setup of authenticated callouts.

## Component Manageability Rules


Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

## Editable Properties After Package Promotion or Installation

 **Note:** In addition to these properties, the Description, ParameterName, ParameterValue, and SequenceNumber properties have the same editability as the NamedCredentialParameters they're included in.

### Only Package Developer Can Edit

- Label
- NamedCredentialType
- Legacy Named Credentials only (deprecated and unsupported in future releases)
  - Endpoint (deprecated)

### Both Package Developer and Subscriber Can Edit

- CalloutOptions
  - AllowMergeFieldsInBody
  - AllowMergeFieldsInHeader

- GenerateAuthorizationHeader
- NamedCredentialParameters
  - AllowedManagedPackageNamespaces (only subscriber editable)
  - Authentication
  - ClientCertificate (only subscriber editable in 2GP)
  - HttpHeaders
  - OutboundNetworkConnection
  - Url
- Legacy Named Credentials only (deprecated and unsupported in future releases)
  - AuthProvider (deprecated)
  - AuthTokenEndpointUrl (deprecated)
  - AwsAccessKey, AwsAccessSecret, AwsRegion, and AwsService (all deprecated)
  - Certificate (deprecated)
  - JwtAudience, JwtFormulaSubject, JwtIssuer, JwtSigningCertificateId, JwtTextSubject, and JwtValidityPeriodSeconds (all deprecated)
  - OAuthRefreshToken, OAuthScope, and OAuthToken (all deprecated)
  - OutboundNetworkConnectionId (deprecated)
  - Password (deprecated)
  - PrincipalType (deprecated)
  - Protocol (deprecated)
  - Username (deprecated)

Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: `NamedCredential`

### Considerations When Packaging


Certificates aren't packageable. If a certificate needs access to an external system, an administrator must upload one to the subscriber org and reference it in the named credential.

### Relationship to Other Components

You must package `NamedCredential` with the associated `ExternalCredential` component.

The named credential defines a callout endpoint and an HTTP transport protocol, while the external credential represents the details of how Salesforce authenticates to an external system via an authentication protocol. Each named credential must be mapped to at least one external credential.

### Legacy Named Credentials

 **Important:** In Winter '23, Salesforce introduced an improved named credential that is extensible and customizable. We strongly recommend that you use this preferred credential instead of legacy named credentials. For information on extensible, customizable named credentials, see [Named Credentials and External Credentials](#). Legacy named credentials are deprecated and will be discontinued in a future release.

After installing a named credential from a managed or unmanaged package, the subscriber must reauthenticate to the external system.

- For password authentication, the subscriber reenters the password in the named credential definition.
- For OAuth, the subscriber updates the callback URL in the client configuration for the authentication provider and then reauthenticates by selecting **Start Authentication Flow on Save** on the named credential.

### Documentation

*Salesforce Help:* [Named Credentials](#)

*Named Credentials Developer Guide:* [Named Credentials Packaging Guide](#)

*Metadata API Developer Guide:* [NamedCredential](#)


## Object Source Target Map

Contains the object-level mappings between the source and the target objects. The source and target objects can be an MktDataLakeObject or an MktDataModelObject. For example, an Email source object can be mapped to the ContactPointEmail object.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- CreationType
- DeveloperName
- MasterLabel
- ParentObject
- SequenceNbr
- SourceObject
- TargetObject

### Both Package Developer and Subscriber Can Edit

- LastDataChangeStatusDateTime
- LastDataChangeStatusErrorCode
- Status

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ObjectSourceTargetMap

Component Type in 1GP Package Manager UI: It's not a top-level component, it can only be spidered in when customer selects some other component. You won't be able to add this component directly to the package.

### License Requirements

Data Cloud must be provisioned.

### Documentation

Metadata API Developer Guide: [ObjectSourceTargetMap](#)


## OcrSampleDocument

Represents the details of a sample document or a document type that's used as a reference while extracting and mapping information from a customer form.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit



- All

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

OcrSampleDocument

Component Type in 1GP Package Manager UI: OcrSampleDocument

### Use Case

Migrate sample documents created with the Intelligent Form Reader or Intelligent Document Reader feature.

### Considerations When Packaging

If you update the package by deleting OcrSampleDocumentFields associated with the OcrTemplate, the OcrSampleDocumentFields are not deleted.

### License Requirements

AWSTextract1000LimitAddOn-1 for the Intelligent Form Reader feature or IntelligentDocumentReaderAddOn-1 for the Intelligent Document Reader feature.

### Relationship to Other Components

DocumentType, ContentAsset, and OcrTemplate (Optional)

### Documentation

*Metadata API Developer Guide:* [OcrSampleDocument](#)

## OcrTemplate

Represents the details of the mapping between a form and a Salesforce object using Intelligent Form Reader.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No



**Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- All

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

OcrTemplate

Component Type in 1GP Package Manager UI: OcrTemplate

### Use Case

Migrate Mappings created with the Intelligent Form Reader or Intelligent Document Reader feature.

### Considerations When Packaging

OcrTemplate has a dependency on OcrSampleDocument. Before deploying the package, make sure to either include OcrSampleDocument in the package or deploy a package that contains OcrSampleDocument.

### License Requirements

AWSTextract1000LimitAddOn-1 for the Intelligent Form Reader feature or IntelligentDocumentReaderAddOn-1 for the Intelligent Document Reader feature.

### Relationship to Other Components

DocumentType and OcrSampleDocument

### Documentation

*Metadata API Developer Guide:* [OcrTemplate](#)

## Outbound Network Connection

Represents a private connection between a Salesforce org and a third-party data service. The connection is outbound because the callouts are going out of Salesforce.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes

Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** You can only delete connections that are in an unprovisioned state.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

 **Note:** 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.

### Only Package Developer Can Edit

- Connection Type
- Developer Name
- Description
- Master Label
- Region
- Service Name

### Both Package Developer and Subscriber Can Edit

- Status

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: OutboundNetworkConnection

Component Type in 1GP Package Manager UI: Outbound Network Connection

### Considerations When Packaging

- Packaged connections are installed as unprovisioned. Alert subscribers about how to provision connections after package installation.
- If a developer changes the Region or Service Name of a packaged connection that is subscriber-provisioned, the upgrade fails for the subscriber. Alert subscribers about tearing down the connection before you update the Region or Service Name fields. As a best practice, avoid changing the Region or Service Name of a packaged connection unless necessary.
- If you package a Named Credential that references an Outbound Network Connection, the referenced Outbound Network Connection component is automatically added to the package.

### License Requirements

This feature is available with the Private Connect license.

### Documentation

*Salesforce Help:* [Secure Cross-Cloud Integrations with Private Connect](#)

*Salesforce Help:* [Establish an Outbound Connection with AWS](#)


## Page Layout

Represents the metadata associated with a page layout.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- None

#### Both Package Developer and Subscriber Can Edit

- All attributes except Page Layout Name

#### Neither Package Developer or Subscriber Can Edit

- Page Layout Name

### More Information

#### Feature Name

Metadata Name: Layout

#### Considerations

The page layout of the person uploading a package is the layout used for Group and Professional Edition orgs and becomes the default page layout for Enterprise, Unlimited, Performance, and Developer Edition orgs.

Package page layouts alongside complimentary record types if the layout is being installed on an existing object. Otherwise, manually apply the installed page layouts to profiles.

If a page layout and a record type are created as a result of installing a package, the uploading user's page layout assignment for that record type is assigned to that record type for all profiles in the subscriber org, unless a profile is mapped during an install or upgrade.

### Documentation

*Metadata API Developer Guide:* [Layout](#)

## Path Assistant

Represents Path records.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- IsActive field

### Neither Package Developer or Subscriber Can Edit

- SubjectType, SubjectProcessField, and RecordType

## More Information

### Feature Name

Metadata Name: PathAssistant

Component Type in 1GP Package Manager UI: Path Assistant

### Documentation

*Metadata API Developer Guide:* [PathAssistant](#)

## Payment Gateway Provider

Represents the metadata associated with a payment gateway provider.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- All fields

### More Information

#### Feature Name

Metadata Name: PaymentGatewayProvider

#### License Requirements

Salesforce Order Management, B2B Commerce, or B2C Commerce (for B2B2C Commerce) licenses are required. These licenses enable the Payment Platform org permission required to use payments objects.

#### Documentation

*Salesforce Help:* [Processing Payments with Payment Gateways](#)


### Permission Set

Represents a set of permissions that's used to grant more access to one or more users without changing their profile or reassigning profiles. You can use permission sets to grant access but not to deny access.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Label
- Custom object permissions
- Custom field permissions
- Apex class access settings
- Visualforce page access settings

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: PermissionSet

Component Type in 1GP Package Manager UI: Permission Set

### Documentation

Metadata API Developer Guide: [PermissionSet](#)

## Permission Set Groups

Represents a group of permission sets and the permissions within them. Use permission set groups to organize permissions based on job functions or tasks. Then, you can package the groups as needed.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- Permission Set Group Components (Developer can add and remove while Subscriber can add)

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: PermissionSetGroup

Component Type in 1GP Package Manager UI: Permission Set Group

### Considerations When Packaging

Don't assume that a subscriber's permission set group is the same as what the developer has specified. Although developers can define the permission set group and what permission sets can go into it, subscribers can add additional permission sets or mute permissions.

### Relationship to Other Components

This feature can only be used in conjunction with Permission Sets.

### Documentation

*Salesforce Help:* [Permission Set Groups](#)

## Platform Cache

Represents a partition in the Platform Cache.



## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Master Label
- Description
- Default Partition

### Both Package Developer and Subscriber Can Edit

- Organization Capacity
- Trial Capacity

### Neither Package Developer or Subscriber Can Edit

- Developer Name

## More Information

### Feature Name

Metadata Name: PlatformCachePartition

Component Type in 1GP Package Manager UI: Platform Cache Partition

### Documentation

[Set Up a Platform Cache Partition with Provider Free Capacity](#)

*Metadata API Developer Guide:* [PlatformCachePartition](#)

*Apex Developer Guide:* [Platform Cache Partitions](#)

## Platform Event Channel

Represents a channel that you can subscribe to in order to receive a stream of events.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

## More Information

### Feature Name

Metadata Name: PlatformEventChannel

Component Type in 1GP Package Manager UI: Platform Event Channel

### Documentation

*Metadata API Developer Guide:* [PlatformEventChannel](#)

### SEE ALSO:

*Change Data Capture Developer Guide:* [Compose Streams of Change Data Capture Notifications with Custom Channels](#)

## Platform Event Channel Member

Represents an entity selected for Change Data Capture notifications on a standard or custom channel, or a platform event selected on a custom channel.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

## More Information

### Feature Name

Metadata Name: PlatformEventChannelMember

Component Type in 1GP Package Manager UI: Platform Event Channel Member

### Considerations When Packaging

- As of Winter '22, installing a managed package that contains Change Data Capture entity selections no longer causes an installation error. Before Winter '22, installing a managed package that contained Change Data Capture entity selections that were over the default allocation caused package installation errors.
- To package Change Data Capture entity selections, create a custom channel through the PlatformEventChannel metadata type. Then add entity selections to the custom channel through the PlatformEventChannelMember metadata type.

### Documentation

Metadata API Developer Guide: [PlatformEventChannelMember](#)

### SEE ALSO:


[Change Data Capture Developer Guide: Compose Streams of Change Data Capture Notifications with Custom Channels](#)

## Platform Event Subscriber Configuration

Represents configuration settings for a platform event Apex trigger, including the batch size and the trigger's running user.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
	 <b>Note:</b> PlatformEventSubscriberConfig is tied to an Apex trigger. If the package developer removes the Apex trigger, PlatformEventSubscriberConfig is also removed.
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- batchSize
- platformEventConsumer
- user

#### Both Package Developer and Subscriber Can Edit

- None

#### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: PlatformEventSubscriberConfig

Component Type in 1GP Package Manager UI: Platform Event Subscriber Configuration

### Use Case

Override the default running user and batch size of a platform event Apex trigger.

### Relationship to Other Components

PlatformEventSubscriberConfig is tied to an Apex trigger.

### Documentation

*Platform Events Developer Guide:* [Configure the User and Batch Size for Your Platform Event Trigger](#)

## Pricing Action Parameters

Represents a pricing action associated to a context definition and a pricing procedure.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- Pricing Action Parameters Name

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: PricingActionParameters

Component Type in 1GP Package Manager UI: PricingActionParameters

**License Requirements**

Salesforce Pricing permissions

**Relationship to Other Components**

All the components that pricing depends on are packaged along with the Pricing Action Parameters component.

**Documentation**

*Salesforce Help:* [Pricing Action Parameters in Salesforce Pricing](#)


## Pricing Recipe

Represents one out of various data models or sets of entities of a particular cloud that'll be consumed by the pricing data store during design and run time.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- None

**Both Package Developer and Subscriber Can Edit**

- Recipe Name

**Neither Package Developer or Subscriber Can Edit**

- None

## More Information

**Feature Name**

Metadata Name: PricingRecipe

Component Type in 1GP Package Manager UI: PricingRecipe

**Considerations When Packaging**

There are two prerequisites currently. All the associated contexts aren't exported. For decision tables, while exporting, column additions made to the associated objects aren't refreshed during export.

**License Requirements**

Salesforce Pricing permissions

**Relationship to Other Components**

All the components that pricing is dependent on are packaged along with the pricing recipe.

**Documentation**

*Salesforce Help:* [Pricing Recipes](#)

## Process

Use Flow instead.

See [Flow](#)

## Process Flow Migration

Represents a process's migrated criteria and the resulting migrated flow.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes

## Editable Properties After Package Promotion or Installation

**Only Package Developer Can Edit**

- None

**Both Package Developer and Subscriber Can Edit**

- None

**Neither Package Developer or Subscriber Can Edit**

- Description
- Label
- Name

## More Information

### Feature Name

Metadata Name: ProcessFlowMigration

Component Type in 1GP Package Manager UI: Process Flow Migration

### Use Case

Include this component only if you've used Migrate to Flow tool and wish to have pending Scheduled Actions from migrated Processes converted into pending Flow Scheduled Paths in subscriber orgs. This occurs after the migrated Flow is activated in the subscriber org.

### Considerations When Packaging

When packaging a Flow that was migrated from a Process, this component is added automatically. When adding a Flow that was migrated from a Process to a change set, this component would need to be added manually.

### Relationship to Other Components

Flows

### Documentation

*Salesforce Help:* [Migrate Processes and Workflows to Flow](#)

## Product Attribute Set

Represents the ProductAttribute information being used as an attribute such as color\_c, size\_c .

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- Description
- Master Label

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: ProductAttributeSet

### License Requirements

A B2B Commerce or D2C Commerce license and access to Commerce objects is required.

### Usage Limits

An org can have a maximum of 100 product attribute sets.

For each product attribute set, you can have a maximum of five associated product attribute set items.

### Documentation

*Salesforce Help:* [Product Variations and Attributes](#)

*Metadata API Developer Guide:* [ProductAttributeSet](#)

## Product Specification Type

Represents the type of product specification provided by the user to make the product terminology unique to an industry. A product specification type is associated with a product specification record type.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label
- Description

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name



## More Information

### Feature Name

Metadata Name: ProductSpecificationType

Component Type in 1GP Package Manager UI: ProductSpecificationType

### License Requirements

Only Salesforce Admins can set up the product specification type. To create and edit product specification type, the Product Catalog Management Designer permission set is required. To view product specification type, the Product Catalog Management Viewer permission set is required.

### Documentation

*Salesforce Help:* [Product Specification](#)

*Salesforce Help:* [Create Product Specification Type and Product Specification Record Type](#)

## Product Specification Record Type

Represents the relationship between industry-specific product specifications and the product record type.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label
- Record Type
- Product Specification Type

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name
- Is Commercial

## More Information

### Feature Name

Metadata Name: ProductSpecificationRecType

Component Type in 1GP Package Manager UI: ProductSpecificationRecType

### License Requirements

Only Salesforce admins can set up the product specification record type. To create and edit product specification record type, the Product Catalog Management Designer permission set is required. To view product specification record type, the Product Catalog Management Viewer permission set is required.

### Documentation

*Salesforce Help:* [Product Specification](#)

*Salesforce Help:* [Create Product Specification Type and Product Specification Record Type](#)

## Prompts (In-App Guidance)

Represents the metadata related to in-app guidance, which includes prompts and walkthroughs.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## More Information

### Feature Name

Metadata Name: Prompt

Component Type in 1GP Package Manager UI: Prompt

### Documentation

*Metadata API Developer Guide:* [Prompt](#)

*Salesforce Help:* [Guidelines for In-App Guidance in Managed Packages](#)


## Quick Action

Represents a specified create or update quick action for an object that then becomes available in the Chatter publisher.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).


## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Field Overrides

### Both Package Developer and Subscriber Can Edit

- All attributes except Field Overrides

 **Note:** You can only modify managed package quick action layouts in Salesforce Setup. You can't make changes using Metadata API.

### Neither Package Developer or Subscriber Can Edit

## More Information

### Feature Name

Metadata Name: QuickAction

Component Type in 1GP Package Manager UI: Quick Action

### Documentation

*Salesforce Help:* [Quick Actions](#)

## Recommendation Strategy

Represents a recommendation strategy. Recommendation strategies are applications, similar to data flows, that determine a set of recommendations to be delivered to the client through data retrieval, branching, and logic operations.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes, except templates

## More Information

### Feature Name

Metadata Name: RecommendationStrategy

Component Type in 1GP Package Manager UI: Recommendation Strategy

### Use Case

You can use this component to create personalized recommendations for end users. A recommendation displays contextually in Salesforce and prompts the end user to accept or reject the suggestion. When an end user accepts or rejects the recommendation, Salesforce automates a process, such as creating or updating a record.

### Considerations When Packaging

When you package a recommendation strategy, you must manually add object dependencies, such as recommendation, recommendationReaction, and flow.

### Usage Limits

An admin must select an object dependency for Recommendation and RecommendationReaction because object dependencies aren't added automatically.

### Documentation

*Salesforce Help:* [Einstein Next Best Action](#)

## Record Action Deployment

Represents configuration settings for the Actions & Recommendations, Action Launcher, and Bulk Action Panel components.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes

Component Has IP Protection

No



**Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- Channel Configurations
- Deployment Contexts
- HasGuidedActions
- HasRecommendations
- Label
- Recommendations
- SelectableItems
- ShouldLaunchActionOnReject

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: RecordActionDeployment

Component Type in 1GP Package Manager UI: RecordAction Deployment

### Considerations When Packaging

If the record action deployment component uses flows, quick actions, objects, or Next Best Action recommendations, include them in the package too.

### Documentation

*Metadata API Developer Guide:* [RecordActionDeployment](#)

*Salesforce Help:* [Create an Actions & Recommendations Deployment](#)

## Record Alert Data Source Expression Set Definition

Represents information about the data source for a record alert and the association with an expression set definition.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All other fields

### Both Package Developer and Subscriber Can Edit

- ExpressionSetDefinition
- ExpressionSetObject
- IsActive
- RecordAlertDataSource

### Neither Package Developer or Subscriber Can Edit

- FullName
- Metadata

## More Information

### RecAlrtDataSrcExpSetDef

Metadata Name: RecAlrtDataSrcExpSetDef

Component Type in 1GP Package Manager UI: RecAlrtDataSrcExpSetDef

### Documentation

[RecAlrtDataSrcExpSetDef](#) in *Financial Services Cloud Developer Guide*.


## Record Type

Represents the metadata associated with a record type. Record types let you offer different business processes, picklist values, and page layouts to different users. Use this metadata type to create, update, or delete record type definitions for a custom object.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Record Type Label

### Both Package Developer and Subscriber Can Edit

- Active
- Business Process

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: RecordType

Component Type in 1GP Package Manager UI: Record Type

### Considerations When Packaging

- If record types are included in the package, the subscriber's org must support record types to install the package.
- When a new picklist value is installed, it's associated with all installed record types according to the mappings specified by the developer. A subscriber can change this association.
- Referencing an object's record type field in a report's criteria—for example, `Account Record Type`—causes a dependency.
- Summarizing by an object's record type field in a report's criteria—for example, `Account Record Type`—causes a dependency.
- If an object's record type field is included as a column in a report, and the subscriber's org isn't using record types on the object or doesn't support record types, the column is dropped during installation.
- If you install a custom report type that includes an object's record type field as a column, that column is dropped if the org doesn't support record types or the object doesn't have record types defined.

**Documentation**

Metadata API Developer Guide: [RecordType](#)


## RedirectWhitelistUrl

Represents a trusted URL that's excluded from redirection restrictions when the redirectionWarning or redirectBlockModeEnabled field on the SessionSettings Metadata type is set to true.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- Url

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

**Feature Name**

Metadata Name: RedirectWhitelistUrl

Component Type in 1GP Package Manager UI: RedirectWhitelistUrl

**Use Case**

Customers can use a Salesforce security setting to specify what happens when a user clicks a hyperlink that redirects to an untrusted URL outside the salesforce.com domain. The customer can choose to block these redirections or alert the user that the link is taking them outside the Salesforce domain. The URLs in RedirectWhiteListURL are considered trusted for the purpose of that security setting.



If the Experience Cloud site pages, Lightning Experience pages, or custom Visualforce pages in your package include hyperlinks to URLs outside the salesforce.com domain, use RedirectWhitelistURL to ensure that users can access those hyperlinks.

### Considerations When Packaging

When you include a RedirectWhitelistURL in a package, the URLs are trusted for redirections across Salesforce. Because this component modifies the security of the org, we don't recommend that you include RedirectWhitelistURL in packages. Instead, instruct customers to use the Trusted URLs for Redirects Setup page or the RedirectWhitelistURL Metadata API type to add the URLs to their allowlist as part of activating your package. If you choose to include RedirectWhitelistURL components in your package, disclose this change prominently in your package documentation to ensure that your customers are aware of the security modification.

### Usage Limits

The RedirectWhiteListURL component is available in API version 48.0 and later.

### Relationship to Other Components

This component can be used only in conjunction with an Aura or Lightning Web Runtime (LWR) page for an Experience Cloud site, a [Lightning Page](#), or a [Visualforce Page](#).

### Documentation

*Metadata API Developer Guide:* [RedirectWhitelistUrl](#)

Salesforce Help: [Manage Redirections to External URLs](#)

*Metadata API Developer Guide:* [SecuritySettings](#)]


## Referenced Dashboard

Represents the ReferencedDashboard object in CRM Analytics. A referenced dashboard stores information about an externally referenced dashboard.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Label

[Both Package Developer and Subscriber Can Edit](#)

- Description

[Neither Package Developer or Subscriber Can Edit](#)

- Application
- Embed URL
- Template Asset Source Name
- Visibility

## More Information

### Feature Name

Metadata Name: ReferencedDashboard

### License Requirements

Enables Tableau Dashboards in CRM Analytics

## Registered External Service

Represents a registered external service, which provides an extension or integration.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- None

[Both Package Developer and Subscriber Can Edit](#)

- All attributes

[Neither Package Developer or Subscriber Can Edit](#)

- None

## More Information

### Feature Name

Metadata Name: RegisteredExternalService

Component Type in 1GP Package Manager UI: RegisteredExternalService

### Documentation

Object Reference for the Salesforce Platform: [RegisteredExternalService](#)


## RelationshipGraphDefinition

Represents a definition of a graph that you can configure in your organization to traverse object hierarchies and record details, giving you a glimpse of how your business works.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All properties

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: RelationshipGraphDefinition

Component Type in 1GP Package Manager UI: RelationshipGraphDefinition

### Documentation

Metadata API Developer Guide: [RelationshipGraphDefinition](#)

## Remote Site Setting

Represents a remote site setting.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- All attributes except Remote Site Name

### Neither Package Developer or Subscriber Can Edit

- Remote Site Name

## More Information

### Feature Name

Metadata Name: RemoteSiteSettings

**Documentation**

Metadata API Developer Guide: [RemoteSiteSettings](#)

## Report

Represents a custom report.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- All attributes except Report Unique Name

### Neither Package Developer or Subscriber Can Edit

- Report Unique Name

## More Information

### Feature Name

Metadata Name: Report

Component Type in 1GP Package Manager UI: Report

### Considerations When Packaging

If a report includes elements that can't be packaged, those elements are dropped or downgraded, or the package creation fails. For example:

- Hierarchy drill-downs are dropped from activity and opportunities reports.
- Filters on unpackageable fields are automatically dropped (for example, in filters on standard object record types).

- Package upload fails if a report includes filter logic on an unpackageable field (for example, in filters on standard object record types).
- Lookup values on the `Select Campaign` field of standard campaign reports are dropped.
- Reports are dropped from packages if they've been moved to a private folder or to the Unfiled Public Reports folder.
- When a package is installed into an org that doesn't have Chart Analytics 2.0:
  - Combination charts are downgraded instead of dropped. For example, a combination vertical column chart with a line added is downgraded to a simple vertical column chart. A combination bar chart with more bars is downgraded to a simple bar chart.
  - Unsupported chart types, such as donut and funnel, are dropped.

### Documentation

Metadata API Developer Guide: [Report](#)


## Report Type

Represents the metadata associated with a custom report type. Custom report types allow you to build a framework from which users can create and customize reports.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 2GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes except Development Status and Report Type Name

### Both Package Developer and Subscriber Can Edit

- Development Status

### Neither Package Developer or Subscriber Can Edit

- Report Type Name

## More Information

### Feature Name

Metadata Name: ReportType

Component Type in 1GP Package Manager UI: Custom Report Type

### Considerations When Packaging

A developer can edit a custom report type in a managed package after it's released, and can add new fields. Subscribers automatically receive these changes when they install a new version of the managed package. However, developers can't remove objects from the report type after the package is released. If you delete a field in a custom report type that's part of a managed package, and the deleted field is part of bucketing or used in grouping, an error message appears.

### Documentation

*Metadata API Developer's Guide:* [ReportType](#)

## ServiceProcess

Represents a process created in Service Process Studio and its associated attributes.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All other fields

### Both Package Developer and Subscriber Can Edit

- Status
- Description
- ServiceProcessAttribute
- ServiceProcessDependency
- ServiceProcessItemGroup

### Neither Package Developer or Subscriber Can Edit

- FullName

## More Information

### ServiceProcess

Metadata Name: ServiceProcess

Component Type in 1GP Package Manager UI: ServiceProcess

### Documentation

ServiceProcess in *Metadata API Developer Guide*.


## Slack App (Beta)

Represents a Slack app.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	Yes

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- AppKey, AppToken, ClientKey, ClientSecret, SigningSecret, BotScopes, UserScopes, Config, IntegrationUser, DefaultUser

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: SlackApp



Component Type in 1GP Package Manager UI: Slack App

#### Use Case

Represents configuration of a Slack application

#### License Requirements

Connect to Slack Permission

#### Relationship to Other Components

Slack apps reference handlers (Apex classes) and view definition components.

#### Documentation

[Apex SDK for Slack Developer Guide](#)

## Service Catalog Category

Represents the grouping of individual catalog items in Service Catalog.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- ParentCategory

### Both Package Developer and Subscriber Can Edit

- SortOrder
- IsActive
- Image

### Neither Package Developer or Subscriber Can Edit

- FullName

## More Information

### Feature Name

Metadata Name: SvcCatalogCategory

Component Type in 1GP Package Manager UI: Service Catalog Category

**Use Case**

Group your service catalog items together by associating them with a catalog category.

**License Requirements**

Service Catalog Add-On License

Service Catalog Builder Permission Set

**Post Install Steps**

Categories appear in the Service Catalog user UI only if they contain active items.

**Documentation**

*Salesforce Help:* [Create a Catalog Category](#)

## Service Catalog Filter Criteria

Represents an eligibility rule that determines if a Service Catalog user has access to a catalog item.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All fields

### Both Package Developer and Subscriber Can Edit

- All fields

### Neither Package Developer or Subscriber Can Edit

- FullName

## More Information

**Feature Name**

Metadata Name: SvcCatalogFilterCriteria

Component Type in 1GP Package Manager UI: Service Catalog Item Definition

**Use Case**

Use the filter criteria to filter on catalog items.

**License Requirements**

Service Catalog Add-On License

Service Catalog Builder Permission Set

**Relationship to Other Components**

Service catalog filter criteria are related to a catalog item definition.

**Documentation**

*Salesforce Help:* [Create a Catalog Category](#)

## Service Catalog Item Definition

Represents the entity associated with a specific, individual service available in the Service Catalog.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- Flow

#### Both Package Developer and Subscriber Can Edit

- Status
- Description
- InternalNotes
- Image
- IsFeatured
- IsPublic

#### Neither Package Developer or Subscriber Can Edit

- FullName

## More Information

### Feature Name

Metadata Name: SvcCatalogItemDef

Component Type in 1GP Package Manager UI: Service Catalog Item Definition

### Use Case

Create a service catalog item that employees can request in the Service Catalog user UI.

### Considerations When Packaging

Subscribers can't change properties stored in the catalog item fulfillment flow unless they make a clone of the item and its related flow.

### License Requirements

Service Catalog Add-On License

Service Catalog Builder Permission Set

### Usage Limits

The org can have only 1000 SvcCatalogItemDefs, including those items installed from a managed package.

### Post Install Steps

If the item was installed in draft mode, it must be activated before employees can see it in the Service Catalog user UI.

### Relationship to Other Components

SvcCatalogItemDef requires a relationship with a catalog category.

### Documentation

*Salesforce Help:* [Create a Catalog Item](#)

## Service Catalog Fulfillment Flow

Represents the flow associated with a specific catalog item in the Service Catalog.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Description

- Flow
- Icon

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- FullName

## More Information

### Feature Name

Metadata Name: SvcCatalogFulfillmentFlow

Component Type in 1GP Package Manager UI: Service Catalog Fulfillment Flow

### Use Case

Make a screen flow available in the Service Catalog builder. You can also use SvcCatalogFulfillmentFlow metadata to describe the flow and its inputs in the builder, enabling a clicks-not-code experience for providing inputs to the flow.

### License Requirements

Service Catalog Add-On License

Service Catalog Builder Permission Set

### Post Install Steps

Fulfillment flows appear in the Service Catalog builder only if the underlying screen flow is active in the org.

### Relationship to Other Components

SvcCatalogFulfillmentFlow must be related to a FlowDefinition.

SvcCatalogFulfillmentFlow can have related SvcCatalogFulfillFlowItem records.

### Documentation

*Salesforce Help:* [Catalog Item Fulfillment Flows](#)

## Stationary Asset Environmental Source Record Type Configuration

Represents the setup object that contains the mapping between the Stationary Asset Environmental Source record type and internal enums. You can primarily use this object for calculations across different record types.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: StnryAssetEnvSrcCnfg

Component Type in 1GP Package Manager UI: Stationary Asset Environmental Source Record Type Configuration

### Use Case

You can use this component to build on top of the current Net Zero Cloud data model and carbon accounting capability to create new stationary asset types for end users.

### License Requirements

- Net Zero Cloud Growth license or Net Zero Cloud Starter license
- Net Zero Cloud Manager permissions set

### Post Install Steps

Enable these org settings:

- Net Zero Cloud
- Manage Carbon Accounting

### Documentation

- [Salesforce Help: Set Up Record Types for Net Zero Cloud](#)
- [Salesforce Help: Create a Stationary Asset Environmental Source Record](#)

## Static Resource


Represents a static resource file, often a code library in a ZIP file.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP and 2GP packages.

Component Has IP Protection	No
-----------------------------	----

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- File

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: StaticResource

Component Type in 1GP Package Manager UI: Static Resource

### Documentation

*Metadata API Developer Guide:* [StaticResource](#)

## Streaming App Data Connector

Represents the connection information specific to Web and Mobile Connectors.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- AppIdentifier
- DataConnectorType
- StreamingAppDataConnectorType

## More Information

### Feature Name

Metadata Name: StreamingAppDataConnector

### Use Case

The StreamingAppDataConnector is included in a package when you add a data stream (DataStreamDefinition). You need this component if you want to package a web or mobile data stream.

### Post Install Steps

The package doesn't contain any connection information. The package subscriber must create the connection in their subscriber org and then select that connection when they deploy the data kit.

### Documentation

*Data Cloud Reference Guide:* [Capture Web Interactions](#)

*Data Cloud Reference Guide:* [Integrate your Mobile Applications](#)

## Sustainability UOM

Represents information about the additional unit of measure values defined by a customer.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No



## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: SustainabilityUom

Component Type in 1GP Package Manager UI: Sustainability Unit of Measure

### License Requirements

- Net Zero Cloud Growth license or Net Zero Cloud Starter license
- Net Zero Cloud Manager permissions set

### Post Install Steps

Enable these org settings:

- Net Zero Cloud
- Manage Carbon Accounting

### Documentation

- *Salesforce Help:* [Create Custom Units of Measure](#)

## Sustainability UOM Conversion

Represents information about the unit of measure conversion for the additional fuel types defined by a customer.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: SustnUomConversion

Component Type in 1GP Package Manager UI: Sustainability Unit of Measure Conversion

### License Requirements

- Net Zero Cloud Growth license or Net Zero Cloud Starter license
- Net Zero Cloud Manager permissions set

### Post Install Steps

Enable these org settings:

- Net Zero Cloud
- Manage Carbon Accounting

### Documentation

- *Salesforce Help*: [Create a Unit of Measure Conversion for a Custom Fuel Type](#)


## Timeline Object Definition

Represents the container that stores the details of a timeline configuration. You can use this resource with Salesforce objects to see their records' related events in a linear time-sorted view.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label
- FullName
- Definition
- IsActive

### Both Package Developer and Subscriber Can Edit

- Label
- FullName
- Definition
- IsActive

### Neither Package Developer or Subscriber Can Edit

- BaseObject

## More Information

### Feature Name

Metadata Name: TimelineObjectDefinition

Component Type in 1GP Package Manager UI: Timeline Object Definition

### Use Case

Provides out-of-the-box Timeline object definitions.

### License Requirements

Industries Health Cloud or any other License that has Timeline Permission enabled in them.

### Legacy Component

There's a legacy Timeline component in the Health Cloud Package which is being deprecated in favor of this component.

### Documentation

*Health Cloud Developer Guide:* [TimelineObjectDefinition](#)


## Timesheet Template

Represents a template for creating time sheets in Field Service.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP packages only.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## More Information

### Feature Name

Metadata Name: TimesheetTemplate

## Translation

Add translations to your managed packages.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: Translation


### Relationship to Other Components

When you add this component to a first-generation managed package, the [Custom Object Translation](#) component is automatically added to your package.

For details on how subscribers can override translations after installing a package, see [Override Translations in Second-Generation Managed Packages and Unlocked Packages](#) in the Salesforce DX Developer Guide.

## Considerations When Packaging (Beta)

Enable Language Extension Packages in Dev Hub to create language extension packages that contain translations of components in other packages.

 **Note:** This feature is a Beta Service. Customer may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms.

Language extension packages can only contain translations: Translations and CustomObjectTranslations. If a base package includes components that can't be translated, those components aren't included when you create a language extension package.

To remove translations delivered by a package extension, uninstall the base package and all related extensions, then reinstall the base package and any other desired extensions. Otherwise, translations delivered by the extension remain until you uninstall all packages with that namespace.


## UI Object Relation Config

Represents the admin-created configuration of the object relation UI component.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Reference Name
- Developer Name
- IsActive

### Both Package Developer and Subscriber Can Edit

- IsActive

### Neither Package Developer or Subscriber Can Edit

- ContextObject

## More Information

### Feature Name

Metadata Name: UIObjectRelationConfig

Component Type in 1GP Package Manager UI: UI Object Relation Configuration

### Use Case

Provides out-of-the-box relationship card configuration in Health Cloud.

### License Requirements

Industries Health Cloud, Industries Insurance, or Industries Automotive licenses

### Documentation

*Salesforce Help:* [Set Up Provider Relationship Cards to Show Practitioner Information](#)

## User Access Policy


Represents a user access policy.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

To confirm whether this component is available in managed 1GP, managed 2GP, or both package types, see [Metadata Coverage Report](#).

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Name
- Label
- User Criteria Filters
- Actions

### Both Package Developer and Subscriber Can Edit

- Order (only subscriber editable)
- Status (only subscriber editable)
- Trigger Type (only subscriber editable)

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: UserAccessPolicy

Component Type in 1GP Package Manager UI: User Access Policy

### Usage Limits

User access policies have their status set to Design when installed and can be activated by the subscriber. Subscribers can have up to 200 active policies at one time.

### Post Install Steps

The subscriber can activate user access policies so that they run automatically when a user record matching the policy's user criteria is created, updated, or both.

### Documentation

*Metadata API Developer Guide:* [UserAccessPolicy](#)

*Salesforce Help:* [User Access Policies](#)


## Validation Rule

Represents a validation rule, which is used to verify that the data a user enters in a record is valid and can be saved.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Error Condition Formula
- Error Location
- Error Message

### Both Package Developer and Subscriber Can Edit

- Active

### Neither Package Developer or Subscriber Can Edit

- Rule Name

## More Information

### Feature Name

Metadata Name: ValidationRule

Component Type in 1GP Package Manager UI: Validation Rule

### Considerations When Packaging

For custom objects that are packaged, any associated validation rules are implicitly packaged as well.

### Documentation

*Metadata API Developer Guide:* [ValidationRule](#)

## Vehicle Asset Emissions Source Record Type Configuration

Represents the setup object that contains the mapping between the Vehicle Asset Emissions Source record type and internal enums. You can primarily use this object for calculations across different record types.



## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- All attributes

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- None

## More Information

### Feature Name

Metadata Name: VehicleAssetEmssnSrcCnfg

Component Type in 1GP Package Manager UI: Vehicle Asset Emissions Source Record Type Configuration

### Use Case

You can use this component to build on top of the current Net Zero Cloud data model and carbon accounting capability to create new vehicle asset types for end users.

### License Requirements

- Net Zero Cloud Growth license or Net Zero Cloud Starter license
- Net Zero Cloud Manager permissions set

### Post Install Steps

Enable these org settings:

- Net Zero Cloud
- Manage Carbon Accounting

### Documentation

- [Salesforce Help: Set Up Record Types for Net Zero Cloud](#)
- [Salesforce Help: Create a Vehicle Asset Emissions Source Record](#)


## View Definition (Beta)

Represents a view definition on a Slack app.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

### Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- TargetType, Content, Description

[Both Package Developer and Subscriber Can Edit](#)

- None

[Neither Package Developer or Subscriber Can Edit](#)

- None

### More Information

#### Feature Name

Metadata Name: ViewDefinition

Component Type in 1GP Package Manager UI: View Definition

#### Use Case

Represents a view within a Slack application

#### License Requirements

Connect to Slack Permission

#### Relationship to Other Components

View definitions are referenced by Slack apps.

**Documentation**[Apex SDK for Slack Developer Guide](#)

## Virtual Visit Config

Represents an external video provider configuration, which relays events from Salesforce to the provider.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- ComprehendServiceType
- ExperienceCloudSiteUrl
- ExternalRoleIdentifier
- Label
- MessagingRegion
- NamedCredential
- StorageBucketName
- UsageType
- VideoCallApptTypeValue
- VideoControlRegion
- VisitRegion

#### Both Package Developer and Subscriber Can Edit

- None

#### Neither Package Developer or Subscriber Can Edit

- Name

### More Information

**Feature Name**

Metadata Name: VirtualVisitConfig

**Documentation***Salesforce Help:* [Virtual Care](#)

## Visualforce Component

Represents a Visualforce component.

### Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	Yes

If a developer removes a public Visualforce component from a new version of your 1GP managed package, the component is removed from the subscriber's org upon upgrade. If the Visualforce component is global, it remains in the subscriber org until the administrator deletes it.

For 2GP packages, Visualforce components are hard deleted, and only components that aren't marked as global can be removed from a package.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

### Editable Properties After Package Promotion or Installation

#### Only Package Developer Can Edit

- API Version
- Description
- Label
- Markup

#### Both Package Developer and Subscriber Can Edit

- None

#### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: ApexComponent

### Documentation

[Visualforce Components](#)

## Visualforce Page

Represents a Visualforce page.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in 1GP and 2GP packages.
Component Has IP Protection	No

If a developer removes a public Visualforce component from a new version of your package, the component is removed from the subscriber's org upon upgrade. If the Visualforce component is global, it remains in the subscriber org until the administrator deletes it.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- API Version
- Description
- Label
- Markup

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: ApexPage

Component Type in 1GP Package Manager UI: Visualforce Page


## Wave Analytic Asset Collection

A collection of CRM Analytics assets.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Folder
- Items
- Label

### Both Package Developer and Subscriber Can Edit

- Color
- Description
- Shares

### Neither Package Developer or Subscriber Can Edit

- Collection Type

## More Information

### Feature Name

Metadata Name: WaveAnalyticAssetCollection

Component Type in 1GP Package Manager UI: Wave Analytic Asset Collection

### Use Case

Represents a collection of CRM Analytics assets.

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics

### Documentation

*Salesforce Help:* [Curate and Share Insights with Collections](#)


## Wave Application

A CRM Analytics application.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Label

[Both Package Developer and Subscriber Can Edit](#)

- Asset Icon
- Description
- Shares

Neither Package Developer or Subscriber Can Edit

- Folder
- Template Origin
- Template Version

## More Information

### Feature Name

Metadata Name: WaveApplication

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics


## Wave Component

A CRM Analytics dashboard component.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).



## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label

### Both Package Developer and Subscriber Can Edit

- Description

### Neither Package Developer or Subscriber Can Edit

- Application
- Template Asset Source Name

## More Information

### Feature Name

Metadata Name: WaveComponent

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics


## Wave Dataflow

A CRM Analytics data prep dataflow.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label

### Both Package Developer and Subscriber Can Edit

- Description

### Neither Package Developer or Subscriber Can Edit

- Application
- Dataflow Type

## More Information

### Feature Name

Metadata Name: WaveDataflow

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics


## Wave Dashboard

A CRM Analytics dashboard.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label

### Both Package Developer and Subscriber Can Edit

- Description

### Neither Package Developer or Subscriber Can Edit

- Application
- Date Version
- Template Asset Source Name

## More Information

### Feature Name

Metadata Name: WaveDashboard

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics


## Wave Dataset

A CRM Analytics dataset.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label

### Both Package Developer and Subscriber Can Edit

- Description

### Neither Package Developer or Subscriber Can Edit

- Application
- Template Asset Source Name
- Type

## More Information

### Feature Name

Metadata Name: WaveDataset

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics


## Wave Lens

A CRM Analytics lens.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label

### Both Package Developer and Subscriber Can Edit

- Description
- Visualization Type

### Neither Package Developer or Subscriber Can Edit

- Application
- Datasets
- Template Asset Source Name

## More Information

### Feature Name

Metadata Name: WaveLens

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics


## Wave Recipe

A CRM Analytics data prep recipe.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label

### Both Package Developer and Subscriber Can Edit

- Description
- Security Predicate
- Target Dataset Alias

### Neither Package Developer or Subscriber Can Edit

- Application
- Dataflow
- Format
- Template Asset Source Name

## More Information

### Feature Name

Metadata Name: Wave Recipe

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics

## Wave Template Bundle


A CRM Analytics template bundle.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes

Component Has IP Protection	No
-----------------------------	----

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label

### Both Package Developer and Subscriber Can Edit

- Asset Icon
- Description

### Neither Package Developer or Subscriber Can Edit

- Asset Version
- Template Type

## More Information

### Feature Name

Metadata Name: WaveTemplateBundle

### Considerations When Packaging

Analytics assets are installed in subscriber orgs via Analytics Templates using the WaveTemplateBundle. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics

## Wave Xmd


The extended metadata for CRM Analytics dataset fields and their formatting for dashboards and lenses.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	Yes

Package Developer Can Remove Component From Package	Yes
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Label

### Both Package Developer and Subscriber Can Edit

- Dates
- Dimensions
- Measures
- Organizations
- Wave Visualization

### Neither Package Developer or Subscriber Can Edit

- Application
- Dataset
- Dataset Connector
- Dataset Fully Qualified Name
- Origin
- Type

## More Information

### Feature Name

Metadata Name: WaveXmd

### Considerations When Packaging

Analytics assets should be installed in subscriber orgs via Analytics Templates. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org. For more information, see the [Analytics Templates Developer Guide](#).

### License Requirements

Manage CRM Analytics

## Web Store Template

Represents a configuration for creating commerce stores.



## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	No

## More Information

### Feature Name

Metadata Name: WebStoreTemplate


### Documentation

Metadata API Developer Guide: [WebStoreTemplate](#)

## Workflow Alert

WorkflowAlert represents an email alert associated with a workflow rule.

## Component Manageability Rules

 **Note:** When creating a new package or package version, use the Flow component instead of Workflow components. If your managed package already includes Workflow components, come up with a plan to migrate to use Flow.

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Both protected and non-protected components can be removed.
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- None

[Both Package Developer and Subscriber Can Edit](#)

- Additional Emails
- Email Template
- From Email Address
- Recipients

[Neither Package Developer or Subscriber Can Edit](#)

- Description

## More Information

### Feature Name

Metadata Name: Workflow


- Salesforce prevents you from uploading workflow alerts that have a public group, partner user, or role recipient. Change the recipient to a user before uploading your app. During installation, Salesforce replaces that user with the user installing the app, and the installer can customize it as necessary.
- You can package workflow rules and associated workflow actions, such as email alerts and field updates. However, any time-based triggers aren't included in the package. Notify your installers to set up any time-based triggers that are essential to your app.
- References to a specific user in workflow actions, such as the email recipient of a workflow email alert, are replaced by the user installing the package. Sometimes workflow actions referencing roles, public groups, account team, opportunity team, or case team roles aren't uploaded.
- References to an org-wide address, such as the `From_email_address` of a workflow email alert, are reset to Current User during installation.

This component can be marked as protected. For more details, see [Protected Components](#) in the *First-Generation Managed Packaging Developer Guide*.

## Workflow Field Update

WorkflowFieldUpdate represents a workflow field update.

## Component Manageability Rules

 **Note:** When creating a new package or package version, use the Flow component instead of Workflow components. If your managed package already includes Workflow components, come up with a plan to migrate to use Flow.

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages. Both protected and non-protected components can be removed.

Component Has IP Protection	No
-----------------------------	----

---

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Field Value
- Formula Value

### Both Package Developer and Subscriber Can Edit

- Lookup

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: Workflow

Component Type in 1GP Package Manager UI: Workflow Field Update

- Salesforce prevents you from uploading workflow field updates that change an `OWNER` field to a queue. Change the updated field value to a user before uploading your app. During installation, Salesforce replaces that user with the user installing the app, and the installer can customize it as necessary.
- Salesforce prevents you from uploading workflow rules, field updates, and outbound messages that reference a record type on a standard or managed-installed object.
- You can package workflow rules and associated workflow actions, such as email alerts and field updates. However, any time-based triggers aren't included in the package. Notify your installers to set up any time-based triggers that are essential to your app.

This component can be marked as protected. For more details, see [Protected Components](#) in the *First-Generation Managed Packaging Developer Guide*.

## Workflow Knowledge Publish

WorkflowKnowledgePublish represents Salesforce Knowledge article publishing actions and information.

## Component Manageability Rules

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes, if protected

Component Has IP Protection	No
-----------------------------	----

---

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Action
- Description
- Unique Name

### Both Package Developer and Subscriber Can Edit

- None

### Neither Package Developer or Subscriber Can Edit

- Object Name

## More Information

### Feature Name

Metadata Name: WorkflowKnowledgePublish

Component Type in 1GP Package Manager UI: Knowledge Action

### Considerations When Packaging

WorkflowKnowledgePublish can only be installed in Salesforce Classic orgs with Knowledge enabled.

WorkflowKnowledgePublish includes the article type \* \_\_kav\_\_, which is not supported by Lightning Knowledge.

If you try to install WorkflowKnowledgePublish into an org with Lightning Knowledge enabled, this message is displayed: When Lightning Knowledge is enabled, you can't add an article type.

### License Requirements

Salesforce Classic orgs with Knowledge enabled can use this package.


### Documentation

*Salesforce Help:* [Create Workflow Actions for Knowledge](#)

## Workflow Outbound Message

WorkflowOutboundMessage represents an outbound message associated with a workflow rule.

## Component Manageability Rules

 **Note:** When creating a new package or package version, use the Flow component instead of Workflow components. If your managed package already includes Workflow components, come up with a plan to migrate to use Flow.

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
-----------------	---

---

Component Is Updated During Package Upgrade	Yes
---	-----

---

Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages. Both protected and non-protected components can be removed.
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Endpoint URL
- Fields to Send
- Send Session ID

### Both Package Developer and Subscriber Can Edit

- User to Send As

### Neither Package Developer or Subscriber Can Edit

- Name

## More Information

### Feature Name

Metadata Name: Workflow

Component Type in 1GP Package Manager UI: Workflow Outbound Message


Salesforce prevents you from uploading workflow rules, field updates, and outbound messages that reference a record type on a standard or managed-installed object.

This component can be marked as protected. For more details, see [Protected Components](#) in the *First-Generation Managed Packaging Developer Guide*.

## Workflow Rule

This metadata type represents a workflow rule.


## Component Manageability Rules

 **Note:** When creating a new package or package version, use the Flow component instead of Workflow components. If your managed package already includes Workflow components, come up with a plan to migrate to use Flow.

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes

Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages.
Component Has IP Protection	No

 **Note:** When a package developer removes this component from a package, the component remains in a subscriber's org after they install the upgraded package. The admin of the subscriber's org can then delete the component, if desired.

Removing components from managed 1GP or 2GP packages requires approval from Salesforce. To request access to the component removal feature, log a support case in the [Salesforce Partner Community](#).

For more details on 2GP component removal, see [Remove Metadata Components from Second-Generation Managed Packages](#).

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- Description
- Evaluation Criteria
- Rule Criteria

### Both Package Developer and Subscriber Can Edit

- Active

### Neither Package Developer or Subscriber Can Edit

- Rule Name


## More Information

- Feature Name:  
Metadata Name: Workflow  
Component Type in 1GP Package Manager UI: Workflow Rule
- Salesforce prevents you from uploading workflow rules, field updates, and outbound messages that reference a record type on a standard or managed-installed object.
- Developers can associate or disassociate workflow actions with a workflow rule at any time. These changes, including disassociation, are reflected in the subscriber's org upon install. In managed packages, a subscriber can't disassociate workflow actions from a workflow rule if it was associated by the developer.
- On install, all workflow rules newly created in the installed or upgraded package, have the same activation status as in the uploaded package.
- You can't package workflow rules with time triggers.

## Workflow Task

This metadata type references an assigned workflow task.

## Component Manageability Rules

 **Note:** When creating a new package or package version, use the Flow component instead of Workflow components. If your managed package already includes Workflow components, come up with a plan to migrate to use Flow.

Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is created and promoted to the released state.

Packageable In:	Second-Generation Managed Packages (2GP), First-Generation Managed Packages (1GP)
Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes. Supported in both 1GP and 2GP packages. Both protected and non-protected components can be removed.
Component Has IP Protection	No

## Editable Properties After Package Promotion or Installation

### Only Package Developer Can Edit

- None

### Both Package Developer and Subscriber Can Edit

- Assign To
- Comments
- Due Date
- Priority
- Record Type
- Status

### Neither Package Developer or Subscriber Can Edit

- Subject

## More Information

### Feature Name

Metadata Name: Workflow

Component Type in 1GP Package Manager UI: Workflow Task

- Salesforce prevents you from uploading workflow tasks that are assigned to a role. Change the `Assigned To` field to a user before uploading your app. During installation, Salesforce replaces that user with the user installing the app, and the installer can customize it as necessary.
- This component can be marked as protected. For more details, see [Protected Components](#) in the *First-Generation Managed Packaging Developer Guide*.

# Behavior of Specific Metadata in Second-Generation Managed Packages

---

Learn how profiles and namespace visibility are handled for second-generation managed packages.

## [Package Data Cloud Metadata Components](#)

Utilize the power of Data Cloud in your apps by including Data Cloud metadata in your managed packages. Working with Data Cloud metadata has some unique requirements. Review these details to understand how to work with Data Cloud metadata in your packages.

## [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.

## [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.

## [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.

## [Namespace-Based Visibility for Apex Classes in Second-Generation Managed Packages](#)

The `@NamespaceAccessible` makes public Apex in a package available to other packages that use the same namespace. Without this annotation, Apex classes, methods, interfaces, and properties defined in a second-generation managed package aren't accessible to the other packages with which they share a namespace. Apex that is declared global is always available across all namespaces, and needs no annotation.

## [Work with Services Outside of Salesforce](#)

## [Package Connected Apps in Second-Generation Managed Packaging](#)

Add a connected app to a second-generation managed package.

## [Test and Respond to the New Order Save Behavior](#)

If you created any type of package that includes the Order object, the installed package sometimes doesn't work, and package upgrades or new package installations are blocked.



## Package Data Cloud Metadata Components

Utilize the power of Data Cloud in your apps by including Data Cloud metadata in your managed packages. Working with Data Cloud metadata has some unique requirements. Review these details to understand how to work with Data Cloud metadata in your packages.

### Enable Data Cloud for Scratch Orgs

To create scratch orgs or package Data Cloud components, you must have Dev Hub enabled in your Partner Business Org. Then, you can request that Data Cloud for Scratch Orgs be enabled by logging a case with [Salesforce Partner Support](#). Data Cloud for Scratch Orgs is only available to scratch orgs associated with the Dev Hub in your Partner Business Org.

### Create Dedicated Data Cloud Packages

When creating a managed package with Data Cloud metadata, you must isolate the Data Cloud metadata from the other Salesforce metadata by creating separate packages that contain only Data Cloud metadata. Then create package dependencies between your dedicated Data Cloud package and any related packages.

### Add Data Cloud Metadata to a Data Kit

When packaging Data Cloud metadata, you must add the metadata to a data kit, and then add the data kit to the managed package. Data kits streamline the package creation and installation process. For more details, see [Packages and Data Kits](#) in the *Data Cloud Developer Guide*.

### Data Cloud One Companion Connected Orgs

Packages can't be installed on orgs that are connected to Data Cloud as Data Cloud One companion orgs. When Data Cloud customers install a managed package containing Data Cloud metadata, they must install the package in their Data Cloud home org. For customers using Data Cloud One, any package installed into data spaces shared with a companion org are automatically installed into the companion org. Companion orgs automatically receive package updates when the package in the home org is upgraded.

These package-related actions can't be initiated in companion connected orgs, and must instead be initiated in the Data Cloud One home org.

- Installing a managed package
- Uninstalling a managed package
- Deleting package metadata
- Receiving a package push upgrade

#### SEE ALSO:

[Data Cloud Developer Guide: Get Started with Data Cloud Development](#)

[Data Cloud Developer Guide: Workflow for Data Cloud Second-Generation Managed Packages](#)

[Data Cloud Developer Guide: Metadata Components for Data Cloud Cheat Sheet](#)

[Salesforce Help: Connect Salesforce CRM Orgs to Data Cloud](#)

## 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 either a full or partial sandbox copy using a template, protected custom objects don't display in the list of objects to copy. As a result, data contained in the records of protected custom objects isn't copied to these sandboxes. If a full sandbox is created without selecting a sandbox template, data from protected custom objects is copied to the sandbox.


SEE ALSO:

[Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs](#)

## 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.

## 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

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?	<ul style="list-style-type: none"> <li>Assigned custom apps</li> <li>Custom object permissions</li> <li>External object permissions</li> <li>Custom field permissions</li> <li>Custom metadata types permissions</li> <li>Custom permissions</li> <li>Custom settings permissions</li> <li>Custom tab visibility settings</li> <li>Apex class access</li> <li>Visualforce page access</li> <li>External data source access</li> <li>Record types</li> </ul> <p><b>Note:</b> Although permission sets include standard tab visibility settings, these settings can't be packaged as permission set components.</p> <p>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.</p>	<ul style="list-style-type: none"> <li>Assigned custom apps</li> <li>Assigned connected apps</li> <li>Tab settings</li> <li>Page layout assignments</li> <li>Record type assignments</li> <li>Custom field permissions</li> <li>Custom metadata type permissions</li> <li>Custom object permissions</li> <li>Custom permissions</li> <li>Custom settings permissions</li> <li>External object permissions</li> <li>Apex class access</li> <li>Visualforce page access</li> <li>External data source access</li> </ul>

### 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

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: <ul style="list-style-type: none"> <li>• Fields (CustomField)</li> <li>• Objects (CustomObject),</li> <li>• Tabs (CustomTab)</li> <li>• Apps (CustomApplication)</li> <li>• Apex classes (ApexClass)</li> <li>• Apex pages (ApexPage)</li> <li>• Layouts (Layout)</li> <li>• Record types (RecordType)</li> <li>• Custom permissions (CustomPermission)</li> <li>• Custom settings (CustomSetting)</li> <li>• Custom metadata types (CustomMetadata)</li> </ul>
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
	<p>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.</p> <p>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.</p>	
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.

### 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.


### How We Handle Profile Settings in Second-Generation Managed Packages

During package version creation for unlocked or second-generation managed packages, the build system inspects the contents of all profiles in the DX project directory, not just the directory specified in the path, and preserves only the profile settings that are directly related to the metadata in the package. The profile itself, and any profile settings unrelated to the package's metadata are discarded from the package.

## 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:

[Salesforce Help: Create a Permission Set Group](#)

[Salesforce Help: Permission Set Group 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.

## How We Handle Profile Settings in Second-Generation Managed Packages

During package version creation for unlocked or second-generation managed packages, the build system inspects the contents of all profiles in the DX project directory, not just the directory specified in the path, and preserves only the profile settings that are directly related to the metadata in the package. The profile itself, and any profile settings unrelated to the package's metadata are discarded from the package.

During package installation, the preserved profile settings are applied only to existing profiles in the subscriber org. The profile itself isn't installed in the subscriber org.

To control which profile settings are included, use the `scopeProfiles` parameter in the [project configuration file](#).

 **Note:** Packages that contain only profiles and no additional metadata aren't allowed and fail during package version creation.


When you select...	The packaged profile settings are applied to...	This installation option is available via...
Install for Admins Only	The System Administrator profile in the subscriber org. CRUD access to custom objects is granted automatically to the System Administration profile.	<ul style="list-style-type: none"> <li>The package installer page</li> <li>Salesforce CLI <code>sf package install</code> command</li> </ul> The default behavior for CLI-based package installs is to install for admins only.
Install for All Users	The System Administrator profile and all cloned profiles in the subscriber org. CRUD access to custom objects is granted automatically to the System Administration profile, and all cloned profiles. <a href="#">Standard profiles</a> can't be modified.	<ul style="list-style-type: none"> <li>The package installer page</li> <li>Salesforce CLI <code>sf package install</code> command</li> </ul> To install for all users via the CLI, include the security type parameter. <code>sf package install --security-type AllUsers</code>
Install for Specific Profiles	Specific profiles in the subscriber org. This selection lets the person installing your package determine how to map the profile settings you packaged to specific profiles in their org.	<ul style="list-style-type: none"> <li>The package installer page</li> </ul> Not available for CLI-based package installations.

To test the behavior of your packaged profile, install your package in a scratch org.

1. From Setup, enter *Profile* in the Quick Find box, and then locate and inspect the profiles you selected during package installation.
2. Check whether your profile settings have been applied to that profile.

Repeat this step for any other profile you expect to contain your profile settings. Don't look for the profile name you created; we apply profile settings to existing profiles in the subscriber org.

Whenever possible, use package permission sets instead of profile settings. Subscribers who install your package can easily assign your permission set to their users.

 **Note:** During a push upgrade, some profile settings related to Apex classes and field-level security aren't automatically assigned to the System Admin profile. To ensure that user access is set up correctly after a push upgrade, communicate with your customer. Make sure they review and update their profile settings after a push upgrade.

## 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
```

## Namespace-Based Visibility for Apex Classes in Second-Generation Managed Packages

The `@NamespaceAccessible` makes public Apex in a package available to other packages that use the same namespace. Without this annotation, Apex classes, methods, interfaces, and properties defined in a second-generation managed package aren't accessible to the other packages with which they share a namespace. Apex that is declared global is always available across all namespaces, and needs no annotation.

### Considerations for Apex Accessibility Across Packages

- You can't use the `@NamespaceAccessible` annotation for an `@AuraEnabled` Apex method.
- You can add or remove the `@NamespaceAccessible` annotation at any time, even on managed and released Apex code. Make sure that you don't have dependent packages relying on the functionality of the annotation before adding or removing it.
- When adding or removing `@NamespaceAccessible` Apex from a package, consider the impact to customers with installed versions of other packages that reference this package's annotation. Before pushing a package upgrade, ensure that no customer is running a package version that would fail to fully compile when the upgrade is pushed.
- If a public interface is declared as `@NamespaceAccessible`, then all interface members inherit the annotation. Individual interface members can't be annotated with `@NamespaceAccessible`.
- If a public or protected variable or method is declared as `@NamespaceAccessible`, its defining class must be either global or public with the `@NamespaceAccessible` annotation.
- If a public or protected inner class is declared as `@NamespaceAccessible`, its enclosing class must be either global or public with the `@NamespaceAccessible` annotation.

This example shows an Apex class marked with the `@NamespaceAccessible` annotation. The class is accessible to other packages within the same namespace. The first constructor is also visible within the namespace, but the second constructor isn't.

```
// A namespace-visible Apex class
@NamespaceAccessible
public class MyClass {
    private Boolean bypassFLS;

    // A namespace-visible constructor that only allows secure use
    @NamespaceAccessible
    public MyClass() {
        bypassFLS = false;
    }

    // A package private constructor that allows use in trusted contexts,
    // but only internal to the package
    public MyClass (Boolean bypassFLS) {
        this.bypassFLS = bypassFLS;
    }
    @NamespaceAccessible
    protected Boolean getBypassFLS() {
        return bypassFLS;
    }
}
```

```
}  
}
```


**SEE ALSO:**[Namespaces for Second-Generation Managed Packages](#)[Create and Register Your Namespace for Second-Generation Managed Packages](#)[Link a Namespace to a Dev Hub Org](#)

## 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.

## Package Connected Apps in Second-Generation Managed Packaging

Add a connected app to a second-generation managed package.

Prerequisites: [Create a connected app](#).

1. Create a first-generation managed package (1GP) and add the connected app. It's fine if the connected app is the only component in the package. Use the same namespace as the 2GP package for the 1GP package.  
Take note of the version number of the connected app; you'll use this number later.
2. From your packaging org, upload the 1GP package to create a package version.
3. Promote the 1GP version to the released state.  
Promoting the 1GP version allows the connected app to be included in a second-generation managed package. You don't need to install the 1GP version into an org.
4. Navigate to the source for your connected app, or pull the source from the org where the connected app is being developed.
5. Create a source `.xml` file in your 2GP directory and reference the connected app you want to include. See the *Sample Source File* section.
6. Create a second-generation managed package and add in the source code for the connected app. Add the source code manually. You can't use `sf project retrieve start` or the `retrieve()` Metadata API call to add the source code.



### Example: Sample Source File

```
<ConnectedApp xmlns="http://soap.sforce.com/2006/04/metadata">
  <developerName>db_0110_ns4__A_Connected_App</developerName>
  <label>A Connected App</label>
  <version>1.0</version>
</ConnectedApp>
```

The `developerName` is the combination of your namespace (`db_0110_ns4`) and the name of your connected app (`A_Connected_App`).

The `version` specified in the source file is the version number of the connected app. Use decimal formatting when specifying the version number. The version number must match the version number of the connected app before it was added to the 1GP managed package.



**Note:** When you add a connected app to a 1GP package, and upload the package, the version number of the connected app is auto incremented. For example, when version 4.0 of a connected app is added to a 1GP package, the package version increments the version number of the connected app from 4.0 to 5.0. When creating the source file for your 2GP package, specify the version number of the connected app before it was uploaded into a 1GP package, in this case, 4.0.

## Test and Respond to the New Order Save Behavior


If you created any type of package that includes the Order object, the installed package sometimes doesn't work, and package upgrades or new package installations are blocked.

[Enable New Order Save Behavior](#) addresses an issue where Salesforce didn't correctly evaluate custom application logic on records associated with the Order object.

To ensure the expected behavior, you must test the Enable New Order Save Behavior release update. Starting in Winter '21, if a subscriber org relies on a different order save behavior than their installed packages, the installed packages sometimes don't work, and package upgrades or new package installations are blocked.

After the Enable New Order Save Behavior release update is enabled, Salesforce evaluates and runs these customizations whenever an update to an order item record changes the parent order record.

- Order and order item validation rules
- Order and order item Apex triggers and classes
- Order and order item flows and processes

 **Note:** The New Order Save Behavior release update affects all package types: unlocked, unmanaged, first-generation managed package (1GP), and second-generation managed package (2GP).

You can install packages that support the Old Order Save Behavior on subscriber orgs that have enabled the New Order Save Behavior. However, you must verify that your package works with the new order save behavior.

After you verify that your package works with the new order save behavior and that all your packages associated with your Dev Hub org work with the new order save behavior, you can choose to enable the release update in your Dev Hub org. We recommend supporting both the new and old order save behavior.

#### Test Unmanaged and First-Generation Managed Packages

- From Setup, in the Quick Find box, enter *Release Updates*, and then select **Release Updates**. Locate the Enable New Order Save Behavior tile, and select **Enable Test Run**.
- Test the impact of the new behavior when an order or order item is edited. Review any custom application logic such as validation rules, Apex triggers and classes, workflow rules, flows, and processes.

We recommend supporting both the new and old order save behavior during the Release Update window.

- To indicate that your package is compatible with both new and old order save conditions, from Setup, in the Quick Find box, enter *Package*. Select the package that you tested and select **Upload**.
- Locate the Package Requirements section and disable **New Order Save Behavior**.

When this setting is disabled and the release update is enabled, subscriber orgs using either the new or old order save behavior can install your package.

#### Test Unlocked and Second-Generation Managed Packages

- After creating a scratch org, enable the Release Update in it. From Setup, in the Quick Find box, enter *Release Updates*, and then select **Release Updates**. Locate the Enable New Order Save Behavior tile, and select **Enable Test Run**.
- Test the impact of the new behavior when an order or order item is edited. Review any custom application logic such as validation rules, Apex triggers and classes, workflow rules, flows, and processes.

When you're ready to create a package version, specify the order save behavior in the definition file.

**Table 2: Order Save Behavior Options**

To Specify	Set Features in Scratch Org Definition File To
Old Order Save Behavior	<pre data-bbox="824 1535 1445 1808"> {   "features": [],   "settings": {     "orderSettings": {       "enableOrders": true     }   } }</pre>

To Specify	Set Features in Scratch Org Definition File To
New Order Save Behavior	<pre data-bbox="824 277 1435 525"> {   "features": ["OrderSaveLogicEnabled"],   "settings": {     "orderSettings": {       "enableOrders": true     }   } }</pre>
New and Old Order Save Behavior	<pre data-bbox="824 588 1435 835"> {   "features": ["OrderSaveBehaviorBoth"],   "settings": {     "orderSettings": {       "enableOrders": true     }   } }</pre>

## Develop Second-Generation Managed Packages

---

Ready to get started? Create your first second-generation managed package, and then update and create new versions of your package.

### [Create a Second-Generation Managed Package](#)

A package is a top-level container that holds important details about the app or package: the package name, description, and associated namespace. When you're ready to test or share your package, use the `sf package create` Salesforce CLI command to create a package.

### [View Package Details for a Second-Generation Managed Package](#)

View the details of previously created second-generation managed packages from the command line.

### [Create Versions of a Second-Generation Managed Package](#)

A package version is a fixed snapshot of the package contents and related metadata. The package version is an installable, immutable artifact that lets you manage changes and track what's different each time you release or deploy a specific set of changes.

### [Guidance for Package Version Numbering](#)

Use package versions to evolve your managed package, and release subsequent package versions without breaking existing package users. Every package version is a fixed snapshot of the package contents and related metadata.

### [View Details about a Second-Generation Managed Package Version](#)

Retrieve details about second-generation managed package versions that are in progress, or have already been created.

### [Project Configuration File for a Second-Generation Managed Package](#)

The project configuration file is a blueprint for your project. The settings in the file create an outline of your managed 2GP package and determine the package attributes and package contents.

### [Get Ready to Promote and Release a Second-Generation Managed Package Version](#)

By now it's likely that you've already created many different versions of your managed 2GP package and tested them. When you have a package version that you're ready to distribute, promoting the package version is the next step.

### [Specify a Package Ancestor in the Project File for a Second-Generation Managed Package](#)

When you create a second-generation managed package version you specify a package ancestor in your `sfdx-project.json` file. We require that the package ancestor you specify is the highest promoted package version number for that package. You can either update the ancestor version number each time you create a package version, or you can use a keyword.

## Create a Second-Generation Managed Package

A package is a top-level container that holds important details about the app or package: the package name, description, and associated namespace. When you're ready to test or share your package, use the `sf package create` Salesforce CLI command to create a package.

To create a package, change to the project directory in the CLI. The package name you enter becomes the package alias, and is automatically added to the project file. You can choose to designate an active Dev Hub org user to receive email notifications for Apex gacks, and install, upgrade, or uninstall failures associated with your packages. For definitions of each parameter shown here, see [sf package create](#) in the Salesforce CLI Reference Guide.

```
sf package create --name "Expenser App" --package-type Managed \
--path "expenser-main" --target-dev-hub my-hub --error-notification-username \
me@devhub.org
```

The package details you supply when you create a package are automatically added to your `sfdx-project.json` [project configuration file](#).

## Package Size Limit

A second-generation managed package can include up to 10,000 metadata files.

## Update Details about a Package

To update the name or description of an existing package, use this command.

```
sf package update --package "Expense App" --name "Expense Manager App" \
--description "The Winter '21 release is packed with an exciting set of features." \
--error-notification-username me2@devhub.org
```

 **Note:** You can't change the package namespace or package type after you create the package.

## View Package Details for a Second-Generation Managed Package

View the details of previously created second-generation managed packages from the command line.

To display a list of all packages in the Dev Hub org, use this command.

```
sf package list --target-dev-hub my-hub
```

You can view the namespace, package name, ID, and other details in the output.

Namespace	Prefix	Name	Id	Alias	Description	Type
db_exp_manager		Expenser App	0HoB00000004CzRKAU	Expenser App		Managed
db_exp_manager		Expenser Logic	0HoB00000004CzMKAU	Expenser Logic		Managed
db_exp_manager		Expenser Schema	0HoB00000004CzHKAU	Expenser Schema		Managed



Include optional parameters to filter the list results based on the modification date, creation date, and to order by specific fields or package IDs. To limit the details, use `--concise`.

To show expanded details, use `--verbose`. The verbose parameter displays these additional details.

- Created By
- Error Notification Username
- Subscriber Package ID

## Create Versions of a Second-Generation Managed Package

A package version is a fixed snapshot of the package contents and related metadata. The package version is an installable, immutable artifact that lets you manage changes and track what's different each time you release or deploy a specific set of changes.

Before you create a package version, first verify package details, such as the package name, dependencies, and update the `versionNumber` parameter in the `sfdx-project.json` file. Verify that the metadata you want to change or add in the new package version is in the package's main directory.


When you create a package version, you have three options regarding how package validations are handled.

- (Default) Complete all validations of dependencies, package ancestors, and metadata before the package version is returned.
- Perform validations asynchronously.
- Skip validation on the package version.

### Create a Managed 2GP Package Version (Default Option)

Create the package version with this command. Specify the package alias or ID (0Ho). You can also include a scratch definition file that contains a list of features and settings that the metadata of the package version depends on.

```
sf package version create --package "Expenser App" --installation-key "HIF83kS8kS7C" \
--definition-file config/project-scratch-def.json --code-coverage --wait 10
```

 **Note:** When creating a package version, specify a `--wait` time to run the command. If the package version is created within that time, the `sfdx-project.json` file is automatically updated with the package version information. If not, you must manually edit the project file.

It can be a long-running process to create a package version, depending on the package size and other variables. You can easily view the status and monitor progress.

```
sf package version create report --package-create-request-id 08cxx0000000YDAAY
```

The output shows details about the request.

```
=== Package Version Create Request
NAME                               VALUE
-----                               -----
Version Create Request Id         08cB00000004CBxIAM
Status                            InProgress
Package Id                        0HoB00000004C9hKAE
Package Version Id                05iB0000000CaaNIAS
Subscriber Package Version Id     04tB0000000NOimIAG
Tag                               git commit id 08dcfsdf
Branch
CreatedDate                       2024-05-08 09:48
```

```
Installation URL
https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB00000000NoimIAG
```

You can find the request ID (08c) in the initial output of `sf package version create`.

Depending on the size of the package and other variables, the create request can take several minutes. When you have more than one pending request to create package versions, you can view a list of all requests with this command.

```
sf package version create list --created-last-days 0
```

Details for each request display as shown here (IDs and labels truncated).

```
=== Package Version Create Requests [3]
ID      STATUS  PACKAGE2 ID PKG2 VERSION ID SUB PKG2 VER ID TAG BRANCH CREATED DATE ===
08c... Error   0Ho...
08c... Success 0Ho... 05i... 04t...                2024-06-22 12:07
08c... Success 0Ho... 05i... 04t...                2024-06-23 14:55
```

## Async Validation

Async validation creates a new package version before completing package validations. If your development team is using continuous integration (CI) scripts, you can leverage async validation to get an installable artifact sooner so you can start post-package creation steps.

To specify async validation, include the `--async-validation` parameter.

```
sf package version create --async-validation <rest of command syntax>
```

Sample Command-Line Output


```
Version create.... Create version status: PerformingValidations
The validations for this package version are in progress, but you can now begin testing
this package version.
To determine whether all package validations complete successfully, run "sf package version
create report --package-create-request-id 08cxx", and review the Status.
Async validated package versions can be promoted only if all validations complete
successfully.
Successfully created the package version [08cxx. Subscriber Package Version Id: 04txx
Package Installation URL:
https://login.salesforce.com/packaging/installPackage.apexp?p0=04txx
As an alternative, you can use the "sf package:install" command.
```

The command-line output provides you a package creation request ID that starts with 08c. To confirm whether all package validations complete successfully, use the 08cxx ID when and run `sf package version create report --package-create-request-id 08cxx`. Then validate that the `Status` is listed as `Success`. Async validated package versions can be promoted only if all validations complete successfully.

## Skip Validation

Skips validation of dependencies, package ancestors, and metadata during package version creation. Skipping validation significantly reduces the time it takes to create a new package version, but package versions created using skip validation can't be promoted to the released state.

```
sf package version create --skip-validation <rest of command syntax>
```

 **Note:** You can't specify both skip validation and code coverage, because code coverage is calculated during validation. You also can't specify both skip validation and async validation at the same time.

## Update Details about a Managed 2GP Package Version

You can update most properties of a package version from the command line. For example, you can change the package version name or description. One important exception is that you can't change the release status.

In this example, we're adding the tag parameter and specifying the git commit ID associated with this package version.

```
sf package version update --package "Expenser App@1.3.0-5" --tag "git commit id 08dcfsdf"
```

After the update is complete, you'll see output that looks like

```
Successfully updated the package version. 04tB0000000KPhnIAG
```

## How Many Managed 2GP Package Versions Can I Create Per Day?

Run this command to see how many package versions you can create per day and how many you have remaining.

```
sf limits api display
```

Look for the `Package2VersionCreates` entry.

NAME	REMAINING	MAXIMUM
Package2VersionCreates	23	50

## Guidance for Package Version Numbering

Use package versions to evolve your managed package, and release subsequent package versions without breaking existing package users. Every package version is a fixed snapshot of the package contents and related metadata.

While the format for package version number is predetermined, how you determine a version number, and whether you enforce uniqueness on package version numbers is left to package developers. The format for package version numbers is MAJOR.MINOR.PATCH.BUILD. Every package version has both a version number that you determine (for example, 2.2.0.1), and a unique [subscriber package version ID](#) (starts with 04t) that is auto-assigned when you create the package version.

Before you promote a particular MAJOR.MINOR.PATCH package version, it's possible to create multiple package versions that have unique 04t IDs, but all share the same version number, for example 2.2.0.1. There are a few approaches you can take to ensure each package version number is unique. Keep reading to learn more, but let's start by learning how to specify a package version number.

## How Do I Specify the Package Version Number?

The `versionNumber` attribute in your `sfdx-project.json` [project configuration file](#) determines the version number that is assigned the next time you create a managed 2GP version. Before creating a new package version, you must manually increment this attribute in the project file. If you don't increment the `versionNumber`, then you can wind up with multiple package versions with the same version number, but unique subscriber package version IDs (starts with 04t).

```
{
  "namespace": "exp-mgr",
  "sfdcLoginUrl": "https://login.salesforce.com",
```

```
"sourceApiVersion": "61.0",
"packageDirectories": [
  {
    "path": "util",
    "default": true,
    "package": "Expense Manager - Util",
    "versionName": "Summer '24",
    "versionDescription": "Summer 2024 Expense Manager Util Package",
    "versionNumber": "2.2.0.1",
    "definitionFile": "config/scratch-org-def.json"
  },
],
```

## Use the Keyword NEXT to Enforce Unique Build Numbers

As best practice, don't create multiple package versions that have the same MAJOR.MINOR.PATCH.BUILD version number. An easy way to ensure the build portion of your version number is unique is to use the keyword `NEXT` when you set the version number in your `sfdx-project.json` file. This way, you don't have to manually increment the version number when you want to create a new package version.

```
{
  "namespace": "exp-mgr",
  "sfdcLoginUrl": "https://login.salesforce.com",
  "sourceApiVersion": "61.0",
  "packageDirectories": [
    {
      "path": "util",
      "default": true,
      "package": "Expense Manager - Util",
      "versionName": "Summer '24",
      "versionDescription": "Summer 2024 Expense Manager Util Package",
      "versionNumber": "2.2.0.NEXT",
      "definitionFile": "config/scratch-org-def.json"
    },
  ],
}
```

## Use the CLI Flag to Override a Package Version Number

You can also override the version number listed in your project file, by using the `--version-number` flag when you create a new package version.

```
sf package version create -p "my2gp" --version-number 2.2.0.NEXT <rest of command syntax>
```

By using the keyword `NEXT` with the `--version-number` flag in the CLI, you ensure the build portion of the version number is unique.



**Note:** Keep in mind, the `--version-number` flag doesn't update your `sfdx-project.json`. To keep the `VersionNumber` in the project file current, update it manually.

## What Happens to Version Numbering After You Promote a Package Version?

After you promote a package version with a specific MAJOR.MINOR.PATCH version you can't continue to create package versions that use that same MAJOR.MINOR.PATCH version number. If you attempt to do so, you receive an error message.

## How Do I Determine Whether to Use a New Major, Minor, or Patch Version?

While there are restrictions on what changes are allowed in a [patch version](#), determining what qualifies as a major or minor change is largely up to you. When introducing major changes, increase the major version number, and increase the minor version number when making smaller improvements.

## View Details about a Second-Generation Managed Package Version

Retrieve details about second-generation managed package versions that are in progress, or have already been created.

### View Status and Progress Details for a Managed 2GP Package Version

Depending on the package size and other variables, creating a package version can be a long-running process. You can easily view the status and monitor progress using this report command.

```
sf package version create report --package-create-request-id 08cxx00000000YDAAY
```

The output shows details about the request.

```
=== Package Version Create Request
NAME                               VALUE
-----                               -----
Version Create Request Id         08cB00000004CBxIAM
Status                             InProgress
Package Id                         0HoB00000004C9hKAE
Package Version Id                 05iB0000000CaaNIAS
Subscriber Package Version Id      04tB0000000NOimIAG
Tag                                 git commit id 08dcfsdf
Branch
CreatedDate                        2018-05-08 09:48
Installation URL
https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB0000000NOimIAG
```

You can find the request ID (08c) in the initial output of `sf package version create`.

If you have more than one pending request to create package versions, you can view a list of all requests with this command.

```
sf package version create list --created-last-days 0
```

Details for each request display as shown here (IDs and labels truncated).

```
=== Package Version Create Requests [3]
ID      STATUS  PACKAGE2 ID PKG2 VERSION ID SUB PKG2 VER ID TAG BRANCH CREATED DATE ===
08c...  Error   0Ho...
08c...  Success 0Ho... 05i... 04t...                2022-06-22 12:07
08c...  Success 0Ho... 05i... 04t...                2022-06-23 14:55
```

## Retrieve List of all Package Versions Associated with a Dev Hub Org

To display a list of all package versions in the Dev Hub org, use this command.

```
sf package version list --target-dev-hub my-hub
```

You can view the namespace, version name, and other details in the output.

Package Name Installation Key	Namespace Released	Version	Sub Pkg Ver Id	Alias
Expenser Schema false	db_exp_manager true	0.1.0.1	04tB0000000719qIAA	Expenser Schema@0.1.0-1
Expenser Schema false	db_exp_manager true	0.2.0.1	04tB000000071AjIAI	Expenser Schema@0.2.0-1
Expenser Schema false	db_exp_manager false	0.3.0.1	04tB000000071AtIAI	Expenser Schema@0.3.0-1
Expenser Schema false	db_exp_manager true	0.3.0.2	04tB000000071AyIAI	Expenser Schema@0.3.0-2
Expenser Schema false	db_exp_manager false	0.3.1.1	04tB0000000KGU6IAO	Expenser Schema@0.3.1-1
Expenser Schema false	db_exp_manager true	0.3.1.2	04tB0000000KGUBIA4	Expenser Schema@0.3.1-2
Expenser Schema false	db_exp_manager true	0.3.2.1	04tB0000000KGUQIA4	Expenser Schema@0.3.2-1
Expenser Logic false	db_exp_manager true	0.1.0.1	04tB0000000719vIAA	Expenser Logic@0.1.0-1
Expenser App false	db_exp_manager true	0.1.0.1	04tB000000071A0IAI	Expenser App@0.1.0-1

To view details about a specific package, include `--package` parameter when you run `sf package version list`.

To show expanded details, use `--verbose`. The verbose parameter displays these additional details.

- Ancestor
- Ancestor Version
- Branch
- Build Duration in Seconds
- Code Coverage
- Code Coverage Met
- Created By
- Created Date
- Description
- Installation URL
- Language
- Managed Metadata Removed
- Package ID
- Package Version ID
- Release Version
- Tag
- Validation Skipped
- WasTransferred

## Project Configuration File for a Second-Generation Managed Package

The project configuration file is a blueprint for your project. The settings in the file create an outline of your managed 2GP package and determine the package attributes and package contents.

Here are some of the parameters you can specify in the project configuration file. For additional parameters, see [Advanced Project Configuration Parameters for Second-Generation Managed Packages](#).

Name	Details
ancestorId	<p><b>Required?</b> It depends on whether you've already promoted a package version of this package. If yes, you must specify either the ancestorId or ancestorVersion. If no, this parameter isn't required.</p> <p><b>Default if Not Specified:</b> None</p> <p>None. The ID of the immediate parent in the package ancestry tree of the package version you're creating. The ancestorId requires the 04t of the package version, or an alias to the package version. When specifying ancestors, you can use either ancestorId or ancestorVersion.</p> <p>Example:</p> <pre data-bbox="824 894 1450 940">"ancestorId": "Expenser Logic@0.1.0-1"</pre> <p>For more information, see <a href="#">Specify a Package Ancestor in the Project File for a Second-Generation Managed Package</a>.</p>
ancestorVersion	<p><b>Required?</b> It depends on whether you've already promoted a package version of this package. If yes, you must specify either the ancestorId or ancestorVersion. If no, this parameter isn't required.</p> <p><b>Default if Not Specified:</b> None</p> <p>The version number of the immediate parent in the package ancestry tree of the package version you're creating.</p> <p>Specify the ancestor version using the format of major.minor.patch.build. When specifying ancestors, you can use either ancestorId or ancestorVersion.</p> <p>Example:</p> <pre data-bbox="824 1461 1450 1507">"ancestorVersion": "0.1.0.1"</pre> <p>For more information, see <a href="#">Specify a Package Ancestor in the Project File for a Second-Generation Managed Package</a>.</p>
default	<p><b>Required?</b> Yes, if you've specified more than one package directory</p> <p><b>Default if Not Specified:</b> true</p> <p>Indicates the default package directory. When metadata is retrieved from a development org (scratch org or source-tracked sandbox) using <code>sf project retrieve</code>, it's placed in the default package directory.</p>

Name	Details
	<p>There can be only one package directory in which the default is set to true.</p>
<p>definitionFile</p>	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p> <p>A reference to an external <code>.json</code> file used to specify the features and org settings required for the metadata of your package, such as the scratch org definition.</p> <p>Example:</p> <pre data-bbox="824 615 1448 688">"definitionFile": "config/project-scratch-def.json",</pre>
<p>namespace</p>	<p><b>Required?</b> Yes</p> <p><b>Default if Not Specified:</b> None</p> <p>A 1–15 character alphanumeric identifier that distinguishes your package and its contents from packages of other developers.</p>
<p>package</p>	<p><b>Required?</b> Yes</p> <p><b>Default if Not Specified:</b> None</p> <p>The package name is specified in the project json file.</p>
<p>packageAliases</p>	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> Salesforce CLI updates this file with the aliases when you create a package or package version. You can also manually update this section for existing packages or package versions. You can use the alias instead of the cryptic package ID when running CLI <code>sf package</code> commands.</p>
<p>path</p>	<p><b>Required?</b> Yes</p> <p><b>Default if Not Specified:</b> None.</p> <p>Specify the location that contains the package metadata in the <code>--path</code> attribute of <code>sf package create</code> Salesforce CLI command.</p>
<p>seedMetadata</p>	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None.</p> <p>Specify the path to your seedMetadata directory.</p> <p>Seed metadata is available to standard value sets only. If your package depends on standard value sets, you can specify a seed metadata directory that contains the value sets.</p>



Name	Details
	<p>Example:</p> <pre data-bbox="824 304 1442 569"> "packageDirectories": [   {     "seedMetadata": {       "path": "my-unpackaged-seed-directory"     }   }, ] </pre>
versionDescription	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p>
versionName	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> If not specified, the CLI uses <code>versionNumber</code> as the version name.</p>
versionNumber	<p><b>Required?</b> Yes</p> <p><b>Default if Not Specified:</b> None</p> <p>The <code>versionNumber</code> field sets the version number that is assigned the next time you create a 2GP version. Version numbers are formatted as MAJOR.MINOR.PATCH.BUILD. For example, 1.2.1.8. To avoid creating multiple package versions with the same MAJOR.MINOR.PATCH.BUILD number, you must increment the <code>versionNumber</code> before creating a new package version.</p> <p>To automatically increment the build number to the next available build for the package, use the keyword NEXT (1.2.1.NEXT).</p> <p>Alternatively, when you create a new package version, you can set the version number using the <code>--versionNumber</code> flag in the CLI.</p> <p>For more details, see <a href="#">Guidance for Version Numbering</a>.</p>

When you specify a parameter using Salesforce CLI, it overrides the value listed in the project definition file.

The Salesforce DX project definition file is a JSON file located in the root directory of your project. Use the `sf project generate` CLI command to generate a project file that you can build upon. Here's how the parameters in `packageDirectories` appear.

```

{
  "namespace": "exp-mgr",
  "sfdcLoginUrl": "https://login.salesforce.com",
  "sourceApiVersion": "61.0",
  "packageDirectories": [
    {
      "path": "util",
      "default": true,

```

```

    "package": "Expense Manager - Util",
    "versionName": "Summer '24",
    "versionDescription": "Welcome to Summer 2024 Release of Expense Manager Util
Package",
    "versionNumber": "4.7.0.NEXT",
    "definitionFile": "config/scratch-org-def.json"
  },
  {
    "path": "exp-core",
    "default": false,
    "package": "Expense Manager",
    "versionName": "v 3.2",
    "versionDescription": "Summer 2024 Release",
    "versionNumber": "3.2.0.NEXT",
    "ancestorVersion": "3.0.0.7",
    "definitionFile": "config/scratch-org-def.json",
    "dependencies": [
      {
        "package": "Expense Manager - Util",
        "versionNumber": "4.7.0.LATEST"
      },
      {
        "package": "External Apex Library - 1.0.0.4"
      }
    ]
  }
],
"packageAliases": {
  "Expense Manager - Util": "0HoB00000004CFpKAM",
  "External Apex Library@1.0.0.4": "04tB0000000IB1EIAW",
  "Expense Manager": "0HoB00000004CFuKAM"
}

```

## What If I Don't Want My Salesforce DX Project Automatically Updated?

In some circumstances, you don't want to have automatic updates to the `sfdx-project.json` file. When you require more control, use these environment variables to suppress automatic updates to the project file.

For This Command	Set This Environment Variable to True
<code>sf package create</code>	<code>SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_CREATE</code>
<code>sf package version create</code>	<code>SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_VERSION_CREATE</code>

SEE ALSO:

[Advanced Project Configuration Parameters for Second-Generation Managed Packages](#)

## Get Ready to Promote and Release a Second-Generation Managed Package Version

By now it's likely that you've already created many different versions of your managed 2GP package and tested them. When you have a package version that you're ready to distribute, promoting the package version is the next step.

Each package version you create is a beta version, unless you promote it to the managed-released state. Beta versions can be installed in only scratch orgs and sandboxes. After you install a beta version into an org, you can't later upgrade that installed beta version. Keep this in mind when you select which org to install and test your beta package version. If you use this sandbox as part of your release pipeline, then using a disposable scratch org is a better option to test your beta package.

A beta package version must pass a 75% code coverage requirement before it can be promoted. To learn more, see [Code Coverage for Second-Generation Managed Packages](#).

To promote a package version to the released state, run the `sf package version promote` Salesforce CLI command. For step-by-step instructions on promoting a package version, see [Release a Second Generation Managed Package](#).

After a package version is promoted, you can install it in either a production org or development orgs, and can be distributed to your customers.

For every minor package version, you can promote only one beta version. For example, if you create several beta versions of package version 2.3, only one of those versions can be promoted. After promoting package version 2.3, start your new development using version number 2.4.

After a package version is promoted to the released state, you can't reverse the promotion. If you discover you don't want to distribute a version you promoted, you can't reverse that version back to the beta state. To ensure that that version isn't inadvertently shared and installed in a customer org, we recommend you use the `sf package version update` Salesforce CLI command and set the installation key to something cryptic and difficult to guess.

### SEE ALSO:

[Considerations for Promoting Packages with Dependencies](#)

[Release a Second-Generation Managed Package](#)

[Code Coverage for Second-Generation Managed Packages](#)

## Specify a Package Ancestor in the Project File for a Second-Generation Managed Package

When you create a second-generation managed package version you specify a package ancestor in your `sfdx-project.json` file. We require that the package ancestor you specify is the highest promoted package version number for that package. You can either update the ancestor version number each time you create a package version, or you can use a keyword.

Here are three different ways to set the package ancestor.

### Use the HIGHEST Keyword (Recommended)

Use the keyword `HIGHEST` with either the `ancestorId` or `ancestorVersion` attribute in the `sfdx-project.json` file. This keyword automatically sets the ancestor to the highest promoted package version number.

```
"packageDirectories": [  
  {  
    "path": "util",  
    "package": "Expense Manager - Util",
```

```
"versionNumber": "4.7.0.NEXT",
"ancestorVersion": "HIGHEST"
},
```

This keyword makes it easy to set your package ancestor to use linear versioning, until you have a reason to break from linear versioning.

## Use the Ancestor Version Attribute

Set the `ancestorVersion` attribute in the `sfdx-project.json` file to the package version's major.minor.patch number. This approach requires you to update the ancestor version number every time the major, minor, or patch value changes.

```
"packageDirectories": [
{
"path": "util",
"package": "Expense Manager - Util",
"versionNumber": "4.7.0.NEXT",
"ancestorVersion": "4.6.0"
},
```

## Use the Ancestor ID Attribute

Set the `ancestorId` attribute in the `sfdx-project.json` file to either the 04t ID or the package version's alias. This approach requires you to update the ancestor version number every time you create a package version.

```
"packageDirectories": [
{
"path": "util",
"package": "Expense Manager - Util",
"versionNumber": "4.7.0.NEXT",
"ancestorId": "04tB0000000cWwnIAE"
},
```

```
"packageDirectories": [
{
"path": "util",
"package": "Expense Manager - Util",
"versionNumber": "4.7.0.NEXT",
"ancestorId": "expense-manager@4.6.0.1"
},
```

 **Note:** Only package versions that have been promoted to managed-released state, can be listed as an ancestor.

## Override Linear Package Ancestry Behavior

To break from linear package versioning, specify a package ancestor that isn't the highest promoted package version and use the Salesforce CLI parameter `--skip-ancestor-check` when you create a package version.

```
sf package version create --package "Expenser App" --skip-ancestor-check
```

The CLI parameter indicates that you're intentionally choosing to specify a package version that isn't the highest promoted package version.

You can choose to not specify a package ancestor by using the keyword, `NONE`, with either the `ancestorId` or `ancestorVersion` attribute in the `sfdx-project.json` file.

```
"packageDirectories": [  
  {  
    "path": "util",  
    "package": "Expense Manager - Util",  
    "versionNumber": "4.7.0.NEXT",  
    "ancestorVersion": "NONE"  
  },  
]
```

Because package ancestors determine package upgrade paths, existing customers can't upgrade to a package version that is created without a specified ancestor. Use `NONE` if you don't plan to promote the package version you're creating.

If you've already promoted a previous package version, and you set the ancestor to `NONE` on a new package version associated with the same package, include `--skip-ancestor-check` when you create that package version. When you create your first package version, you can also set the ancestor to `NONE` and specify `--skip-ancestor-check`.

## What to Remember about Package Ancestry

- Package ancestry determines whether existing packages can be upgraded to newer package versions. If you're breaking from linear versioning, or plan to abandon a package version that is installed in customer orgs, consider how your existing customers will be impacted, and whether an upgrade path is available to them.
- If you abandon a package version, delete the version using the Salesforce CLI command `sf package version delete`.  
If you aren't able to delete the package version, then update the package version's installation key so the abandoned package version can't be inadvertently installed. Use `sf package version update` to update the installation key.

## Install and Uninstall Second-Generation Managed Packages

---

Use a disposable scratch org to test your second-generation managed packages (managed 2GP). You can install or uninstall a managed 2GP package using a Salesforce CLI command, or from the Setup page. Because you can't upgrade a beta package version, be sure you don't install it in a sandbox that you use in your release pipeline, such as UAT or staging.

### [Use the CLI to Install a Second-Generation Managed Package](#)

If you're working with the Salesforce CLI, you can use the `sf package install` command to install packages in a scratch org or target subscriber org.

### [Use a URL to Install a Second-Generation Managed Package](#)

Install a second-generation managed package from a browser.

### [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.

### [Upgrade a Second-Generation Managed Package Version](#)

A package upgrade occurs when you install a new package version into an org that has a previous version of that package installed.

### [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.

[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.

[Customize Second-Generation Managed Package Installs and Uninstalls Using Scripts](#)

Customize a second-generation managed package (managed 2GP) install or upgrade by specifying an Apex post install script to run automatically after a subscriber installs or upgrades a managed 2GP package. You can also specify an Apex uninstall script to run automatically when a subscriber uninstalls a managed 2GP package.

[Sample Script for Installing Second-Generation Managed Packages with Dependencies](#)

Use this sample script as a basis to create your own script to install second-generation managed packages with dependencies. This script contains a query that finds dependent packages and installs them in the correct dependency order.

[Uninstall a Second-Generation Managed Package](#)

You can uninstall a second-generation managed package from an org using Salesforce CLI or from the Setup UI. When you uninstall second-generation managed packages, all components in the package, including any deprecated components that were previously associated with the package, are deleted from the org.

## Use the CLI to Install a Second-Generation Managed Package

If you're working with the Salesforce CLI, you can use the `sf package install` command to install packages in a scratch org or target subscriber org.

Before you install a second-generation managed package (managed 2GP) in a scratch org, run this command to list all the packages and locate the ID or package alias.

```
sf package version list
```


Identify the version you want to install. Enter this command, supplying the package alias or package ID (starts with 04t).

```
sf package install --package "Expense Manager@1.2.0-12" --target-org jdoe@example.com
```

By default, the package install command provides admins access to the installed package. To provide access to all users, specify `--security-type AllUsers` when you run the package install command.

If you've already set the scratch org with a default username, enter just the package version ID.

```
sf package install --package "Expense Manager@1.2.0-12"
```

 **Note:** If you've defined an alias (with the `-a` parameter), you can specify the alias instead of the username for `--target-org`.

The CLI displays status messages regarding the installation.

```
Waiting for the subscriber package version install request to get processed. Status =
InProgress Successfully installed the subscriber package version: 04txx0000000FIuAAM.
```

## Control Managed 2GP Package Installation Timeouts

When you issue a `sf package install` command, it takes a few minutes for a package version to become available in the target org and for installation to complete. To allow sufficient time for a successful install, use these parameters that represent mutually exclusive timers.

- `--publish-wait` defines the maximum number of minutes that the command waits for the package version to be available in the target org. The default is 0. If the package is not available in the target org in this time frame, the install is terminated.

Setting `--publish-wait` is useful when you create a new package version and then immediately try to install it to target orgs.



**Note:** If `--publish-wait` is set to 0, the package installation immediately fails, unless the package version is already available in the target org.

- `--wait` defines the maximum number of minutes that the command waits for the installation to complete after the package is available. The default is 0. When the `--wait` interval ends, the install command completes, but the installation continues until it either fails or succeeds. You can poll the status of the installation using `sf package install report`.



**Note:** The `--wait` timer takes effect after the time specified by `--publish-wait` has elapsed. If the `--publish-wait` interval times out before the package is available in the target org, the `--wait` interval never starts.

For example, consider a package called Expense Manager that takes five minutes to become available on the target org, and 11 minutes to install. The following command has `publish-wait` set to three minutes and `wait` set to 10 minutes. Because Expense Manager requires more time than the set `publish-wait` interval, the installation is aborted at the end of the three-minute `publish-wait` interval.

```
sf package install --package "Expense Manager@1.2.0-12" --publish-wait 3 --wait 10
```

The following command has `publish-wait` set to six minutes and `wait` set to 10 minutes. If not already available, Expense Manager takes five minutes to become available on the target org. The clock then starts ticking for the 10-minute `wait` time. At the end of 10 minutes, the command completes because the `wait` time interval has elapsed, although the installation is not yet complete. At this point, `sf package install report` indicates that the installation is in progress. After one more minute, the installation completes and `sf package install report` indicates a successful installation.

```
sf package install --package "Expense Manager@1.2.0-12" --publish-wait 6 --wait 10
```

SEE ALSO:

[Salesforce CLI Command Reference package install](#)

[Salesforce Help: Determine Which Users Can Access a Package](#)

## Use a URL to Install a Second-Generation Managed Package

Install a second-generation managed package from a browser.

If you create packages from the CLI, you can derive an installation URL for the package by adding the subscriber package ID to your Dev Hub URL. You can use this URL to test different deployment or installation scenarios.

For example, if the package version has the subscriber package ID, 04tB00000009oZ3JBI, add the ID as the value of `apvId`.

```
https://MyDomainName.lightning.force.com/packagingSetupUI/ipLanding.app?apvId=04tB00000009oZ3JBI
```

Anyone with the URL and a valid login to a Salesforce org can install the package.

To install the package:

1. In a browser, enter the installation URL.
2. Enter your username and password for the Salesforce org in which you want to install the package, and then click **Login**.
3. If the package is protected by an installation key, enter the installation key.
4. For a default installation, click **Install**.

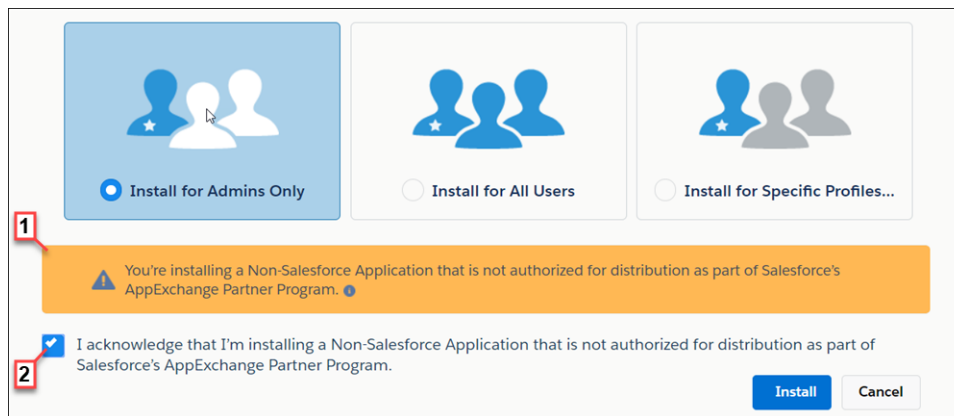
A message describes the progress. You receive a confirmation message when the installation is complete.

SEE ALSO:

[Salesforce Help: Determine Which Users Can Access a Package](#)

## 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.



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](#).

## Upgrade a Second-Generation Managed Package Version

A package upgrade occurs when you install a new package version into an org that has a previous version of that package installed.

When you perform a package upgrade, here's what to expect for metadata changes.

- Metadata introduced in the new version is installed as part of the upgrade.
- Metadata modified in the new version is updated as part of the upgrade.
- Metadata removed in the new version is either deprecated or deleted as part of the upgrade.

To upgrade a package, use the package install CLI command

```
sf package install --package 04t... --target-org me@example.com
```



For more examples and details about this command, see [package install](#) in the *Salesforce CLI Command Reference*.

Beta packages aren't upgradeable. To install a new beta package or released version, first uninstall the beta package.

To upgrade a package version, the new version must be a direct descendent of the package version installed in your org. See [Specify a Package Ancestor in the Project File for a Second-Generation Managed Package](#) for more information.

SEE ALSO:

[Salesforce CLI Command Reference package install](#)

## 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` or `inherit sharing` keyword. These keywords 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
<code>compareTo</code>	<code>System.Version version</code>	Integer	Compares the current version with the specified version and returns one of the following values: <ul style="list-style-type: none"> <li>• Zero if the current package version is equal to the specified package version</li> <li>• An Integer value greater than zero if the current package version is greater than the specified package version</li> </ul>

Method	Arguments	Return Type	Description
			<ul style="list-style-type: none"> <li>An Integer value less than zero if the current package version is less than the specified package version</li> </ul> <p>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.</p>
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 <code>null</code> 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.

SEE ALSO:

[Customize Second-Generation Managed Package Installs and Uninstalls Using Scripts](#)

## Customize Second-Generation Managed Package Installs and Uninstalls Using Scripts

Customize a second-generation managed package (managed 2GP) install or upgrade by specifying an Apex post install script to run automatically after a subscriber installs or upgrades a managed 2GP package. You can also specify an Apex uninstall script to run automatically when a subscriber uninstalls a managed 2GP package.

For more information, see [Run Apex on Package Install/Upgrade](#) and [Run Apex on Package Uninstall](#).

Specify post install and uninstall scripts in the `sfdx-project.json` file.

```
"packageDirectories": [
  {
    "path": "expenser-schema",
    "default": true,
    "package": "Expense Schema",
    "versionName": "ver 0.3.2",
    "versionNumber": "0.3.2.NEXT",
    "postInstallScript": "PostInstallScript",
    "uninstallScript": "UninstallScript",
    "postInstallUrl": "https://expenser.com/post-install-instructions.html",
    "releaseNotesUrl": "https://expenser.com/winter-2020-release-notes.html"
  },
],
{
  "namespace": "db_exp_manager",
  "sfdcLoginUrl": "https://login.salesforce.com",
  "sourceApiVersion": "47.0",
  "packageAliases": {
    "Expenser Schema": "0HoB00000004CzHKAU",
    "Expenser Schema@0.1.0-1": "04tB0000000719qIAA"
  }
}
```

You can also use the `--post-install-script` and the `--uninstall-script` Salesforce CLI parameters with the `sf package version create` command. The CLI parameters override the scripts specified in the `sfdx-project.json` file.

 **Note:** Include the Apex classes for your post-install and uninstall scripts with the metadata in your package.

You can designate an active Dev Hub org user to receive email notifications for Apex gacks, and install, upgrade, or uninstall failures associated with your packages. In Salesforce CLI run `sf package create --error-notification-username me@devhub.org` or `sf package update --error-notification-username me@devhub.org`. In Tooling API, use the `PackageErrorUsername` field on the `Package2` object.

## Sample Script for Installing Second-Generation Managed Packages with Dependencies

Use this sample script as a basis to create your own script to install second-generation managed packages with dependencies. This script contains a query that finds dependent packages and installs them in the correct dependency order.

### Sample Script

 **Note:** Be sure to replace the package version ID and scratch org user name with your own specific details.

```
#!/bin/bash

# The execution of this script stops if a command or pipeline has an error.
# For example, failure to install a dependent package will cause the script
# to stop execution.

set -e

# Specify a package version id (starts with 04t)
# If you know the package alias but not the id, use sf package version list to find it.
PACKAGE=04tB0000000NmnHIAS

# Specify the user name of the subscriber org.
USER_NAME=test-bvdfz3m9tqdf@example.com

# Specify the timeout in minutes for package installation.
WAIT_TIME=15

echo "Retrieving dependencies for package Id: "$PACKAGE

# Execute soql query to retrieve package dependencies in json format.
RESULT_JSON=`sf data query -u $USER_NAME -t -q "SELECT Dependencies FROM
SubscriberPackageVersion WHERE Id='$PACKAGE'" --json`

# Parse the json string using python to test whether the result json contains a list of
ids or not.
DEPENDENCIES=`echo $RESULT_JSON | python -c 'import sys, json; print`
```

```
json.load(sys.stdin) ["result"] ["records"] [0] ["Dependencies"] '`

# If the parsed dependencies is None, the package has no dependencies. Otherwise, parse
the result into a list of ids.

# Then loop through the ids to install each of the dependent packages.

if [[ "$DEPENDENCIES" != 'None' ]]; then

    DEPENDENCIES=`echo $RESULT_JSON | python -c '

import sys, json

ids = json.load(sys.stdin) ["result"] ["records"] [0] ["Dependencies"] ["ids"]

dependencies = []

for id in ids:

    dependencies.append(id["subscriberPackageVersionId"])

print " ".join(dependencies)

`,`

    echo "The package you are installing depends on these packages (in correct dependency
order): "$DEPENDENCIES

    for id in $DEPENDENCIES

    do

        echo "Installing dependent package: "$id

        sf package install --package $id -u $USER_NAME -w $WAIT_TIME --publish-wait 10

    done

else

    echo "The package has no dependencies"

fi

# After processing the dependencies, proceed to install the specified package.

echo "Installing package: "$PACKAGE
```

```
sf package install --package $PACKAGE -u $USER_NAME -w $WAIT_TIME --publish-wait 10

exit 0;
```

## Uninstall a Second-Generation Managed Package

You can uninstall a second-generation managed package from an org using Salesforce CLI or from the Setup UI. When you uninstall second-generation managed packages, all components in the package, including any deprecated components that were previously associated with the package, are deleted from the org.

To use the CLI to uninstall a package from the target org, authorize the Dev Hub org and run this command.

```
sf package uninstall --package "Expense Manager@2.3.0-5"
```

You can also uninstall a package from the web browser. Open the Salesforce org where you installed the package.

```
sf org open -u me@my.org
```

Then uninstall the package.

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**.

## Considerations on Uninstalling Packages

- 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, 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.

SEE ALSO:

[Salesforce CLI Command Reference package uninstall](#)

## Prepare to Distribute Your Second-Generation Managed Package

---

Before you release a version of your second-generation managed package, ensure that you understand the code coverage requirements, release logistics, and how to publish your app on AppExchange.

### [Code Coverage for Second-Generation Managed Packages](#)

Before you can release and distribute a second-generation managed package version on AppExchange, the Apex code must meet a minimum 75% code coverage requirement. And every Apex Trigger in a package needs test coverage.

### [Package Installation Key for Second-Generation Managed Packages](#)

To ensure the security of the metadata in your second-generation managed package, you must specify an installation key when creating a package version. Package creators provide the key to authorized subscribers so they can install the package. Package installers provide the key during installation, whether installing the package from the CLI or from a browser. An installation key is the first step during installation. The key ensures that no package information, such as the name or components, is disclosed until the correct installation key is supplied.

### [Release a Second-Generation Managed Package](#)

Each new second-generation managed package version is marked as beta when created. As you develop your package, you may create several package versions before you create a version that is ready to be released and distributed. Only released package versions can be listed on AppExchange and installed in customer orgs.

### [Share Release Notes and Post-Install Instructions for Second-Generation Managed Packages](#)

Share details with your subscribers about what's new and changed in a released second-generation managed package.

### [Publishing Your App on AppExchange](#)

If you've published a first-generation managed package, you'll notice the process for publishing a second-generation managed package (managed 2GP) is different. After you link your Dev Hub org to the AppExchange partner console, all your released managed 2GP package versions are visible in the partner console.

## Code Coverage for Second-Generation Managed Packages

Before you can release and distribute a second-generation managed package version on AppExchange, the Apex code must meet a minimum 75% code coverage requirement. And every Apex Trigger in a package needs test coverage.

To compute code coverage using Salesforce CLI, use the `--code-coverage` parameter when you run the `sf package version create` command.

Package version creation often takes longer to complete when code coverage is being computed, so consider when to include the code coverage parameter. You can create beta package versions without computing code coverage, but these beta versions can't be promoted.

If you try to promote a beta package version to managed-released and the version was created without specifying code coverage, or the code coverage in the package version is less than 75%, the package promotion fails. Code coverage is calculated during package version validation. If you skip validation using the `--skip-validation` parameter, code coverage isn't calculated for that package version.

View code coverage information for a package version using `sf package version list` with the `--verbose` parameter, or `sf package version report` command in Salesforce CLI.

## Package Installation Key for Second-Generation Managed Packages

To ensure the security of the metadata in your second-generation managed package, you must specify an installation key when creating a package version. Package creators provide the key to authorized subscribers so they can install the package. Package installers provide the key during installation, whether installing the package from the CLI or from a browser. An installation key is the first step during installation. The key ensures that no package information, such as the name or components, is disclosed until the correct installation key is supplied.

To set the installation key, add the `--installation-key` parameter to the command when you create the package version. This command creates a package and protects it with the installation key.

```
sf package version create --package "Expense Manager" --installation-key "JSB7s8vXU93fI"
```

Supply the installation key when you install the package version in the target org.

```
sf package install --package "Expense Manager" --installation-key "JSB7s8vXU93fI"
```

## Change the Installation Key for an Existing Package Version

You can change the installation key for an existing package version with the `sf package version update` command.

```
sf package version update --package "Expense Manager@1.2.0-4" --installation-key "HIF83ks8ks7C"
```

## Create a Package Version Without an Installation Key

If you don't require security measures to protect your package metadata, you can create a package version without an installation key.

```
sf package version create --package "Expense Manager" --installation-key-bypass
```

## Check Whether a Package Version Requires an Installation Key

To determine whether a package version requires an installation key, use the `sf package version list` CLI command.

## Release a Second-Generation Managed Package

Each new second-generation managed package version is marked as beta when created. As you develop your package, you may create several package versions before you create a version that is ready to be released and distributed. Only released package versions can be listed on AppExchange and installed in customer orgs.

Before you promote the package version, ensure that the user permission, **Promote a package version to released**, is enabled in the Dev Hub org associated with the package. Consider creating a permission set with this user permission, and then assign the permission set to the appropriate user profiles.

When you're ready to release, use `sf package version promote`.

```
sf package version promote --package "Expense Manager@1.3.0-7"
```

If the command is successful, a confirmation message appears.

```
Successfully promoted the package version, ID: 04tB0000000719qIAA to released.
```

After the update succeeds, view the package details.

```
sf package version report --package "Expense Manager@1.3.0.7"
```

Confirm that the value of the Released property is true.

```
=== Package Version
NAME                               VALUE
-----                               -----
Name                               ver 1.0
Alias                              Expense Manager-1.0.0.5
Package Version Id                 05iB0000000CaahIAC
Package Id                         0HoB0000000CabmKAC
Subscriber Package Version Id      04tB0000000NPbBIAW
Version                             1.0.0.5
Description                         update version
Branch
Tag                                 git commit id 08dcfsdf
Released                            true
Created Date                       2021-05-08 09:48
Installation URL
https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB0000000NPbBIAW
```

You can promote and release only one time for each package version number, and you can't undo this change.

## Share Release Notes and Post-Install Instructions for Second-Generation Managed Packages

Share details with your subscribers about what's new and changed in a released second-generation managed package.

You can specify a release notes URL to display on the package detail page in the subscriber's org. And you can share instructions about using your package by specifying a post install URL. The release notes and post install URLs display on the Installed Packages page in Setup, after a successful package installation. For subscribers who install packages using an installation URL, the package installer page displays a link to release notes. And subscribers are redirected to your post install URL following a successful package installation or upgrade.

Specify the `postInstallUrl` and `releaseNotesUrl` attributes in the `packageDirectories` section for the package.

```
"packageDirectories": [
  {
    "path": "expenser-schema",
    "default": true,
    "package": "Expense Schema",
    "versionName": "ver 0.3.2",
    "versionNumber": "0.3.2.NEXT",
    "postInstallScript": "PostInstallScript",
    "uninstallScript": "UninstallScript",
    "postInstallUrl": "https://expenser.com/post-install-instructions.html",
    "releaseNotesUrl": "https://expenser.com/winter-2020-release-notes.html"
  },
],
```

```
{
  "namespace": "db_exp_manager",
  "sfdcLoginUrl": "https://login.salesforce.com",
  "sourceApiVersion": "47.0",
  "packageAliases": {
    "Expenser Schema": "0HoB00000004CzHKAU",
    "Expenser Schema@0.1.0-1": "04tB0000000719qIAA"
  }
}
```

You can also use the `--post-install-url` and the `--release-notes-url` Salesforce CLI parameters with the `sf package version create` command. The CLI parameters override the URLs specified in the `sfdx-project.json` file.

## Publishing Your App on AppExchange

If you've published a first-generation managed package, you'll notice the process for publishing a second-generation managed package (managed 2GP) is different. After you link your Dev Hub org to the AppExchange partner console, all your released managed 2GP package versions are visible in the partner console.

To list an app on AppExchange, it must pass the AppExchange security review. For more information, see [Pass the AppExchange Security Review](#) in the ISVforce Guide.

## Link Dev Hub to the AppExchange Partner Console

- Log in to the [Salesforce Partner Community](#).
- Select the **Publishing** tab
- Click **Technologies**
- Click **Org**
- Click **Connect Technology**, and **Org**
- Click **Connect Org** and **Allow**, and enter the login credentials for your Dev Hub org.

## Register Your Managed 2GP Package

- From the Solutions tab, locate the package version you want to register, and click **Register Package**. Registering a package links the package to your [license management app](#).
- Enter the login credentials for the Dev Hub org associated with the package in the modal window.
- Set the default license behavior for the package, including trial length, and number of seats included with the license, and click **Save**.

Packages that share a namespace can be associated with the same License Management Org (LMO), or you can associate the packages with different LMOs.

SEE ALSO:

[ISVforce Guide: Create or Edit Your AppExchange Listing](#)

[ISVforce Guide: Pass the AppExchange Security Review](#)

## Push a Package Upgrade for Second-Generation Managed Packages

Push upgrades enable you to upgrade second-generation managed packages installed in subscriber orgs, without asking customers to install the upgrade themselves. You can choose which orgs receive a push upgrade, what version the package is upgraded to, and when you want the upgrade to occur. Push upgrades are helpful if you need to push a change for a hot bug fix.

Use SOAP API to initiate the push upgrade, track the status of each job, and review error messages if any push upgrades fail. Here are the objects that help with push upgrades.

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*.

To Do This:	Use This Object:
Retrieve details about your package version.	<a href="#">MetadataPackageVersion</a> SOAP API
Retrieve information about the subscriber org, such as the org ID and the package version currently installed.	<a href="#">PackageSubscriber</a> SOAP API
Schedule a push upgrade, or check the status of the push upgrade.	<a href="#">PackagePushRequest</a> SOAP API
Specify the org to receive the push upgrade. Create an individual package push job for every org receiving the push upgrade.	<a href="#">PackagePushJob</a> SOAP API
Review any error messages associated with a push upgrade request.	<a href="#">PackagePushError</a> SOAP API

### Push Upgrade Considerations for Second-Generation Managed Packages

- You can push upgrades to only packages that have passed AppExchange security review.
- The same manageability rules for package version upgrades are applicable to push upgrades.
- When a push upgrade is installed, the Apex in package is compiled.
- Push upgrades can be used even if the package version requires a password.

#### [Schedule a Push Upgrade Using SOAP API for First- and Second-Generation Managed Packages](#)

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

#### [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.


## Schedule a Push Upgrade Using SOAP API for First- and Second-Generation Managed Packages

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

For code samples and more detailed steps, see SOAP API object documentation linked in each step.


1. Authenticate to your [Dev Hub org](#).

2. Query [MetadataPackage](#) to verify package details.
3. Query [MetadataPackageVersion](#) to verify the package version to use for the push upgrade.
4. Query [PackageSubscriber](#) to retrieve details about subscriber orgs such as the org ID and installed package version. To retrieve information about more than 2,000 subscribers, use SOAP API [queryMore \(\)](#) call.
5. Create a [PackagePushRequest](#) object. Specify the `PackageVersionId` and `ScheduledStartTime` (optional). If you omit the `ScheduledStartTime`, the push begins when you set the `PackagePushRequest`'s status to `Pending`.
6. Create a [PackagePushJob](#) for each subscriber and associate it with the `PackagePushRequest` you created in the previous step.
7. Schedule the push upgrade by changing the status of the `PackagePushRequest` to `Pending`.

 **Note:** Scheduled push upgrades begin as soon as resources are available on the Salesforce instance, which is either at or after the start time you specify. In certain scenarios, the push upgrade could start a few hours after the scheduled start time.

## 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.   <b>Note:</b> 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 317.

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 = 'vfpagel'];

        //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);  
    }  
}
```

## Advanced Features for Second-Generation Managed Packages

---

After you're comfortable with creating second-generation managed packages, learn about these advanced features to customize your package development processes.

### [Package Ancestors for Second-Generation Managed Packages](#)

Second-generation managed packaging (managed 2GP) offers a flexible package versioning model that lets you break your linear versioning and abandon a package version you no longer want to build upon. We call these versioning decisions *package ancestry*.

### [Patch Versions for Second-Generation Managed Packages](#)

Patch versions are a way to fix small issues with your second-generation managed package without introducing major feature changes. Customers who are using an older version of your package can install a patch and not be forced to upgrade to a new major package version.

### [Create Dependencies Between Second-Generation Managed Packages](#)

To avoid monolithic package development practices, you plan to develop smaller, modular packages that group similar functionality and components. You can then define the dependencies between these packages. A package dependency is when metadata contained in one package depends on metadata contained in another package. For example, defining dependencies allow you to extend the functionality of a base package with components and metadata located in a separate package.

### [Considerations for Promoting Packages with Dependencies](#)

If your company is developing a package that has a package dependency, ask yourself these questions before promoting (releasing) a new package version.

### [Advanced Project Configuration Parameters for Second-Generation Managed Packages](#)

As your managed 2GP package development becomes more complex, consider including these optional parameters in your `sfdx-project.json` file.

### [Second-Generation Managed Packaging Keywords](#)

A keyword is a variable that you can use to specify a package version number.

### [Target a Specific Release for Your Second-Generation Managed Packages During Salesforce Release Transitions](#)

During major Salesforce release transitions, you can specify `preview` or `previous` when creating a package version. Specifying the release version for a package allows you to test upcoming features, run regression tests, and support customers regardless of which Salesforce release their org is on. Previously, you could only create package versions that matched the Salesforce release your Dev Hub org was on.



[Use Branches in Second-Generation Managed Packaging](#)

Development teams who use branches in their source control system (SCS), often build package versions based on the metadata in a particular branch of code.

[Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages](#)

For scenarios where you require metadata that isn't part of your second-generation managed package, but is necessary for Apex test runs, you can specify the path containing unpackaged metadata in the `sfdx-project.json` file. The unpackaged metadata isn't included in the package and isn't installed in subscriber orgs.

[Package IDs and Aliases for Second-Generation Managed Packages](#)

During the package lifecycle, packages and package versions are identified by an ID or package alias. When you create a second-generation managed package or package version, Salesforce CLI creates a package alias based on the package name, and stores that name in the `packageAliases` section of the `sfdx-project.json` file. When you run CLI commands or write scripts to automate packaging workflows, it's often easier to reference the package alias, instead of the package ID or package version ID.

[Avoid Namespace Collisions in Second-Generation Managed Packages](#)

Namespaces impact the combination of package types that you can install in an org.

[Remove Metadata Components from Second-Generation Managed Packages](#)

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

[Delete a Second-Generation Managed Package or Package Version](#)

Use the `sf package version delete` and `sf package delete` commands to delete packages and package versions that you no longer need.

[Frequently Used Packaging Operations for Second-Generation Managed Packages](#)[Transfer a Second-Generation Managed Package to a Different Dev Hub](#)

You can transfer the ownership of a second-generation managed package (managed 2GP) from one Dev Hub org to another. These transfers can occur either internally between two Dev Hub orgs your company owns, or you can transfer a package externally to another Salesforce Partner or ISV. This change provides a way to sell a second-generation managed package to a different company.

## Package Ancestors for Second-Generation Managed Packages

Second-generation managed packaging (managed 2GP) offers a flexible package versioning model that lets you break your linear versioning and abandon a package version you no longer want to build upon. We call these versioning decisions *package ancestry*.

 **Note:** Only package versions that have been promoted to the managed-released state can be specified as a package ancestor.

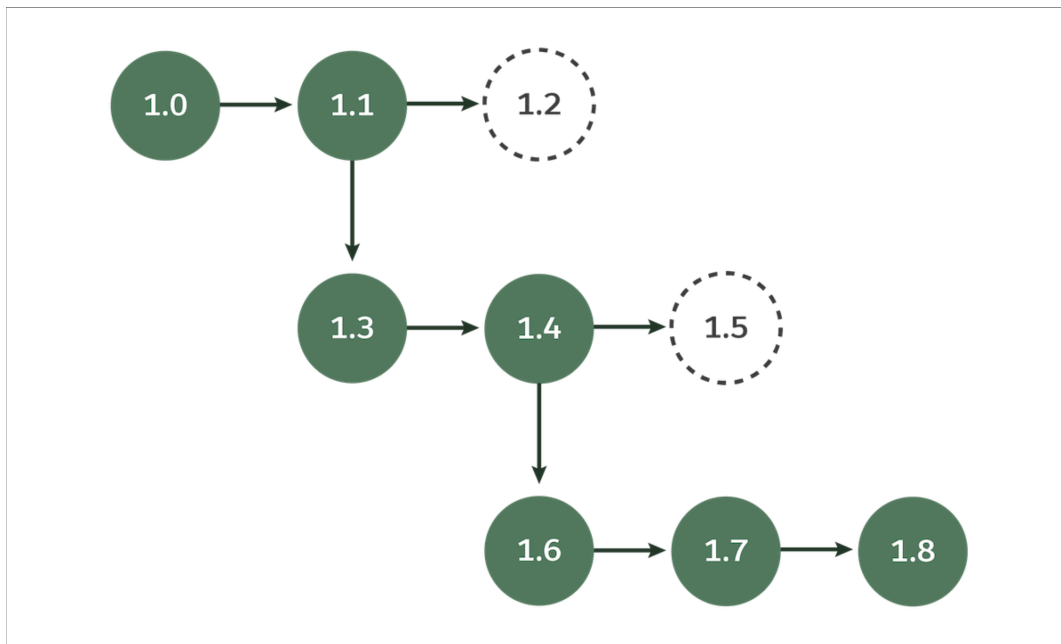
When package versioning is linear, the package version number (formatted as `major.minor.patch.build`) always increments to an increasing number. For example, looking at just the major and minor version numbers, linear versioning looks something like 1.0 1.1 1.2 2.0. The next package version created in this linear versioning example must be higher than 2.0.

### How Managed 2GP Package Versioning Affects Package Upgrades

Before we dig into package ancestry and how managed 2GP lets you break your linear versioning, let's clarify how package versioning impacts package upgrades. Let's use our previous example of a package version history that looks like this, 1.0 1.1 1.2 2.0. A customer could install version 1.0 and upgrade through each of the subsequent package versions, or they could skip versions and upgrade from 1.0 to 2.0. As long as they upgrade from a lower package version number to a higher package version number, the package upgrade succeeds.

But what if during your development process you create a package version that you don't want to build upon? Managed 2GP lets you break free from linear versioning and select a different package version to build upon.

Say your team creates version 1.0, then 1.1, then 1.2 and oops! 1.2 made a mess of 1.1. Not a problem. When you create a package version, you specify which package version is the ancestor. So you abandon 1.2, and make 1.1 the ancestor of 1.3. And this process can be repeated. For example, the illustration shows how to abandon 1.5, and build 1.6 off 1.4.



This more complex and tree-like versioning has the added benefit of making it possible for two or more development teams to do parallel package development.

## With Great Power Comes Great Responsibility

The flexibility to break from linear versioning is powerful. But remember that if abandoned versions like 1.2 and 1.5 are installed in customer orgs, those customers no longer have an upgrade path. Packages can only upgrade along the ancestry line. For example, you can upgrade from version 1.1 to 1.7, but not from version 1.5 to 1.7.

## Patch Versions and Package Ancestry

You can't specify a [patch version](#), such as 1.0.2, as a direct ancestor of a non-patch version. Instead, use the [keyword](#) "ancestorVersion" : "HIGHEST", or specify a non-patch version as the ancestor. Installed patch versions inherit the upgrade path of the non-patch version with the same major and minor number. For example, patch version 1.0.3 has the same upgrade path as 1.0.0.

### [Understanding Package Upgrades with Ancestry](#)

Review how package ancestry impacts which package version upgrades are allowed.

[View Package Ancestry](#)

Use Salesforce CLI commands to quickly confirm your package’s ancestor, or to create a visualization of the package ancestry tree.

SEE ALSO:

[Understanding Package Upgrades with Ancestry](#)

[View Package Ancestry](#)

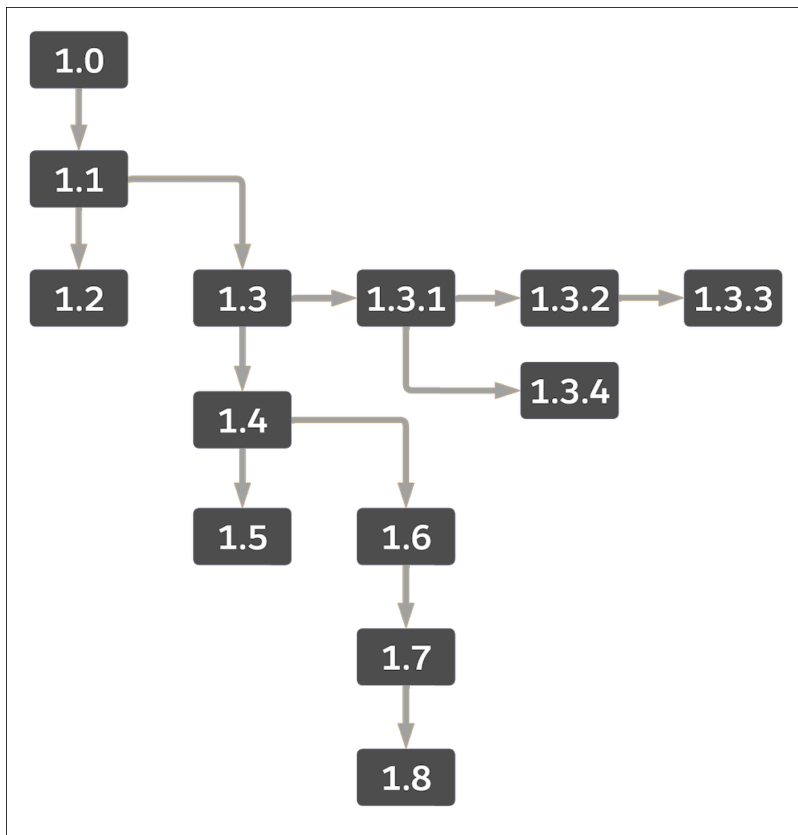
[Namespace-Based Visibility for Apex Classes in Second-Generation Managed Packages](#)

## Understanding Package Upgrades with Ancestry

Review how package ancestry impacts which package version upgrades are allowed.

Refer to this table and the package ancestry tree to understand whether your subscribers can upgrade between these 2GP package versions.

**Example Package Ancestry Tree**



Upgrade From	Upgrade To	Will This Package Upgrade Succeed?
1.1	1.7	Yes
1.3.2	1.3.4	Yes. Both 1.3.2 and 1.3.4 are patch versions within the same major and minor version. You’re allowed to upgrade between patch

Upgrade From	Upgrade To	Will This Package Upgrade Succeed?
		versions that share the same major and minor version.
1.3.1	1.7	Yes. 1.3.1 is a patch version. Because upgrading from 1.3 to 1.7 is allowed, you can also upgrade from 1.3.x to 1.7.
1.2	1.3	No. These two versions don't share an ancestry path.
1.5	1.7	No. These two versions don't share an ancestry path.
1.4	1.3.3	No. Downgrading an installed package isn't allowed.
1.0	1.8	Yes

## View Package Ancestry

Use Salesforce CLI commands to quickly confirm your package's ancestor, or to create a visualization of the package ancestry tree.

### View Package Ancestor Details in Salesforce CLI

Use the `sf package version report` or `sf package version list` command to view the name and version number of the package ancestor.

Output from `sf package version report` command.

```

=== Package Version
Name                               Value
-----
Name                               ver 0.2
Subscriber Package Version Id      04txx00000040T0AAM
Package Id                         0H0xx0000004G2yCAE
Version                            0.2.0.1
Description
Branch
Tag
Released                           false
Validation Skipped                  false
Ancestor                           04txx0000004MnmAAE
Ancestor Version                   0.1.0.2
Code Coverage                       17%
Code Coverage Met                   false
    
```

Output from `sf package version list` command.

```

Validation Skipped  Ancestor      Ancestor Version  Branch
-----
false              04txx0000004M10AAU  0.7.0.1
false
    
```

## Visualize Package Ancestry

Use the `displayancestry` CLI command to create visualizations of your package or package version's ancestry tree. You can view the visualization in Salesforce CLI or use the `dot-code` parameter to generate output that can be used in graph visualization software.

Use `sf package version displayancestry` to quickly visualize your package ancestry and understand the possible package upgrade paths.

```
$ sfdx force:package:version:displayancestry --package 0Hoxx0000004V00CAU
├── 0.1.0.1
│   ├── 0.2.0.1
│   │   ├── 0.2.0.2
│   │   ├── 0.3.0.1
│   │   ├── 0.3.0.2
│   │   └── 0.3.0.3
│   └── 0.4.0.1
└── 1.0.0.1
    ├── 1.1.0.1
    └── 1.0.0.2
```

```
$ sfdx force:package:version:displayancestry --package 04txx0000004k94AAA
0.2.0.2 -> 0.1.0.1 (root)
├── 0.2.0.2
│   ├── 0.3.0.1
│   ├── 0.3.0.2
│   └── 0.3.0.3
└── 0.4.0.1
    ├── 1.0.0.1
    ├── 1.1.0.1
    └── 1.0.0.2
```

To generate `dotcode` output, specify `sf package version displayancestry --dot-code`.

## Patch Versions for Second-Generation Managed Packages

Patch versions are a way to fix small issues with your second-generation managed package without introducing major feature changes. Customers who are using an older version of your package can install a patch and not be forced to upgrade to a new major package version.

Package versions follow a `major.minor.patch.build` number format. Any package version number that contains a non-zero patch number is a patch version. For example, `1.1.2.5` is a patch version, but `1.1.0.4` isn't.

Patch versions are intended for small changes like a fixing a bug. You can't:

- Add package components.
- Delete existing package components.
- Change the API and dynamic Apex access controls.
- Deprecate any Apex code.
- Add new Apex class relationships, such as extends.
- Add Apex access modifiers, such as virtual or global.
- Add features, settings, package dependencies, or web services.
- Change a component from protected to global.
- Change the visibility of CustomSettings or CustomMetadataType from protected to public.

When creating a patch version, you must specify the package ancestor. The major and minor numbers of the patch version and the package ancestor must match. And the specified package ancestor must be managed-released.

You can specify another patch version as the package ancestor of a patch version. But you can't specify a patch version as a direct ancestor of a non-patch version. Instead, use the keyword `"ancestorVersion" : "HIGHEST"`, or specify a non-patch version as the ancestor.

Installed patch versions inherit the upgrade path of the non-patch version with the same major and minor number. For example, patch version 1.0.3 has the same upgrade path as 1.0.0. See [Specify a Package Ancestor in the Project File for a Second-Generation Managed Package](#) for more information about how to specify a package ancestor.

When you create a patch version, the patch automatically inherits the features and settings defined in the package ancestor's scratch org definition file. To create a patch, follow the same steps as you do when you create a package version, and increment the patch number.

 **Note:** To enable patch versioning, log a case in the [Salesforce Partner Community](#) and request that patch versioning be enabled in the org where you created the namespace for this package. Patch versioning is available only to packages that have passed AppExchange security review.

SEE ALSO:


[Specify a Package Ancestor in the Project File for a Second-Generation Managed Package](#)

[Second-Generation Managed Packaging Keywords](#)

## Create Dependencies Between Second-Generation Managed Packages

To avoid monolithic package development practices, you plan to develop smaller, modular packages that group similar functionality and components. You can then define the dependencies between these packages. A package dependency is when metadata contained in one package depends on metadata contained in another package. For example, defining dependencies allow you to extend the functionality of a base package with components and metadata located in a separate package.

### How to Specify a Managed 2GP Package Dependency

 **Note:** To understand which combination of managed 2GP and managed 1GP package dependencies are supported, see [Which Package Types Can Your Package Depend On?](#)

To specify dependencies between managed packages associated with the same Dev Hub, use either the package version alias or a combination of the package name and the version number.

Example 1:


```
"dependencies": [
  {
    "package": "MyPackageName@0.1.0.1"
  }
]
```

Example 2:

```
"dependencies": [
  {
    "package": "MyPackageName",
    "versionNumber": "1.0.0.RELEASED"
  }
]
```

To specify a dependency on a managed package that isn't associated with your Dev Hub:

```
"dependencies": [
  {
    "package": "04txxx"
  }
]
```

 **Note:** You can use the RELEASED keyword for the version number to set the dependency.

To denote dependencies with package IDs instead of package aliases, use:

- The 0H0 ID if you specify the package ID along with the version number
- The 04t ID if you specify only the package version ID

## Specifying Multiple Package Dependencies

If your package has more than one dependency, provide a comma-separated list of packages in the order of installation.

For example, if your package depends on the package Expense Manager - Util, which in turn depends on the package External Apex Library, the package dependencies are:

```
"dependencies": [
  {
    "package" : "External Apex Library - 1.0.0.4"
  },
  {
    "package": "Expense Manager - Util",
    "versionNumber": "4.7.0.RELEASED"
  }
]
```

## Which Types of Dependencies Are Supported?

### Circular Dependencies

Circular dependencies among packages aren't supported.

A circular dependency occurs when pkgC depends on pkgB, pkgB depends on pkgA, and pkgA depends on pkgC.

### Multi-level Dependencies

Multi-level package dependencies are supported.

A multi-level dependency occurs when pkgC depends on pkgB, and pkgB depends on pkgA.

List multi-level dependencies in the `sfdx-project.json` file in package installation order. In this example, pkgA must be installed first, followed by pkgB, and then pkgC.

```
{
  "packageDirectories": [
    {
      "path": "pkgA-wsp",
      "default": true,
      "package": "pkgA",
      "versionName": "ver 1.3",
```

```

    "versionNumber": "1.3.0.NEXT",
    "ancestorVersion": "1.1.0.RELEASED"
  },
  {
    "path": "pkgB-wsp",
    "default": false,
    "package": "pkgB",
    "versionName": "ver 2.3",
    "versionNumber": "2.3.0.NEXT",
    "ancestorVersion": "2.0.0.RELEASED",
    "dependencies": [
      {
        "package": "pkgA@1.1.0.RELEASED"
      }
    ]
  },
  {
    "path": "pkgC-wsp",
    "default": false,
    "package": "pkgC",
    "versionName": "ver 0.1",
    "versionNumber": "0.1.0.NEXT",
    "dependencies": [
      {
        "package": "pkgA@1.1.0.RELEASED"
      },
      {
        "package": "pkgB@2.0.0.RELEASED"
      }
    ]
  }
],
}

```

The specified package version number also impacts the installation of package dependencies. Before pkgB can be installed, pkgA version 1.1 or higher must first be installed. If this condition isn't met, the installation of pkgB fails.

#### SEE ALSO:

[Which Package Types Can Your Package Depend On?](#)

[Considerations for Promoting Packages with Dependencies](#)

## Considerations for Promoting Packages with Dependencies

If your company is developing a package that has a package dependency, ask yourself these questions before promoting (releasing) a new package version.

Are you:

- Developing the base and extension package in parallel?
- Specifying skip validation when creating new package versions?



- Using the keywords `LATEST` or `RELEASED` when specifying the package dependency?

If you answered no to all these questions, your package doesn't have any tricky dependency scenarios and you can promote it when it's ready. If you answered yes to any of these questions, keep reading.

## Specifying Skip Validation

When you create a package version and specify [skip validation](#), the version is created without validating dependencies, package ancestors, or metadata.

If you develop your base package using skip validation, test your extension package using either a stable and previously promoted version of the base package, or a non-skip validated base package version.

Most importantly, if you're developing a version of your base package and extension package in parallel, ensure that you:

- First promote the base package version.
- Then specify the promoted package version in the dependency section of your extension package using the keyword `RELEASED`.
- Finally, create the extension package version.

After testing the extension package version, you then promote it. This process ensures that the extension package version that you promote to the released state has as its dependency the promoted base package version.

## Using the Keyword `LATEST` or `RELEASED`

A keyword is a variable that you can use to specify a package version number. The keyword `LATEST` maps to the most recently created package version, which might not be the same as the promoted and released package version.

The keyword `RELEASED` maps to the promoted and released package version.

For example: If you create versions 1.0.0.1, 1.0.0.2, and 1.0.0.3, and promote version 1.0.0.2, then `1.0.0.RELEASED` = 1.0.0.2, but `1.0.0.LATEST` = 1.0.0.3.

## Example

Your company created a base package called `PkgBase`, and an extension package called `PkgExtn`.

`PkgBase` is under active development, and the development team is creating versions that specify `--skip-validation`.

`PkgExtn` version 2.3 is under active development and references its dependency on `PkgBase` by using the following definition in the `sfdx-project.json`.

```
{
  "path": "pkg-extension",
  "default": false,
  "package": "PkgExtn",
  "versionName": "v 2.3",
  "versionDescription": "Winter 2025",
  "versionNumber": "2.3.0.NEXT",
  "dependencies": [
    {
      "package": "PkgBase",
      "versionNumber": "1.1.0.LATEST"
    }
  ],
}
```

Before promoting version 2.3 of PkgExtn, you must test it using the promoted version 1.1.0 of PkgBase. Update the PkgExtn dependency section of your `sfdx-project.json` and change the dependency from 1.1.0.LATEST to 1.1.0.RELEASED. If the tests succeed, then create a new version of PkgExtn and ensure it works as expected with the promoted base package version.

## SEE ALSO:

[Create and Update Versions of a Second-Generation Managed Package](#)

[Get Ready to Promote and Release a Second-Generation Managed Package Version](#)

[Create Dependencies Between Second-Generation Managed Packages](#)

[Second-Generation Managed Packaging Keywords](#)

## Advanced Project Configuration Parameters for Second-Generation Managed Packages

As your managed 2GP package development becomes more complex, consider including these optional parameters in your `sfdx-project.json` file.

Name	Details
apexTestAccess	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p> <p>Assign permission sets and permission set licenses to the user in context when your Apex tests run at package version creation.</p> <pre data-bbox="820 1050 1445 1344"> "apexTestAccess": {   "permissionSets": [     "Permission_Set_1",     "Permission_Set_2"   ],   "permissionSetLicenses": [     "SalesConsoleUser"   ] } </pre> <p>See <a href="#">Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages</a></p>
branch	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p> <p>If your package has an associated branch, but your package dependency is associated with a different branch, use this format.</p> <pre data-bbox="820 1638 1445 1858"> "dependencies": [   {     "package": "pkgB",     "versionNumber": "1.3.0.LATEST",     "branch": "featureC"   } ] </pre>

Name	Details
	<p>If your package has an associated branch, but your package dependency doesn't have an associated branch, use this format.</p> <pre data-bbox="824 342 1448 573">"dependencies": [   {     "package": "pkgB",     "versionNumber": "1.3.0.LATEST",     "branch": ""   } ]</pre> <p>See <a href="#">Use Branches in Second-Generation Managed Packaging</a></p>
dependencies	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p> <p>Specify the dependencies on other packages.</p> <p>To specify dependencies for managed packages within the same Dev Hub, use either the package version alias or a combination of the package name and the version number.</p> <pre data-bbox="824 909 1448 1077">"dependencies": [   {     "package": "MyPackageName@1.1.0.1"   } ]</pre> <pre data-bbox="824 1098 1448 1287">"dependencies": [   {     "package": "MyPackageName",     "versionNumber": "1.1.0.RELEASED"   } ]</pre> <p>To specify dependencies for managed packages outside of the Dev Hub use:</p> <pre data-bbox="824 1392 1448 1560">"dependencies": [   {     "package": "04txxx"   } ]</pre> <p>To set the dependency, you can use the <a href="#">keywords</a> RELEASED or LATEST for the version number.</p> <p>To denote dependencies with package IDs instead of package aliases, use:</p> <ul data-bbox="824 1738 1448 1843" style="list-style-type: none"> <li>• The 0H0 ID if you specify the package ID along with the version number</li> <li>• The 04t ID if you specify only the package version ID</li> </ul>

Name	Details
	<p>If the package has more than one dependency, provide a comma-separated list of packages in the order of installation. For example, if a package depends on the package Expense Manager - Util, which in turn depends on the package External Apex Library, the package dependencies are:</p> <pre data-bbox="824 443 1448 831"> "dependencies": [   {     "package" : "External Apex Library - 1.0.0.4"   },   {     "package": "Expense Manager - Util",     "versionNumber": "4.7.0.RELEASED"   } ] </pre> <p>See: <a href="#">Considerations for Promoting Packages with Dependencies</a></p>
postInstallScript	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p> <p>An Apex script that runs automatically in the subscriber org after the managed package is installed or upgraded.</p>
postInstallURL	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p> <p>A URL to post-install instructions for subscribers.</p>
releaseNotesUrl	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p> <p>A URL to release notes.</p>
scopeProfiles	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> false</p> <p>The scopeProfiles parameter is a child of packageDirectories. If you set scopeProfiles to true for a package directory, profile settings from only the package directory being packaged are included, and profile settings outside of that package directory are ignored.</p> <p>When you set scopeProfiles to false (the default value), the new package version includes relevant pieces of profile settings in any package directory defined in <code>sfdx-project.json</code>.</p>
unpackagedMetadata	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p>

Name	Details
	See <a href="#">Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages</a> .
uninstallScript	<p><b>Required?</b> No</p> <p><b>Default if Not Specified:</b> None</p> <p>An Apex script to run automatically in the subscriber org before the managed package is uninstalled.</p>

## SEE ALSO:

[Project Configuration File for a Second-Generation Managed Package](#)

## Second-Generation Managed Packaging Keywords

A keyword is a variable that you can use to specify a package version number.

You can use keywords to automatically increment the value of the package build numbers, ancestor version numbers, set the package dependency to the latest version, or the latest released and promoted version.

Use the Keyword	Example
LATEST to specify the latest version of the package dependency when you create a package version.	<pre>"dependencies": [   {     "package": "MyPackageName",     "versionNumber": "0.1.0.LATEST"   } ]</pre>
<p>NEXT to increment the build number to the next available for the package version.</p> <p>If you don't use <code>NEXT</code>, and you also forget to update the version number in your <code>sfdx-project.json</code> file, the new package version uses the same number as the previous package version. Although we don't enforce uniqueness on package version numbers, every package version is assigned a unique subscriber package version ID (starts with 04t).</p>	<pre>"versionNumber": "1.2.0.NEXT"</pre>
RELEASED to specify the latest promoted and released version of the package dependency when you create a package version.	<pre>"dependencies": [   {     "package": "pkgB",     "versionNumber": "2.1.0.RELEASED"   } ]</pre>
HIGHEST to automatically set the package ancestor to the highest promoted and released package version number.	<pre>"packageDirectories": [   {</pre>

Use the Keyword	Example
Use only with ancestor version or ancestor ID.	<pre>"path": "util", "package": "Expense Manager - Util", "versionNumber": "4.7.0.NEXT", "ancestorVersion": "HIGHEST" },</pre>
<p>NONE in the ancestor version or ancestor ID field.</p> <p>Ancestry defines package upgrade paths. If the package ancestor is set to NONE, an existing customer can't upgrade to that package version.</p>	<pre>"packageDirectories": [ { "path": "util", "package": "Expense Manager - Util", "versionNumber": "4.7.0.NEXT", "ancestorVersion": "NONE" },</pre>

## Target a Specific Release for Your Second-Generation Managed Packages During Salesforce Release Transitions

During major Salesforce release transitions, you can specify `preview` or `previous` when creating a package version. Specifying the release version for a package allows you to test upcoming features, run regression tests, and support customers regardless of which Salesforce release their org is on. Previously, you could only create package versions that matched the Salesforce release your Dev Hub org was on.

To create a package version based on a preview or previous Salesforce release version, create a scratch org definition file that includes either:

```
{
  "release": "previous"
}
```

or

```
{
  "release": "preview"
}
```

In the `sfdx-project.json` file, set the `sourceApiVersion` to correspond with the release version of the package version you're creating. If you are targeting a previous release, any `sourceApiVersion` value below the current release is accepted.

Then when you create your package version, specify the scratch org definition file.

```
sf package version create --package pkgA --definition-file config/project-scratch-def.json
```

Preview start date is when sandbox instances are upgraded. Preview end date is when all instances are on the GA release.

Release Version	Preview Start Date	Preview End Date
Winter '25	September 1, 2024	October 12, 2024
Spring '25	January 5, 2025	February 15, 2025
Summer '25	May 11, 2025	June 14, 2025

## Use Branches in Second-Generation Managed Packaging

Development teams who use branches in their source control system (SCS), often build package versions based on the metadata in a particular branch of code.

To identify which branch in your SCS a package version is based on, tag your package version with a branch name using `--branch` attribute in this Salesforce CLI command.

```
sf package version create --branch featureA
```

You can specify any alphanumeric value up to 240 characters as the branch name.

You can also specify the branch name in the package directories section of the `sfdx-project.json` file.

```
"packageDirectories": [
  {
    "path": "util",
    "default": true,
    "package": "pkgA",
    "versionName": "Spring '21",
    "versionNumber": "4.7.0.NEXT",
    "branch": "featureA"
  }
]
```

When you specify a branch, the package alias for that package version is automatically appended with the branch name. You can view the package alias in the `sfdx-project.json` file.

```
"packageAliases": {
  "pkgA@1.0.0.4-featureA": "04tB0000000IB1EIAW"
}
```

Keep in mind that version numbers increment within each branch, and not across branches. For example, you could have two or more beta package versions with the version number 1.3.0.1.

Branch Name	Package Version Alias
featureA	pkgA@1.3.0-1-featureA
featureB	pkgA@1.3.0-1-featureB
Not specified	pkgA@1.3.0-1

Although more than one beta package version can have the same version number, there can be only one promoted and released package version for a given major.minor.patch package version.

## Package Dependencies and Branches

By default, your package can have dependencies on other packages in the same branch. For package dependencies based on packages in other branches, explicitly set the branch attribute in the `sfdx-project.json` file.

To specify a package dependency	Use this format
Using the branch attribute	<pre>"dependencies": [   {     "package": "pkgB",</pre>

To specify a package dependency	Use this format
	<pre>"versionNumber": "1.3.0.LATEST", "branch": "featureC" }]</pre>
Using the most recent promoted and released version of package	<pre>"dependencies": [   {     "package": "pkgB",     "versionNumber": "2.1.0.RELEASED"   } ]</pre>
If your package has an associated branch, but the dependent package doesn't have a branch	<pre>"dependencies": [   {     "package": "pkgB",     "versionNumber": "1.3.0.LATEST",     "branch": ""   } ]</pre>
Using the package alias	<pre>"dependencies": [   {     "package": "pkgB@2.1.0-1-featureC"   } ]</pre>

## Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages

For scenarios where you require metadata that isn't part of your second-generation managed package, but is necessary for Apex test runs, you can specify the path containing unpackaged metadata in the `sfdx-project.json` file. The unpackaged metadata isn't included in the package and isn't installed in subscriber orgs.

### Specify Unpackaged Metadata for Package Version Creation Tests

Specify the path to the unpackaged metadata in your `sfdx-project.json` file.

In this example, metadata in the `my-unpackaged-directory` is available for test runs during the package version creation of the `TV_un1` package.

```
"packageDirectories": [
  {
    "path": "force-app",
    "package": "TV_un1",
    "versionName": "ver 0.1",
    "versionNumber": "0.1.0.NEXT",
    "default": true,
    "unpackagedMetadata": {
      "path": "my-unpackaged-directory"
    }
  }
]
```



```
    },
  ]
}
```

The `unpackagedMetadata` attribute is intended for metadata that isn't part of your package. You can't include the same metadata in both an unpackaged directory and a packaged directory.

## Manage Apex Access for Package Version Creation Tests

Sometimes the Apex tests that you write require a user to have certain permission sets or permission set licenses. Use the `apexTestAccess` setting to assign permission sets and permission set licenses to the user in whose context your Apex tests get run at package version creation.

```
"packageDirectories": [
  {
    "path": "force-app",
    "package": "TV_un1",
    "versionName": "ver 0.1",
    "versionNumber": "0.1.0.NEXT",
    "default": true,
    "unpackagedMetadata": {
      "path": "my-unpackaged-directory"
    },
    "apexTestAccess": {
      "permissionSets": [
        "Permission_Set_1",
        "Permission_Set_2"
      ],
      "permissionSetLicenses": [
        "SalesConsoleUser"
      ]
    }
  },
],
]
```


 **Note:** To assign user licenses, use the [runAs Method](#). User licenses can't be assigned in the `sfdx-project.json` file.

## Package IDs and Aliases for Second-Generation Managed Packages

During the package lifecycle, packages and package versions are identified by an ID or package alias. When you create a second-generation managed package or package version, Salesforce CLI creates a package alias based on the package name, and stores that name in the `packageAliases` section of the `sfdx-project.json` file. When you run CLI commands or write scripts to automate packaging workflows, it's often easier to reference the package alias, instead of the package ID or package version ID.

Package aliases are stored in the `sfdx-project.json` file as name-value pairs, in which the name is the alias and the value is the ID. You can modify package aliases for existing packages and package versions in the project file.

At the command line, you also see IDs for things like package members (a component in a package) and requests (like a `sf package version create` request).


 **Note:** As a shortcut, the documentation sometimes refers to an ID by its three-character prefix. For example, a package version ID always starts with `04t`.

Here are the most commonly used IDs.

ID Example	Short ID Name	Description
033J0000dAb27uxVRE	Subscriber Package ID	Use this ID when contacting Salesforce for packaging or security review support. To locate this ID for your package, run <code>sf package list --verbose</code> against the Dev Hub that owns the package.
04t6A0000004eytQAA	Subscriber Package Version ID	Use this ID to install a package version. Returned by <code>sf package version create</code> .
0Hox00000000CqCAI	Package ID	Use this ID on the command line to create a package version. Or enter it into the <code>sfdx-project.json</code> file and use the directory name. Generated by <code>sf package create</code> .
08cxx00000000BEAAY	Version Creation Request ID	Use this ID to view the status and monitor progress for a specific request to create a package version such as <code>sf package version create report</code>

## Avoid Namespace Collisions in Second-Generation Managed Packages

Namespaces impact the combination of package types that you can install in an org.

 **Important:** When sharing a namespace, be intentional about managing component names across packages within that namespace. Ensure that packages associated with the same namespace don't include components with the same API name. If two packages include a component with the same API name, you can't install these packages into the same org.

To understand how namespaces affect the types of packages you can install in a namespaced or no-namespace org, review this table.

Installation Org	No-namespace Unlocked Package	Namespaced Unlocked Package	Second-generation Managed Package (2GP)	First-generation Managed Package (1GP)
<b>Org with a namespace</b> For example, a 1GP packaging org, 1GP patch org, Developer Edition org with namespace, or a scratch org with namespace	Fail. You can't install a no-namespace unlocked package in an org with a namespace.	Pass. Regardless of whether the namespace matches or is different from the org's namespace, you can install one or many namespaced unlocked packages.	Pass (scratch orgs). Regardless of whether the namespace matches or is different from the scratch org's namespace, you can install one or many 2GP packages. Fail (1GP packaging and patch orgs). To prevent 1GP packages from depending on 2GP packages, we block the installation of 2GP	Pass. If the namespace of the 1GP is different from the namespace of the org, you can install one or many packages. Fail. If the namespace of the 1GP is the same as the namespace of the org, you can't install the 1GP into the org.

Installation Org	No-namespace Unlocked Package	Namespaced Unlocked Package	Second-generation Managed Package (2GP)	First-generation Managed Package (1GP)
			packages in a 1GP packaging or patch org. We also block the installation of 2GP packages in Developer Edition (DE) orgs that have an associated namespace, unless it's a DE scratch org.	
<b>Org without a namespace</b>	Pass. You can install one or many no-namespace unlocked packages.	Pass. You can install one or many namespaced unlocked packages.	Pass. You can install one or many 2GP packages.	Pass. You can install one or many 1GP packages.

To understand how namespaces affect the combination of packages that can be installed into one org, review this table.

Namespace and Package Type	Unlocked Package with Namespace Y	Second-generation Managed Package (2GP) with Namespace Y	First-generation Managed Package (1GP) with Namespace Y
<b>First-generation Managed Package (1GP) with namespace X</b>	Pass. If the 1GP and unlocked package use unique namespaces, you can install them in the same org.	Pass. If the 1GP and 2GP use unique namespaces, you can install them in the same org.	Pass. If each 1GP uses a unique namespace, you can install multiple 1GP packages in the same org.
<b>First-generation Managed Package (1GP) with namespace Y</b>	Fail. If the 1GP and unlocked package share a namespace, you can't install them in the same org.	Fail. If the 1GP and 2GP share a namespace, you can't install them in the same org.	Fail. If the 1GP packages share a namespace, you can't install them in the same org.
<b>Second-generation Managed Package (2GP) with namespace X</b>	Pass. You can install a 2GP and a namespaced unlocked package in the same org. The packages can share a namespace or use unique namespaces.	Pass. You can install multiple 2GP packages with unique namespaces, or the same namespace.	Pass. If the 1GP and 2GP use unique namespaces, you can install them in the same org.

Namespace and Package Type	Unlocked Package with Namespace Y	Second-generation Managed Package (2GP) with Namespace Y	First-generation Managed Package (1GP) with Namespace Y
<b>Second-generation Managed Package (2GP) with namespace Y</b>	Pass. You can install a 2GP and a namespaced unlocked package in the same org. The packages can share a namespace or use unique namespaces.	Pass. You can install multiple 2GP packages with the same namespace in the same org.	Fail. If the 1GP and 2GP share a namespace, you can't install them in the same org.

## SEE ALSO:

[Namespaces for Second-Generation Managed Packages](#)

[Create and Register Your Namespace for Second-Generation Managed Packages](#)

[Link a Namespace to a Dev Hub Org](#)

## Remove Metadata Components from Second-Generation Managed Packages

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

### Impact of Component Removal in Subscriber Orgs

During a package upgrade, only certain component types are hard deleted and removed from the subscriber org. Most metadata components that were removed from a package version remain in the subscriber org after package upgrade and are 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. If the subscriber uninstalls the package, deprecated metadata that was previously associated with the package is deleted.

You can remove these metadata components from second-generation managed packages.

Metadata Component	Upon Package Upgrade, the Metadata Component is ...
Analytic Snapshot	Marked as deprecated
Apex Class (excluding global Apex classes)	Hard deleted
Apex Trigger	Hard deleted
Aura Component	Marked as deprecated
Compact Layout	Marked as deprecated
Custom Application	Marked as deprecated
Custom Application Component	Marked as deprecated
Custom Field	Marked as deprecated

<b>Metadata Component</b>	<b>Upon Package Upgrade, the Metadata Component is ...</b>
Custom Labels	Marked as deprecated
Custom Metadata Type Records	Marked as deprecated, if visible to the subscriber org; otherwise, hard deleted.
Custom Object	Marked as deprecated
Custom Permission	Marked as deprecated
Custom Tab	Marked as deprecated
Dashboard	Marked as deprecated
Dashboard Folder	Marked as deprecated
Document	Marked as deprecated
External Client App Header	Hard deleted
External Client App Settings	Hard deleted
External Credential	Marked as deprecated
Field Set	Marked as deprecated
Flow	Marked as deprecated
Lightning Page	Marked as deprecated
Lightning Web Component	Marked as deprecated
List View	Marked as deprecated
Named Credential	Marked as deprecated
Page Layout	Marked as deprecated
Permission Set	Marked as deprecated
Platform Event Channel	Hard deleted
Platform Event Channel Member	Hard deleted
Profile	Marked as deprecated
Quick Action	Marked as deprecated
Record Type	Marked as deprecated
Remote Site Setting	Marked as deprecated
Report	Marked as deprecated
Report Folder	Marked as deprecated
Report Type	Marked as deprecated
Sharing Reason	Marked as deprecated

Metadata Component	Upon Package Upgrade, the Metadata Component is ...
Static Resource	Marked as deprecated
Validation Rule	Marked as deprecated
Visualforce Component (excluding global components)	Hard deleted
Visualforce Page	Marked as deprecated
WebLink (Custom Button or Custom Link)	Marked as deprecated
Workflow Email Alert, Workflow Field Update, Workflow Outbound Message, Workflow Rule, Workflow Task	Marked as deprecated

## How to Remove Metadata Components

To request access to this feature, log a case at [Salesforce Partner Community](#).

After your request is approved, remove the metadata component's source file from your Salesforce DX project, and create a package version. Test the new package version to ensure it's working properly without the removed metadata.

## Before You Remove Metadata Components from Second-Generation Managed Packages

To ensure you can successfully remove metadata components from a second-generation managed package, keep these details in mind.

- Request access to the feature, if you haven't already.
- Familiarize yourself with the list of metadata components that can be removed.
- Ensure that there aren't 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, you can't access any customizations that depend on the removed component.

## Remove Metadata Dependencies Within a Package

If there are dependencies to the metadata component you plan to remove, resolve the dependency before removing the metadata component.

For example, before deleting a custom field that is referenced in a page layout, edit the page layout and remove the reference to the custom field. Then remove the custom field from your source file, and create a package version.

Some scenarios require a two-step approach to component removal. For example, let's say you plan to remove a Visualforce page that contains a Visualforce component and replace it with a Lightning page that contains a Lightning component. Removing both the Visualforce page and Visualforce component in a single upgrade could cause issues for your subscribers. These issues occur because Visualforce components are deleted, and Visualforce pages are deprecated during package upgrade.

To avoid issues for your subscribers in this example, remove the reference to the Visualforce component from the Visualforce page, create a package version, and push the upgrade. Then remove the Visualforce page from your package version, and push this upgrade to subscribers.

## Remove Dependencies Located in Other Packages

Before you remove a metadata component, first remove all references to the metadata, including references in other packages that depend on that metadata component. For example, if you're removing a public Apex class, ensure your other packages aren't referencing that class using the Apex `@namespaceAccessible` annotation.

In this section, PackageA refers to the package in which you plan to remove a metadata component. And PackageB is any package that depends on the metadata you're removing from PackageA. If you have references to the metadata component or Apex class in PackageB, follow these steps:

1. Remove the reference to the metadata component from PackageB.
2. Create a version of PackageB.
3. Push the new version of PackageB to your subscribers.
4. Repeat these steps if any other packages include a reference to the metadata you plan to remove from PackageA.

After you've removed all references to the metadata component, remove the metadata component's source file from the Salesforce DX project of PackageA. Then create a version of PackageA. Before pushing this upgrade to subscribers, test the new package version to ensure it's working properly.

### [What to Consider Before Removing Metadata Components](#)

In most cases, removing metadata components from a second-generation managed package marks the component as deprecated and doesn't hard delete the component from the subscriber org. This approach to component removal ensures that package upgrades don't disrupt a subscriber's org.

## What to Consider Before Removing Metadata Components

In most cases, removing metadata components from a second-generation managed package marks the component as deprecated and doesn't hard delete the component from the subscriber org. This approach to component removal ensures that package upgrades don't disrupt a subscriber's org.

But there's a scenario where a deprecated component can lead to a package upgrade issue. This issue only pertains to deprecated components, and no action is needed for hard deleted components.

To see which components are deprecated and which are deleted, see [Remove Metadata Components from Second-Generation Managed Packages](#).

Here's an example scenario of how a deprecated component leads to a package upgrade issue.

1. Subscriber A installs version 1.0 of a managed package.
2. A package developer removes `project__c` custom object, and creates package version 2.0.
3. Subscriber A upgrades from version 1.0 to version 2.0, and `project__c` is now marked as deprecated in their org. Any integration with `project__c` that the subscriber created continues to work.
4. The package developer continues to refine their app, and then releases several new versions.
5. During development of version 5.0, the package developer adds a component named `project__c` to the package.
6. A new subscriber, Subscriber B, successfully installs version 5.0.
7. Subscriber A tries to upgrade to version 5.0, but the installation fails because the admin at Subscriber A never deleted `project__c` from their org.
8. The package developer has two paths to unblock Subscriber A.
  - a. Ask Subscriber A to remove all references to `project__c`, and then delete the component from their org.

- b. Remove project\_\_c from the package and release a new package version.

To prevent this kind of API name collisions in your packages, here are some best practices.

### Communicate within Your Team and Company

Before you remove any metadata, assess the impact to the package and to any packages that depend on that package. If you remove metadata in one package, that action has the potential to break the functionality of a package that depends on the removed metadata. Communicate within your team and company so that other developers are aware of this change.

### Document Package Changes for Future Developers

If you internally document the major changes that your package undergoes, including the name of metadata components that were removed, you can help alert future package developers about previously used API names.

### Communicate Changes with Your Subscribers

Educate your customers about the potential impact from any components you remove. In the Release Notes for your upgraded package, list all components you've removed and notify customers of any necessary actions.

## Delete a Second-Generation Managed Package or Package Version

Use the `sf package version delete` and `sf package delete` commands to delete packages and package versions that you no longer need.

To delete a package or package version, users need the Delete Second-Generation Packages user permission. Before you delete a package, first delete all associated package versions.

Package Type	Can I delete beta packages and package versions?	Can I delete released packages and package versions?
Second-Generation Managed Packages	Yes	No
Unlocked Packages	Yes	Yes

### Considerations for Deleting a Package or Package Version

- Deletion is permanent.
- Attempts to install a deleted package version will fail.
- Before deleting, ensure that the package or package version isn't referenced as a dependency.

### Examples:

```
$ sf package delete -p "Your Package Alias"
```

```
$ sf package delete -p 0Ho...
```

```
$ sf package version delete -p "Your Package Version Alias"
```

```
$ sf package version delete -p 04t...
```

These CLI commands can't be used with first-generation managed packages or package versions. To delete a first-generation managed package, see [View Package Details](#) in the *First-Generation Managed Packaging Developer Guide*.




## Frequently Used Packaging Operations for Second-Generation Managed Packages

For a complete list of Salesforce CLI packaging commands, see: [Salesforce Command Line Reference Guide](#).

Salesforce CLI command	What it Does
<code>sf package create</code>	Creates a package. When you create a package, you specify its package type and name, among other things.
<code>sf package version create</code>	Creates a package version.
<code>sf package install</code>	Installs a package version in a scratch, sandbox, or production org.
<code>sf package uninstall</code>	Removes a package that has been installed in an org. This process deletes the metadata and data associated with the package.
<code>sf package version promote</code>	Changes the state of the package version from beta to the managed-released state.
<code>sf org create scratch</code>	Creates a scratch org.
<code>sf org open</code>	Opens an org in the browser.

## Transfer a Second-Generation Managed Package to a Different Dev Hub

You can transfer the ownership of a second-generation managed package (managed 2GP) from one Dev Hub org to another. These transfers can occur either internally between two Dev Hub orgs your company owns, or you can transfer a package externally to another Salesforce Partner or ISV. This change provides a way to sell a second-generation managed package to a different company.

 **Note:** Package transfers are only available for second-generation managed packages that have passed AppExchange security review. If your managed 2GP package hasn't passed security review, consider creating a new managed 2GP using your preferred Dev Hub.

The package transfer feature is also available to unlocked packages. Dev Hub orgs aren't used with first-generation managed packages or unmanaged packages, so this feature doesn't apply to those package types.

## Request a Package Transfer to a Different Dev Hub

Start by logging a case with Salesforce Customer Support, and provide the following details.

**Subject:** Managed 2GP Package Transfer to a different Dev Hub

**Description:**

In the description, list:

- Subscriber package ID of the package you're transferring. This ID starts with 033.  
To verify the 033 ID of your package, run the `sf package list` command with the `--verbose` flag on the source Dev Hub org.
- Dev Hub org ID for the source org.
- Dev Hub org ID for the destination org. The destination Dev Hub org can't be a Developer Edition org or a trial org.

- Namespace of the package being transferred.
- Details about whether this package transfer is internal or external.  
An external transfer occurs when you transfer a package to a Salesforce Partner or ISV who doesn't work at your company.
- Acknowledge that you've reviewed and completed the steps listed in the `Prepare to Transfer Your Package` section, including linking your namespace to the destination Dev Hub, and clearing your Apex Error Notification User.

If you're transferring more than one package, file a separate case for each package.

After your case has been reviewed and approved, someone from Salesforce Customer Support will contact you to arrange a time to initiate the package transfer.

 **Note:** For security reasons, package transfers between a Dev Hub located in Government Cloud and a Dev Hub located outside Government Cloud aren't permitted.

## Package Transfers to External Customers

If you're transferring a package to another Salesforce Partner or ISV, provide:

- The source code and config settings needed to properly set up their Salesforce DX environment.  
All config settings needed to properly set up the `sfdx-project.json` file, and a complete list of features and settings that must be specified in their scratch org definition file.
- The login credentials to the namespace org. This information is required to link the package namespace to their Dev Hub org.

## Prepare to Transfer Your Package

Here's how you can help ensure a smooth package transfer.

- Keep the namespace linked to the source Dev Hub. Before the package transfer, the [namespace must be linked](#) to both the source and destination Dev Hub orgs.
- Before the package transfer process is initiated, ensure all push upgrades or package version creation processes have completed.
- Delete package versions that are no longer needed.
- If specified, clear the package's Error Notification User using the `sf package update --error-notification-username=` command. If you're transferring the package to a Dev Hub org that you own, you can set the Error Notification User to a user in the destination Dev Hub after the package transfer is complete. Note: Specifying `--error-notification-username=` with no value after the equals sign clears any previously set username.

## During the Package Transfer Process

All push upgrades or package version creation processes must be complete before the package transfer process is initiated. Salesforce Customer Support will alert you about the date the package transfer will occur.

## After the Package Transfer Is Complete

Run `sf package list` and verify that the package is no longer associated with your Dev Hub.

If the transferred package is still visible in your CLI output, and the recipient of the package transfer indicates the package transfer succeeded, log a case with Salesforce Customer Support to remove the association of the package with your Dev Hub org.

Next, unpublish your existing AppExchange listing for this package.

## Impact of Package Transfers on Package IDs

ID Type	ID starts with	After package transfer is complete ...
Subscriber Package ID	033	This ID remains the same.
Subscriber Package Version ID	04t	This ID remains the same.
Package ID	0Ho	The transferred package receives a new and unique package ID.

## Update Your Package Project File

Before you create new packages or package versions on your Dev Hub, update your `sfdx-project.json` file and remove all references to the transferred package from the package directory and package alias sections.

If you have packages in your Dev Hub that depend on the package that you're transferring, update the package dependency section in your `sfdx-project.json` file to explicitly specify the 04t ID of the transferred package that you depend on.

For example, if you transferred `pkgA` to a different Dev Hub, and your `sfdx-project.json` file lists the package dependency like this.

```
"dependencies": [
  {
    "package": "pkgA"
    "versionNumber": "2.0.0.LATEST"
  }
]
```

Update the dependency to either specify the 04t ID of `pkgA`.

```
"dependencies": [
  {
    "package": "04tB0000000UzH5IAK"
  }
]
```

Or specify the dependency using a package alias.

```
"dependencies": [
  {
    "package": "pkgA2.0.0-1"
  }
]
"packageAliases": {
  "pkgA2.0.0-1": "04tB0000000UzH5IAK"
}
]
```

## What Package History Is Transferred?

When a package is transferred, all package versions, and all lines of ancestry are transferred. Customer upgrade paths aren't affected.

Regardless of whether the package transfer occurred between two Dev Hub orgs you own, or the package was transferred externally to a Dev Hub you don't own, we transfer the package version history.

We transfer:

- Package name, namespace, type, and IDs. One exception is that the transferred package gets a new 0Ho ID.
- Package version info. This includes all the info that is typically displayed when you run the `sf package version list` or `sf package version report` command.

We don't transfer:

- Push upgrade history.
- Package version create requests.
- The username of the Dev Hub user who received Apex and other types of error notifications. This optional user is set using `--error-notification-username`.
- Deleted package versions.


#### [Take Ownership of a Second-Generation Managed Package Transferred from a Different Dev Hub](#)

You can take ownership of a second-generation managed package that is transferred from another Dev Hub org.

## Take Ownership of a Second-Generation Managed Package Transferred from a Different Dev Hub

You can take ownership of a second-generation managed package that is transferred from another Dev Hub org.

To initiate a package transfer from your Dev Hub org, see [Transfer a Second-Generation Managed Package to a Different Dev Hub](#).

 **Note:** For security reasons, package transfers between a Dev Hub located in Government Cloud and a Dev Hub located outside Government Cloud aren't permitted.

## Transfers from External Customers

If you're receiving the package from another Salesforce Partner or ISV, make sure they provide the source code for the package, and an outline for the config settings needed to properly set up your Salesforce DX environment.

Request all the configuration settings required to properly set up the `sfdx-project.json` file, and a complete list of features and settings that must be specified in your scratch org definition file.

Also ensure that the company who is transferring the ownership of the package provides the login credentials for the namespace org they used. This information is needed to link the package namespace to your Dev Hub org.

## Receive a Package Transfer

For internal transfers, skip this step. Only log the case described in [Transfer a Second-Generation Managed Package to a Different Dev Hub](#).

If you're receiving a package from a different Salesforce Partner or ISV, start by linking the namespace of the package you are receiving to your Dev Hub org. See [Link a Namespace to a Dev Hub Org](#) in the *Salesforce DX Developer Guide*.

Next, log a case with Salesforce Customer Support, and provide the:

- Dev Hub org ID for the source org.
- Subscriber package ID of the package you're receiving. This ID begins with 033.
- Dev Hub org ID for the destination org.

## After the Package Transfer Is Complete

After the package transfer is complete, you'll be notified by Salesforce Customer Support.

To verify that the transferred package is associated with your Dev Hub, run `sf package list`.

## Impact of Package Transfers on Package IDs

ID Type	ID starts with	After package transfer is complete ...
Subscriber Package ID	033	This ID remains the same.
Subscriber Package Version ID	04t	This ID remains the same.
Package ID	0Ho	The transferred package receives a new and unique package ID.

## Update Your Package Project File

Open and review the contents of the `sfdx-project.json` file that you received from the original package owner.

Open and review the contents of any scratch org definition files that you received from the original package owner. Definition files help in setting up your scratch orgs during development. Use the `--definition-file` parameter to specify a definition file when you create a new package version.

If the package directories section lists additional packages that weren't transferred to you, remove those references from the `sfdx-project.json` file.

Next, review the package alias section of the `sfdx-project.json` file, and remove any references to package aliases that aren't associated with the package that was transferred.

Update the package alias of the transferred package to specify its 0Ho package ID.

## Before You Create a New Package Version

Similar to how you go about creating any new package versions, you must update the `sfdx-project.json` file, and update the version number and ancestor ID. We recommend you set the ancestor ID to HIGHEST.

To designate a Dev Hub user to receive email notifications for unhandled Apex exceptions, and install, upgrade, or uninstall failures associated with your package, run the `sf package update` command, and use the `--error-notification-username` parameter.

## What Package History Is Transferred?

Regardless of whether the package transfer occurred between two Dev Hub orgs you own, or the package was transferred externally to a Dev Hub you don't own, we transfer the package version history.

We transfer:

- Package name, namespace, type, and IDs. One exception is that the transferred package gets a new 0Ho ID.
- Package version info. This includes all the info that is typically displayed when you run the `sf package version list` or `sf package version report` command.

We don't transfer:

- Push upgrade history.
- Package version create requests.
- The username of the Dev Hub user who received Apex and other types of error notifications.
- Deleted package versions.

## Next Steps

You've verified that the package is associated with your Dev Hub, you've updated your `sfdx-project.json` file, and perhaps you've even created a new package version. Congrats! There's still a couple more items of business left to complete.

1. Register the transferred package with your License Management Org.

If this is an external transfer, log a case with Salesforce Customer Support and request provide both your LMO org ID, and the 033 package ID.

2. [Publish Your Package on AppExchange](#)

## Best Practices for Second-Generation Managed Packages

We suggest that you follow these best practices when working with second-generation managed packages.

- We recommend that you work with only one Dev Hub, and enable Dev Hub in your partner business org.
- The Dev Hub org against which you run the `sf package create` command becomes the owner of the package. If the Dev Hub org associated with a package expires or is deleted, its packages no longer work.
- Include the `--tag` option when you use the `sf package version create` and `sf package version update` commands. This option helps you keep your version control system tags in sync with specific package versions.
- Create user-friendly aliases for packaging IDs, and include those aliases in your Salesforce DX project file and when running CLI packaging commands. See: [Package IDs and Aliases for Second-Generation Managed Packages](#).
- When adding components to your package, check the product documentation for that component to ensure that the product is generally available (GA). If you choose to package a non-GA component, it may have limitations and isn't guaranteed to GA. This scenario is particularly risky if the component can't be removed from a managed package.

## Manage Licenses for Managed Packages

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	<a href="#">Get Started with the License Management App</a> on page 364
Bill subscribers or monitor license expiration	Object Permissions: Read	<a href="#">Lead and License Records in the LMA</a> on page 367

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

I need to...	Permissions	For details, see...
Convert trial subscriptions into paying customers	Object Permissions: Edit	<a href="#">Modify a License Record</a> on page 367
Customize the LMA	Object Permissions: Edit	<a href="#">Extend the LMA</a> on page 368
Provision licenses to a subscriber	Object Permissions: Edit	<a href="#">Modify a License Record</a> on page 367
Support subscribers with technical issues	Various permissions (see <a href="#">Assign Permissions to the Subscriber Support Console</a> on page 366 for details)	<a href="#">Troubleshoot Subscriber Issues</a>

 **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.

### [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.

#### 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).

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.

#### USER PERMISSIONS

To install packages:

- Download AppExchange Packages

## 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.

A single LMO can manage multiple 1GP and 2GP packages, but a package can be associated with only one LMO.

#### USER PERMISSIONS


To manage licenses in the Partner Community:

- Manage Listings



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. Provide a username and a password with a security token appended. For example, if the password is ABC and the token is 123, enter ABC123. Don't remember your token? [Reset your security token](#).  
 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.

SEE ALSO:

[Salesforce Help: Reset Your Security Token](#)

## 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:

Custom Object	Object Permissions
	Assign READ and EDIT permissions
Package	To view package records: Assign READ 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.

 **Warning:** You can't use the LMA to modify licenses provisioned through AppExchange Checkout. To modify licenses provisioned through Checkout, have your customers follow the instructions in [Add or Remove Licenses from an AppExchange Checkout Subscription](#).

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 Modified 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 <b>Does not expire</b> .
Seats	Enter the number of licensed seats, or select <b>Site License</b> 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. <ul style="list-style-type: none"> <li>• <b>Trial</b>—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.</li> <li>• <b>Active</b>—Lets the customer use your package according to the license agreement.</li> <li>• <b>Suspended</b>—Prohibits the customer from accessing your offering.</li> </ul> <p> <b>Note:</b> When your offering is uninstalled, its status is set to Uninstalled, and the license can't be edited.</p>

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 370
- [Package](#) on page 369
- [Package Version](#) on page 369

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.

Package Version Object Fields	Description
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 367 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 <code>Does not expire</code> (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 <code>Site License</code> (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.

License Custom Object Fields	Description
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: <ul style="list-style-type: none"> <li>• A customer uninstalled the package.</li> <li>• <code>Licensed Seats</code> is set to Site License.</li> </ul>

## 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.

### 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 an Apex trigger or create a flow to email a sales rep on your team before the license expires.

### 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.

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.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### USER PERMISSIONS

To install packages:

- Download AppExchange Packages

To manage licenses in the Partner Community:

- Manage Listings

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 364.
3. [Associate your packages with the new org](#) on page 364.
4. [Refresh licenses for your packages](#) on page 368.

## Troubleshoot the License Management App

If you're experiencing issues with the License Management App, review these troubleshooting tips.

### [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.

### 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.

### **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](https://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. It's also not possible to log into a scratch org using the log in to subscriber org feature.

## 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 trail 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. To prevent your session from being automatically logged out of your LMO when you log in to a subscriber org, use the org's My Domain login URL.
- 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 366.

## 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 Second-Generation Managed Packages

---

Take the License Management App (LMA) a step further by extending it with the Feature Management App (FMA).

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 package metadata, as records of custom objects in your LMO, 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 package metadata, as records of custom objects in your LMO, 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 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`

 **Note:** After a feature parameter is included and released in the package version, the data flow direction can't be changed.

## Lifecycle of a Feature Parameter

### Set Up the Feature Parameter

Start by defining your feature parameter in an XML file. [Create one XML file](#) for each feature parameter.

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 the 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 for Your Second-Generation Managed Package](#)

To create a feature parameter for a 2GP managed package, create an individual XML file. Here are details on the file naming convention, folder structure, and the attributes you use when creating feature parameters.


## 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 for Your Second-Generation Managed Package

To create a feature parameter for a 2GP managed package, create an individual XML file. Here are details on the file naming convention, folder structure, and the attributes you use when creating feature parameters.


 **Note:** Feature parameters for managed 1GP packages are created in the packaging org's UI, see [Create Feature Parameters in Your Packaging Org](#) in the *First-Generation Managed Packaging Developer Guide* for details.

A package can include up to 200 feature parameters.

### Folder Structure

Feature parameters are stored as files in your Salesforce DX project folder.

Under the root `force-app` folder, create a folder and name it `featureParameters`. Store your feature parameter files in the `featureParameters` folder. Each feature parameter you create must have its own separate file.

 **Note:** It's not possible to create feature parameters using a scratch org's user interface.

### File Naming Convention

The naming format for feature parameter files is `<name>.featureParameter<type>-meta.xml`.

The name is the API name of the feature parameter.

The type is the feature parameter type. Feature parameters can be booleans, integers, or dates.

Type	File Name Format
Boolean	<code>.featureParameterBoolean-meta.xml</code>
Date	<code>.featureParameterDate-meta.xml</code>
Integer	<code>.featureParameterInteger-meta.xml</code>

### Feature Parameter Attributes

Feature parameters include these three fields.

Field Name	Description
<code>dataflowDirection</code>	Indicates which direction this parameter is transferring data. Each feature parameter value gets transferred in one of two directions: <ul style="list-style-type: none"> <li>From your LMO to a subscriber org (LmoToSubscriber)</li> <li>From a subscriber org to your LMO (SubscriberToLmo)</li> </ul>
<code>masterLabel</code>	The label of the feature parameter. This label displays in the app.
<code>value</code>	The value of the feature parameter. Booleans, integers, and dates are all valid values. Integer values can't exceed nine digits.



**Note:** After a feature parameter is included and released in the package version, the data flow direction can't be changed.

### Examples of Feature Parameter file

#### AdvancedPricingEnabled.featureParameterBoolean-meta.xml

```
<FeatureParameterBoolean xmlns="http://soap.sforce.com/2006/04/metadata">
  <dataflowDirection>SubscriberToLmo</dataflowDirection>
  <masterLabel>Advanced Pricing Enabled</masterLabel>
  <value>true</value>
</FeatureParameterBoolean>
```

#### NumberofLedgers.featureParameterInteger-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<FeatureParameterInteger xmlns="http://soap.sforce.com/2006/04/metadata">
  <dataflowDirection>SubscriberToLmo</dataflowDirection>
  <masterLabel>Number of Ledgers</masterLabel>
  <value>7</value>
</FeatureParameterInteger>
```

**ProjectActivationDate.featureParameterDate-meta.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<FeatureParameterDate xmlns="http://soap.sforce.com/2006/04/metadata">
  <dataflowDirection>LmoToSubscriber</dataflowDirection>
  <masterLabel>Date of Activation of the Project</masterLabel>
  <value>2020-01-25</value>
</FeatureParameterDate>
```

## 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');`



## 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.

## 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`. After you've set the visibility to `Protected`, you can later update it to `Unprotected`. To change the visibility of an object, use the [CustomObject](#) Metadata API and update the `visibility` field.

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:

[Protected Components in Managed Packages](#)

[Metadata API Developer Guide: customObject](#)

[Apex Reference Guide: Feature Management Methods, changeProtection](#)

## 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.

## Get Started with AppExchange App Analytics

---

AppExchange App Analytics provides usage data about how subscribers interact with your AppExchange 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 first- and second-generation (1GP and 2GP) managed 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.

### [App Analytics Use Cases](#)

To achieve your business objectives, use App Analytics across your teams. Read this guide to understand common use cases and how to map App Analytics data to sample product features.

### [Enable App Analytics on Your Second-Generation Managed Package](#)

Activate AppExchange App Analytics on your second-generation (2GP) managed package to access AppExchange App Analytics package usage logs and subscriber snapshots. Package usage summaries are available by default.

### [Download Package Usage Logs, Package Usage Summaries, and Subscriber Snapshots](#)

To request package usage logs, monthly package usage summaries, and subscriber snapshots, use the `AppAnalyticsQueryRequest` object. Usage logs, usage summaries, and subscriber snapshots are downloadable comma-separated value (.csv) files.

### [Considerations for Custom Interactions](#)

Easily create and log custom interactions on your managed package using Apex. As subscribers interact with your package and your Apex code is executed, the custom interactions that you defined are logged. Retrieve your custom interactions in your package's AppExchange App Analytics usage logs and usage summaries.

### [AppExchange App Analytics Best Practices](#)

To plan and maximize your AppExchange App Analytics query strategy, follow our best practices. First, use file compression to reduce your data results file size. Second, schedule and automate your regular App Analytics queries. Third, plan, schedule, and automate catch-up queries to supplement your regular query data.

### [Package Usage Summaries](#)

Package usage summaries provide high-level metrics by calendar month. Discover how many users access your package and which operations they perform.

### [Package Usage Logs](#)

Analyze adoption and user behavior, then make informed feature development decisions based on data from package usage logs. AppExchange App Analytics tracks UI, API-based, Lightning-based, and Apex operations, and it logs each CRUD operation on components and custom objects in packages. Events from sandbox and trial orgs are tracked in package usage logs. Events from scratch orgs aren't tracked.

### [Subscriber Snapshots](#)

Subscriber snapshots give you a point-in-time summary of subscriber activity. Use subscriber snapshots to see usage trends by org and package.

### [Test Custom Integrations](#)

To test your custom integrations in a nonproduction environment, use AppExchange App Analytics Simulation Mode. Submit an App Analytics query request and receive sample usage data.

### [AppExchange App Analytics Developer Cookbook](#)

Delve deeper into your AppExchange App Analytics managed package usage data by creating key performance indicators (KPIs). First, complete some prerequisites and retrieve your App Analytics data. Next, prepare your CRM Analytics environment. Finally, to build your KPIs, complete App Analytics recipes.

## App Analytics Use Cases

To achieve your business objectives, use App Analytics across your teams. Read this guide to understand common use cases and how to map App Analytics data to sample product features.

### App Analytics Use Cases

While there are various use cases for App Analytics, these cases tend to be the most common.

Partner User	Goal	How App Analytics Helps
Presales Engineer or Account Executive	Provide a great customer trial experience and close deals	<p>App Analytics provides detailed package usage logs for sandboxes and trial orgs.</p> <ul style="list-style-type: none"> <li>• Use log data to know how many users are trialing your package and which features they're actively testing.</li> <li>• Analyze log data to provide customized recommendations to your prospective subscribers.</li> <li>• Help prospective subscribers try more features or further configure the trial experience to demonstrate how your solutions can address their use cases.</li> </ul>
Customer Success Representative	Drive feature adoption and minimize subscriber attrition	<p>App Analytics provides package usage logs, package usage summaries, and subscriber snapshots in production for subscribers.</p> <ul style="list-style-type: none"> <li>• Use comprehensive usage data from all three data types to know how and when the various users for a subscriber are employing your solutions.</li> <li>• Provide tailored recommendations to drive feature adoption, identify upsell opportunities, and forecast attrition risk.</li> </ul>

#### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

Partner User	Goal	How App Analytics Helps
		<ul style="list-style-type: none"> <li>Track a user's activity across multiple packages to help you determine where opportunities and risks lie within a particular package.</li> <li>Combine your App Analytics data with license usage data from the License Management App to get a better picture of how your subscriber is adopting your product.</li> </ul>
Product Manager	Obtain product insights and drive roadmap prioritization	<p>App Analytics provides you with usage data from your entire subscriber base so that you can carry out high-level and detailed user-level analysis.</p> <p>As a product manager, you can analyze usage data to:</p> <ul style="list-style-type: none"> <li>Identify your most-used features.</li> <li>Identify incomplete user journeys.</li> <li>Identify user pain points.</li> <li>Identify unexpected usage patterns or validate expected behavior.</li> <li>Retire rarely used or end-of-life features.</li> </ul> <p>These insights help you design and build product improvements and fixes. Then you can prioritize and adjust your product roadmap accordingly.</p>
Software Engineer	Optimize code	<p>App Analytics provides usage data on Apex. Use your data on its own or combine it with more data to optimize your code, making it more performant and reliable.</p> <p>Consider using your Apex data in combination with these products.</p> <ul style="list-style-type: none"> <li>Use the Subscriber Support Console to troubleshoot subscriber issues and access logs as code is executed in real time.</li> <li>Use the Salesforce Code Analyzer to identify and fix problems in your code while you develop.</li> </ul>

There are other use cases where App Analytics isn't a good fit. For example, we don't recommend that you use App Analytics to audit customer license usage based on the `user_id_token` in package usage logs. We provide usage data for users licensed to use your package, for users who indirectly interact with it, and for automated processes.

## Mapping App Analytics Data to Product Features

For the most common App Analytics use cases, analyze App Analytics usage data at a feature level. Feature-level analysis supports conversations about those features that you have with subscribers and with your teams.

App Analytics data is organized around the concept of a `custom_entity`, which is the developer name of the components that are included in your managed package. `custom_entity` information is included in package usage summaries, package usage logs, and subscriber snapshots.



**Example:** Imagine that you want to understand how subscribers are using a new feature in your solution that enables them to easily manage newsletter subscriptions from Salesforce. To build this feature, your developers add these components to your managed package.

- A new custom object, `Newsletter_Subscription`
- A new Lightning Page, `SubscriptionPage`
- A new Lightning Component, `SubscriptionComponent`
- A new Apex Class, `SubscriptionHandler`

As subscribers interact with your components, interaction data flows through in App Analytics.

Component	Package Usage Log (Daily)	Package Usage Summary (Monthly)	Subscriber Snapshot (Daily)
<code>Newsletter_Subscription</code>	Create Read Update and Delete (CRUD) events	CRUD events	Record Counts
<code>SubscriptionPage</code>	Lightning interactions	Lightning interactions	—
<code>SubscriptionComponent</code>	Lightning component interactions	Lightning component interactions	—
<code>SubscriptionHandler</code>	Apex executions	Apex executions	—

The volume of total App Analytics data from your feature's data mixed with data for your entire solution across all subscribers can be vast. To make it easier for you to analyze, employ one of these strategies.

- Select a single component that best represents usage for this feature, and look solely at the data where it appears under `custom_entity`. In this example, the custom object `Newsletter_Subscription` is a good candidate because it tracks CRUD events from all sources, not only from the other components.
- Select a combination of components for a user journey that you care about. Using our example, select an interaction for `SubscriptionPage`, followed by `SubscriptionComponent`, `SubscriptionHandler` and CRUD for `Newsletter_Subscription`.

Package usage logs and subscriber snapshots are updated daily so that you can track subscriber usage more closely and more frequently. Package usage summaries are updated monthly. To understand how we gather and make this data available to you, read [How Does AppExchange App Analytics Data Flow?](#)

## SEE ALSO:

[How to Read App Analytics Package Usage Log Data](#)

[Customer Success Recipes](#)

[Troubleshoot Subscriber Issues](#)

## Enable App Analytics on Your Second-Generation Managed Package

Activate AppExchange App Analytics on your second-generation (2GP) managed package to access AppExchange App Analytics package usage logs and subscriber snapshots. Package usage summaries are available by default.

To ensure that you're running the latest version of Salesforce CLI and its plug-ins, run `sf update` and `sf plugins update`.

1. Activate App Analytics on your managed 2GP package. `sf package update --package "Your Package Alias" --enable-app-analytics`

To deactivate App Analytics on your managed 2GP package, run this CLI command. `sf package update --package "Your Package Alias" --no-enable-app-analytics`

2. For any additional package that you want App Analytics data for, repeat step 1.

## Download Package Usage Logs, Package Usage Summaries, and Subscriber Snapshots

To request package usage logs, monthly package usage summaries, and subscriber snapshots, use the `AppAnalyticsQueryRequest` object. Usage logs, usage summaries, and subscriber snapshots are downloadable comma-separated value (.csv) files.

To enable App Analytics on your second-generation (2GP) managed packages, follow these [instructions](#). To enable App Analytics on your first-generation (1GP) managed packages, follow these [instructions](#).

Then determine which team members need create, read, update, and delete (CRUD) access to the `AppAnalyticsQueryRequest` object, and consider [creating a permission set](#) for them. By default, admins have the permissions required to request package usage logs and usage summaries using the `AppAnalyticsQueryRequest` object.

In a 24-hour period, you can download up to 20 GB of AppExchange App Analytics data.

Package usage summary data is available to download for 10 years from the summary file log date. Package usage log data is available to download for 45 days from the date that the log event occurred. Subscriber snapshot data is available to download for 45 days from the snapshot date.

The usage data that AppExchange App Analytics collects depends on the org type and data type.

### 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

**Table 3: Data Type Collection Varies by Org Type**

Data Type	Data is Collected on...	Data isn't Collected on...
PackageUsageLog	Production, sandbox, and trial orgs	Scratch orgs
PackageUsageSummary	Production orgs	Sandbox, scratch, and trial orgs
SubscriberSnapshot	Production org and trial orgs	Sandbox and scratch orgs

 **Note:** AppExchange App Analytics is subject to certain usage restrictions as described in [AppExchange Program Policies](#).

1. Log in to the License Management Org (LMO) that the package is registered to.
2. From the LMO, complete the required fields in the [AppAnalyticsQueryRequest](#) object.
3. Retrieve the App Analytics Query Request object created in the API request. The `DownloadURL` field populates after the request is completed.
4. Click the URL in the `DownloadURL` field in the App Analytics Query Request object, and download the .csv file.

 **Note:** The download URL expires after 60 minutes.

## Considerations for Custom Interactions

Easily create and log custom interactions on your managed package using Apex. As subscribers interact with your package and your Apex code is executed, the custom interactions that you defined are logged. Retrieve your custom interactions in your package's AppExchange App Analytics usage logs and usage summaries.

As an ISV partner, the complex features that you develop in your managed packages could involve multiple actions on different objects, callouts to Apex functions, and much more. It can be difficult to interpret how your subscribers interacted with specific packaged components via your downloaded App Analytics package usage logs and summaries.

To provide you with more clarity about your subscribers' events in custom ways and at different granularity levels, create custom interactions in your managed packages using Apex.

With Apex custom interactions, you can discover:

- Which app feature a user interacted with
- How users flowed through a specific user journey
- Which UI components a user interacted with

Keep these considerations in mind:

- A custom interaction can appear for a given user request up to 50 times. This limit avoids flooding the logs due to large loops.
- We recommend that you don't call `IsvPartners.AppAnalytics.logCustomInteraction` from inside a loop.
- If the `IsvPartners.AppAnalytics.logCustomInteraction` method is called from a running Apex test, no AppExchange App Analytics package usage log or package usage summary data is produced.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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



### [Log Custom Interactions](#)

Create and log custom interactions with your managed package using Apex.

#### SEE ALSO:

[Apex Developer Guide: Enums](#)

[Download Package Usage Logs, Package Usage Summaries, and Subscriber Snapshots](#)

[Apex Reference Guide: IsvPartners Namespace](#)

[Custom Interactions](#)

## Log Custom Interactions

Create and log custom interactions with your managed package using Apex.

1. In your packaged Apex code, include Apex enums that are associated with the events that you want to log as custom interactions.
2. In your Apex code, invoke `IsvPartners.AppAnalytics.logCustomInteraction`, using the enums that you created.
3. Test your code by running it in your development environment and checking your debug logs to be certain that the custom interactions you created are being logged. Ensure that your debug log levels for `Apex Code` are set to `FINE`.
4. After you're finished with your implementation, publish a new version of your managed package.
5. After subscribers install your package, retrieve your package usage logs and package usage summaries. Filter your package usage log data on `custom_entity_type` by `CustomInteractionLabel`, and on `log_record_type` by `CustomInteraction`. Or filter your package usage summary data on `custom_entity_type` by `CustomInteractionLabel`.
6. Analyze your custom interaction data.



**Example:** Let's suppose you have a Lightning Web Component (LWC). Your LWC provides a list of related contacts for each Account record, uses a table layout, and is wired to an Apex class. You add a new card layout to your LWC. To track how well users are adopting this new layout, you log an interaction when a user switches between

Change data view

Table Card

Name	Title	Email	Phone
Arthur Song	CEO		
Edna Frank	VP, Technology		
Jack Rogers	VP, Facilities		

layouts.

In your code, include Apex enums and invoke `IsvPartners.AppAnalytics.logCustomInteraction`.

Your LWC HTML code:

```
<template>
  <div
    class="slds-var-m-top_medium slds-var-m-bottom_x-large slds-box
slds-theme_default"
  >
    <h2 class="slds-text-heading_medium slds-var-m-bottom_medium">
      Change data view
    </h2>
    <!-- Button group: simple buttons -->
    <lightning-button-group class="slds-var-m-bottom_medium">
      <lightning-button
        label="Table"
        variant={tableVariant}
        onclick={handleClick}
      ></lightning-button>
    </lightning-button-group>
  </div>
```

```

    <lightning-button
      label="Card"
      variant={cardVariant}
      onclick={handleClick}
    ></lightning-button>
  </lightning-button-group>
  <template lwc:if={displayTable}>
    <lightning-datatable
      key-field="id"
      data={records}
      columns={columns}
    ></lightning-datatable>
  </template>
  <template lwc:if={displayCard}>
    <div class="slds-grid slds-wrap slds-grid_pull-padded-small">
      <template for:each={records} for:item="contact">
        <div
          class="slds-col slds-small-size_1-of-1 slds-large-size_1-of-2
slds-var-p_small"
          key={contact.id}
        >
          <lightning-card
            variant="Narrow"
            title={contact.name}
            icon-name="standard:contact"
          >
            <div class="slds-var-p-horizontal_small">
              <p>{contact.name}</p>
              <p>{contact.title}</p>
              <p>
                <lightning-formatted-phone
                  value={contact.phone}
                ></lightning-formatted-phone>
              </p>
              <p>
                <lightning-formatted-email
                  value={contact.email}
                ></lightning-formatted-email>
              </p>
            </div>
          </lightning-card>
        </div>
      </template>
    </div>
  </template>
</div>
</template>

```

Your LWC JavaScript code:

```

import { LightningElement, wire, api } from "lwc";
import { getRelatedListRecords } from "lightning/uiRelatedListApi";
import logInteraction from "@salesforce/apex/LogContactListInteraction.log";

export default class ContactList extends LightningElement {

```

```

@api recordId;
error;
records;
displayTable = true;
displayCard = false;
columns = [
  { label: "Name", fieldName: "name" },
  { label: "Title", fieldName: "title" },
  { label: "Email", fieldName: "email", type: "email" },
  { label: "Phone", fieldName: "phone", type: "phone" }
];
@wire(getRelatedListRecords, {
  parentRecordId: "$recordId",
  relatedListId: "Contacts",
  fields: [
    "Contact.Name",
    "Contact.Id",
    "Contact.Phone",
    "Contact.Email",
    "Contact.Title"
  ],
  sortBy: ["Contact.Name"]
})
contactList({ error, data }) {
  if (data) {
    this.records = data.records.map((item) => {
      return {
        name: item.fields.Name.value,
        id: item.fields.Id.value,
        title: item.fields.Title.value,
        email: item.fields.Email.value,
        phone: item.fields.Phone.value
      };
    });
    this.error = undefined;
  } else if (error) {
    this.error = error;
    this.records = undefined;
  }
}

handleClick(event) {
  if (event.target.label.toLowerCase() === "table") {
    this.displayTable = true;
    this.displayCard = false;
    logInteraction({ type: "table" });
  } else if (event.target.label.toLowerCase() === "card") {
    this.displayTable = false;
    this.displayCard = true;
    logInteraction({ type: "card" });
  }
}

get cardVariant() {
  return this.displayCard === true ? "brand" : "";
}

```

```

    }
    get tableVariant() {
        return this.displayTable === true ? "brand" : "";
    }
}

```

Your Apex class:

```

public class LogContactListInteraction {
    public Enum ContactListLayouts { TABLE, CARD }

    @AuraEnabled
    public static void log(String type) {
        try {
            IsvPartners.AppAnalytics.logCustomInteraction(getInteractionLabel(type));

        } catch (Exception e) {
            throw new AuraHandledException(e.getMessage());
        }
    }

    private static ContactListLayouts getInteractionLabel(String type) {
        if (type.toLowerCase() == 'table') {
            return ContactListLayouts.TABLE;
        } else if (type.toLowerCase() == 'card') {
            return ContactListLayouts.CARD;
        }
        return null;
    }
}

```

Next, you test your code. With your Apex code debug log level set to FINE, confirm that the custom interactions are logged by finding events in your debug logs called APP\_ANALYTICS\_FINE, APP\_ANALYTICS\_WARN, or APP\_ANALYTICS\_ERROR.

```

APP_ANALYTICS_FINE [External]IsvPartners.AppAnalytics.logCustomInteraction was called,
but not from an installed managed package.
This means that the code is ready to be packaged.

```

SEE ALSO:

[Package Usage Logs Schema](#)

[Considerations for Custom Interactions](#)

## AppExchange App Analytics Best Practices

To plan and maximize your AppExchange App Analytics query strategy, follow our best practices. First, use file compression to reduce your data results file size. Second, schedule and automate your regular App Analytics queries. Third, plan, schedule, and automate catch-up queries to supplement your regular query data.

### [How Does AppExchange App Analytics Data Flow?](#)

As your customers use your managed packages, they produce data. Their usage data is collected daily in our data lake from each Salesforce instance. Usage data arrives to our data lake throughout the day. From time to time, there can be data arrival delays. Also, data builds and timestamps vary by data type. For these reasons, to optimize your data retrieval, plan out your AppExchange App Analytics query strategy.

### [How Should I Plan My App Analytics Query Strategy?](#)

Your detailed query strategy depends on the size and scope of your business and the data types that you're querying.

### [Recommendations](#)

Your query strategy varies based on your business size and scope. Also, your query strategy must adapt as your business grows. To stay current, follow our App Analytics query recommendations for small, medium, and large-sized partners.

### [Where Do I Go for More Information About AppExchange App Analytics Queries?](#)

Questions are natural when you start automating your queries and planning your query strategy. To find a good solution when you have questions, review your code base and the size and skill of your development team.

## How Does AppExchange App Analytics Data Flow?

As your customers use your managed packages, they produce data. Their usage data is collected daily in our data lake from each Salesforce instance. Usage data arrives to our data lake throughout the day. From time to time, there can be data arrival delays. Also, data builds and timestamps vary by data type. For these reasons, to optimize your data retrieval, plan out your AppExchange App Analytics query strategy.

Because Salesforce instances and your subscribers are located around the world, the time of data collection varies by region. EU (EMEA) data arrives first, then North America (NA) data. Data from Asia Pacific (AP) instances arrives last.

Our AppExchange App Analytics jobs run on local instance times on a non-peak schedule. Depending on when you query for your data and where your customers are located, sometimes you retrieve 100% of your data at one time. Other times you must issue more queries to retrieve it all.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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



Data delivery to and arrival in our data lake also depends on factors that can affect a given instance, such as the health of the instance or technical dependencies. Ordinarily you can expect all your org data to arrive in the data lake by 23 : 00 Coordinated Universal Time (UTC) the day after it was recorded. However, occasionally, there can be delays.

Each AppExchange App Analytics data type is also compiled at different times.

Data Type	Build Information	Example
Subscriber Snapshots	<ul style="list-style-type: none"> <li>• Snapshots use data collected at approximately 01 : 00 instance local time.</li> <li>• Snapshots are generated nightly at approximately 03 : 00 instance local time.</li> <li>• All timestamps are normalized to 00 : 00 UTC of that day.</li> </ul>	For the March 1, 2021 snapshot: <ul style="list-style-type: none"> <li>• All records have this timestamp: 2021-03-01T00 : 00 : 00Z.</li> <li>• All data normally arrives by March 2, 2021 23 : 00 UTC .</li> </ul>
Package Usage Summaries	<ul style="list-style-type: none"> <li>• Summaries use data collected for an entire month.</li> <li>• Summaries are built monthly.</li> <li>• All timestamps are normalized to 00 : 00 UTC on the last day of the month.</li> </ul>	For the March 2021 summary available on April 1, 2021: <ul style="list-style-type: none"> <li>• All records have this timestamp: 2021-03-31T00 : 00 : 00Z.</li> <li>• All data normally arrives by April 1, 2021 23 : 00 UTC.</li> <li>• In this example, we recommend that you query for your summary data on April 3, 2021 or later. We recommend a 2-day query delay to ensure that all of your subscriber data from all worldwide instances finished processing.</li> </ul>

Data Type	Build Information	Example
Package Usage Logs	<ul style="list-style-type: none"> <li>Usage logs use data from the previous day.</li> <li>Usage logs are generated nightly at approximately 05:00 instance local time.</li> </ul>	For the March 1, 2021 log file: <ul style="list-style-type: none"> <li>All records have precise timestamps associated with when that log event occurred.</li> <li>All data normally arrives by March 2, 2021 23:00 UTC.</li> </ul>

## How Should I Plan My App Analytics Query Strategy?

Your detailed query strategy depends on the size and scope of your business and the data types that you’re querying.

All partners can take advantage of these query strategies.

- Choose a data results `FileType` value, and select a corresponding `FileCompression`. With this query strategy, you can choose `gzip` compression for `csv` files or `snappy` column compression for `parquet` files.
- Create regularly scheduled, automated queries.
- To sweep in late-arriving data, create catch-up queries using the `AvailableSince` field.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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


## Compress Your Results Files

Your App Analytics query plan starts with your results file type and file compression. Data can eat up time and space, so do more with less by specifying the type of file you download. Reduce your data download time by specifying how your results file is compressed.

If you don’t specify file type or file compression, your results file defaults to `csv` with no compression for backwards compatibility reasons. If you choose the `parquet` file type, your results file includes data type information for each column.

We recommend that you always compress your results files. Choose from these SOAP API `AppAnalyticsQueryRequest` `FileType` and `FileCompression` value combinations.

FileType	FileCompression
<code>csv</code> (default)	<ul style="list-style-type: none"> <li><code>none</code> (default when <code>FileType</code> is <code>csv</code>)</li> <li><code>gzip</code></li> </ul>
<code>parquet</code>	<ul style="list-style-type: none"> <li><code>snappy</code> (default when <code>FileType</code> is <code>parquet</code>)</li> <li><code>gzip</code></li> <li><code>none</code></li> </ul>

 **Note:** When you download your App Analytics query result data, the HTTP response contains one or two important headers. The `Content-Type` header indicates the file type (`txt/csv` or `application/parquet`). For queries with `csv` `FileType` and `gzip` `FileCompression`, the `Content-Encoding` header indicates `gzip` encoding. Modern browsers often decode the `gzip`-encoded file automatically, which results in a saved, uncompressed `.csv` file. Regardless if the file is automatically decoded or not, its filename extension is `.csv`.

## Schedule and Automate Your Queries

After you determine what queries to run and how often to run them, you want to schedule those queries. The easiest way is via automation.

What do we mean by automation? Write code that creates query request records on your schedule, monitors them, retrieves the data, and stores your AppExchange App Analytics data somewhere. For example, you can store the data in a custom object in your License Management Org.

Your automation options include, but aren't limited to:

- Custom API integrations using REST or SOAP API calls
- Salesforce DX automation using the CLI
- Salesforce flows
- Apex triggers

For example, automate the retrieval of package usage summaries using Apex triggers.

If you want to also automate the retrieval of package usage log data, look to a different storage solution that scales with the data volume the logs contain.

## Create Catch-Up Queries

A catch-up query is like a broom, sweeping for data newly added to our data lake. Catch-up queries rely on you already having regular queries in place.

For example, on March 2, 2021 18:00 UTC you run this regular query that retrieves package usage log data for March 1, 2021:

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-01T00:00:00Z
EndTime=2021-03-02T00:00:00Z
DataType=PackageUsageLog
FileType=csv
FileCompression=gzip"
```

Rerun that exact same query on March 3, 2021 18:00 UTC, but add the `AvailableSince` field set to the day and time you ran your original query: `2021-03-02T18:00:00Z`. This query is your ad hoc catch-up query. It retrieves any data newly added to the data lake for March 2 since you ran your regular query:


```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-01T00:00:00Z
EndTime=2021-03-02T00:00:00Z
DataType=PackageUsageLog
FileType=csv
FileCompression=gzip
AvailableSince=2021-03-02T18:00:00Z"
```

You can use catch-up queries in many different ways, which we discuss in more detail in the Recommendations section.

When creating catch-up queries, keep these considerations in mind.

- If `StartTime` is specified, the `AvailableSince` date must be later.
- If `EndTime` is specified, the `AvailableSince` date must be later.
- All queries must include `StartTime` or `AvailableSince` or both.
- `AvailableSince` must be earlier than now.



 **Note:** What happens when you want to create an ad hoc catch-up query, but you forgot when you ran the original query? Use Salesforce CLI and your original query's `sObjectID` to look up the `QuerySubmittedTime`, like this: `sf data get record --subjecttype AppAnalyticsQueryRequest --subjectid 0XIXXXXXXXXXXXXXXXXXX` Set your ad hoc catch-up query `AvailableSince` value to equal the `QuerySubmittedTime`.


SEE ALSO:

[Apache Parquet](#)

[Automate AppAnalytics - AWS Stack](#)

## Recommendations

Your query strategy varies based on your business size and scope. Also, your query strategy must adapt as your business grows. To stay current, follow our App Analytics query recommendations for small, medium, and large-sized partners.

 **Note:** In the unlikely event of data delays, we regenerate data for log events that happened up to 30 days in the past. To ensure that you consistently retrieve the most complete data, we recommend that you schedule catch-up queries that look back 30 days.

### Small-Sized Partners

Small-sized partners have manageable subscriber bases and one or two managed packages. A small partner's total daily usage data across all managed packages is 5 GB or less. Also, small partner's queries complete well under the 15-minute processing time limit.

### Medium-Sized Partners

Medium-sized partners have bigger subscriber bases and about six managed packages. A medium-sized partner's total daily usage data across all managed packages is at or just over 20 GB. Also, this partner's queries approach or hit the 15-minute processing time limit.

### Large-Sized Partners

Large partners have large subscriber bases and many managed packages. A large partner's total daily data usage is more than 20 GB. Sometimes a large partner's data from just one managed package is larger than the 20-GB daily limit. Also, large partners often must create a smaller time range for each query to complete in under the 15-minute processing time limit.

## Small-Sized Partners

Small-sized partners have manageable subscriber bases and one or two managed packages. A small partner's total daily usage data across all managed packages is 5 GB or less. Also, small partner's queries complete well under the 15-minute processing time limit.

Given how manageable smaller partners' data is, after you run your regular queries one time, we recommend that you run a daily catch-up query as your main query. Sweep in all data for all your managed packages for the last 30 days.

Data Type	How to Get Started	How to Schedule Catch-Up Queries
Subscriber Snapshots	An initial query to retrieve data from when App Analytics was	<ul style="list-style-type: none"> <li>One daily catch-up query.</li> </ul>

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

Data Type	How to Get Started	How to Schedule Catch-Up Queries
	enabled for your managed package.	<ul style="list-style-type: none"> <li>• Set <code>AvailableSince</code> to the day and time your last regular query ran.</li> <li>• Set <code>StartTime</code> to 30 days ago.</li> <li>• Omit <code>EndTime</code>.</li> <li>• Each day advance <code>StartTime</code> and <code>AvailableSince</code> by 1 day.</li> </ul>
Package Usage Summaries	An initial query to retrieve data from when App Analytics was enabled for your managed package.	<ul style="list-style-type: none"> <li>• One daily catch-up query.</li> <li>• Set <code>AvailableSince</code> to the day and time your last regular query ran.</li> <li>• Set <code>StartTime</code> to the first of the previous month.</li> <li>• Omit <code>EndTime</code>.</li> <li>• Each day advance <code>AvailableSince</code> by 1 day.</li> <li>• Each month advance <code>StartTime</code> to the first of the previous month.</li> </ul>
Package Usage Logs	An initial query to retrieve data from when App Analytics was enabled for your managed package.	<ul style="list-style-type: none"> <li>• One daily catch-up query.</li> <li>• Set <code>AvailableSince</code> to the day and time your last regular query ran.</li> <li>• Set <code>StartTime</code> to 30 days ago.</li> <li>• Omit <code>EndTime</code>.</li> <li>• Each day advance <code>StartTime</code> and <code>AvailableSince</code> by 1 day.</li> </ul>



**Example:** Most of your customers use your package on an NA or EU instance, so you run your queries at 18:00 UTC. You have a couple customers on an AP instance, so you create catch-up queries to ensure that you capture data from around the world.

- On March 31 at 18:00 UTC, run your regular queries.

Subscriber Snapshot

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "DataType=SubscriberSnapshot
FileType=csv
FileCompression=gzip
StartTime=2020-03-30T00:00:00Z
EndTime=2020-03-31T00:00:00Z"
```

Package Usage Summary

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
```

```
--values "DataType=PackageUsageSummary
FileType=csv
FileCompression=gzip
StartTime=2020-02-01T00:00:00Z
EndTime=2020-03-01T00:00:00Z"
```

#### Package Usage Log

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "DataType=PackageUsageLog
FileType=csv
FileCompression=gzip
StartTime=2020-03-30T00:00:00Z
EndTime=2020-03-31T00:00:00Z"
```

- On April 1 at 18:00 UTC run these three catch-up queries.

#### Subscriber Snapshot Catch-Up Query

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "DataType=SubscriberSnapshot
FileType=csv
FileCompression=gzip
StartTime=2020-03-02T00:00:00Z
AvailableSince=2020-03-31T18:00:00Z"
```

#### Package Usage Summary Catch-Up Query

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "DataType=PackageUsageSummary
FileType=csv
FileCompression=gzip
StartTime=2020-03-01T00:00:00Z
AvailableSince=2020-03-31T18:00:00Z"
```

#### Package Usage Log Catch-Up Query

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "DataType=PackageUsageLog
FileType=csv
FileCompression=gzip
StartTime=2020-03-02T00:00:00Z
AvailableSince=2020-03-31T18:00:00Z"
```

- On April 2 at 18:00 UTC, run the same catch-up queries, but advance the subscriber snapshot and package usage log AvailableSince and StartTime date by 1 day each. Advance the package usage summary AvailableSince by 1 day.

#### Subscriber Snapshot Catch-Up Query

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "DataType=SubscriberSnapshot
```

```
FileType=csv
FileCompression=gzip
StartTime=2020-03-03T00:00:00Z
AvailableSince=2020-04-01T18:00:00Z"
```

Package Usage Summary Catch-Up Query

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "DataType=PackageUsageSummary
FileType=csv
FileCompression=gzip
StartTime=2020-03-01T00:00:00Z
AvailableSince=2020-04-01T18:00:00Z"
```

Package Usage Log Catch-Up Query

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "DataType=PackageUsageLog
FileType=csv
FileCompression=gzip
StartTime=2020-03-03T00:00:00Z
AvailableSince=2020-04-01T18:00:00Z"
```

### Medium-Sized Partners

Medium-sized partners have bigger subscriber bases and about six managed packages. A medium-sized partner’s total daily usage data across all managed packages is at or just over 20 GB. Also, this partner’s queries approach or hit the 15-minute processing time limit.

We recommend that after you run your regular queries one time, use catch-up queries as your main queries for subscriber snapshots and package usage summaries. Use a combination of daily queries and catch-up queries for package usage logs.

#### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

Data Type	How to Get Started	How to Schedule Catch-Up Queries
Subscriber Snapshots	An initial query to retrieve data from when App Analytics was enabled for your managed packages.	<ul style="list-style-type: none"> <li>• One daily query.</li> <li>• Set <code>AvailableSince</code> to the day and time your last regular query ran.</li> <li>• Set <code>StartTime</code> to 30 days ago.</li> <li>• Omit <code>EndTime</code>.</li> <li>• Each day advance <code>StartTime</code> and <code>AvailableSince</code> by 1 day.</li> </ul>

Data Type	How to Get Started	How to Schedule Catch-Up Queries
Package Usage Summaries	An initial query to retrieve data from when App Analytics was enabled for your managed packages.	<ul style="list-style-type: none"> <li>• One daily catch-up query.</li> <li>• Set <code>AvailableSince</code> to the day and time your last regular query ran.</li> <li>• Set <code>StartTime</code> to the first of the previous month.</li> <li>• Omit <code>EndTime</code>.</li> <li>• Each day advance <code>AvailableSince</code> by 1 day.</li> <li>• Each month advance <code>StartTime</code> to the first of the previous month.</li> </ul>
Package Usage Logs	One regular daily query per package.	<ul style="list-style-type: none"> <li>• One daily catch-up query per package.</li> <li>• Set <code>AvailableSince</code> to the day and time your last regular query ran.</li> <li>• Set <code>StartTime</code> to 30 days ago.</li> <li>• Set <code>EndTime</code> equal to the <code>StartTime</code> of your regular query.</li> <li>• Each day advance <code>StartTime</code>, <code>EndTime</code>, and <code>AvailableSince</code> by 1 day.</li> </ul>



**Example:** Half of your customers use your package on an NA or EU instance, so you run your regular queries at 18:00 UTC. The other half of your customers are on an AP instance, so you create catch-up queries to ensure that you capture data from around the world.

- On March 31 at 18:00 UTC, run your regular package usage log queries for each of your packages.

Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-30T00:00:00Z
EndTime=2021-03-31T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=csv
FileCompression=gzip"
```

Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-30T00:00:00Z
EndTime=2021-03-31T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX"
```

```
FileType=csv
FileCompression=gzip"
```

- On April 1 at 18:00 UTC onwards, run regular and catch-up package usage log queries.



#### A. Regular Queries

##### Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-31T00:00:00Z
EndTime=2021-04-01T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=csv
FileCompression=gzip"
```

##### Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-31T00:00:00Z
EndTime=2021-04-01T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=csv
FileCompression=gzip"
```

#### B. Catch-Up Queries

##### Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-01T00:00:00Z
EndTime=2021-03-31T00:00:00Z
AvailableSince=2021-03-31T18:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=csv
FileCompression=gzip"
```

##### Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-01T00:00:00Z
EndTime=2021-03-31T00:00:00Z
AvailableSince=2021-03-31T18:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX"
```

```
FileType=csv
FileCompression=gzip"
```

- On April 2, repeat the same queries that you ran on April 1, but advance the queries by a day.



A. Regular Queries

Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-04-01T00:00:00Z
EndTime=2021-04-02T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=csv
FileCompression=gzip"
```

Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-04-01T00:00:00Z
EndTime=2021-04-02T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=csv
FileCompression=gzip"
```

B. Catch-Up Queries

Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-02T00:00:00Z
EndTime=2021-04-01T00:00:00Z
AvailableSince=2021-04-01T18:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=csv
FileCompression=gzip"
```

Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2020-03-02T00:00:00Z
EndTime=2021-04-01T00:00:00Z
AvailableSince=2021-04-01T18:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX"
```

```
FileType=csv
FileCompression=gzip"
```

## Large-Sized Partners

Large partners have large subscriber bases and many managed packages. A large partner’s total daily data usage is more than 20 GB. Sometimes a large partner’s data from just one managed package is larger than the 20-GB daily limit. Also, large partners often must create a smaller time range for each query to complete in under the 15-minute processing time limit.

Large partners frequently create one query per managed package per 12, 6, or 1-hour increments throughout a 24-hour period. How frequently you schedule your queries really depends on your data volume.

We recommend that you use a combination of queries and multiple catch-up queries for all data types

### EDITIONS


Available in: both Salesforce Classic and Lightning Experience

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

Data Type	How to Get Started	How to Schedule Catch-Up Queries
Subscriber Snapshots	One daily query per package.	<ul style="list-style-type: none"> <li>One daily query per package.</li> <li>Set <code>AvailableSince</code> to the day and time your last regular query ran.</li> <li>Set <code>StartTime</code> to 30 days ago.</li> <li>Omit <code>EndTime</code>.</li> <li>Each day advance <code>StartTime</code> and <code>AvailableSince</code> by 1 day.</li> </ul>
Package Usage Summaries	One daily query per package.	<ul style="list-style-type: none"> <li>One daily catch-up query per package.</li> <li>Set <code>AvailableSince</code> to the day and time your last regular query ran.</li> <li>Set <code>StartTime</code> to the first of the previous month.</li> <li>Omit <code>EndTime</code>.</li> <li>Each day advance <code>AvailableSince</code> by 1 day.</li> <li>Each month advance <code>StartTime</code> to the first of the previous month.</li> </ul>
Package Usage Logs	<ul style="list-style-type: none"> <li>To retrieve all your data, create multiple segmented daily, automated App Analytics queries spread throughout the day.</li> <li>Break up your requests by managed package and by time increments throughout the day.</li> </ul>	<p>Create two levels of catch-up queries per day.</p> <ul style="list-style-type: none"> <li>Create one catch-up query per package that sweeps data from 2 days ago.</li> <li>Create a second catch-up query that sweeps data from 3 to 30 days ago.</li> </ul>



Data Type	How to Get Started	How to Schedule Catch-Up Queries
		<ul style="list-style-type: none"> <li>Each day advance <code>StartTime</code>, <code>EndTime</code>, and <code>AvailableSince</code> by 1 day.</li> </ul>

 **Example:** Your customers use your package on all Salesforce instances around the world, and your managed packages produce significant amounts of data. You schedule queries to run at the same time, each covering a 12-hour period, and you create a layered catch-up query plan to capture data from all instances.

In this example, we show two of your dozens of managed packages.

- On March 31 at 18:00 UTC, run your regular package usage log queries.

Package 1

```
sf data create record data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-30T00:00:00
EndTime=2021-03-30T12:00:00
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

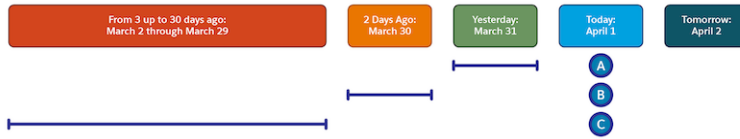
```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-30T12:00:00
EndTime=2021-03-31T00:00:00
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-30T00:00:00
EndTime=2021-03-30T12:00:00
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-30T12:00:00
EndTime=2021-03-31T00:00:00
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

- On April 1 at 18:00 UTC, run your regular and catch-up package usage log queries.



A. Package Usage Log Regular Queries

Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-31T00:00:00Z
EndTime=2021-03-31T12:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-31T12:00:00Z
EndTime=2021-04-01T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-31T00:00:00Z
EndTime=2021-03-31T12:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-31T12:00:00Z
EndTime=2021-04-01T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

B. Package Usage Log 2 Days Ago Catch-Up Queries

Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-30T00:00:00Z
```

```

EndTime=2021-03-31T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy
AvailableSince=2020-03-31T18:00:00Z"
    
```

Package 2

```

sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-30T00:00:00Z
EndTime=2021-03-31T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy
AvailableSince=2020-03-31T18:00:00Z"
    
```

C. Package Usage Log From 3 to 30 Days Ago Catch-Up Queries

Package 1

```

sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-01T00:00:00Z
EndTime=2021-03-30T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy
AvailableSince=2020-03-31T18:00:00Z"
    
```

Package 2

```

sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-01T00:00:00Z
EndTime=2021-03-30T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy
AvailableSince=2020-03-31T18:00:00Z"
    
```

- On April 2 onwards, run your regular and your catch-up package usage log queries, advancing the dates by 1 day.



A. Package Usage Log Regular Queries

## Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-04-01T00:00:00Z
EndTime=2021-04-01T12:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-04-01T12:00:00Z
EndTime=2021-04-02T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

## Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-04-01T00:00:00Z
EndTime=2021-04-01T12:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-04-01T12:00:00Z
EndTime=2021-04-02T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy"
```

## B. Package Usage Log 2 Days Ago Catch-Up Queries

## Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-31T00:00:00Z
EndTime=2021-04-01T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy
AvailableSince=2020-04-01T18:00:00Z"
```

## Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-31T00:00:00Z
EndTime=2021-04-01T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy
AvailableSince=2020-04-01T18:00:00Z"
```

## C. Package Usage Log From 3 to 30 Days Ago Catch-Up Queries

## Package 1

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-02T00:00:00Z
EndTime=2021-03-31T00:00:00Z
DataType=PackageUsageLog
PackageIds=0336XXXXXXXXXX
FileType=parquet
FileCompression=snappy
AvailableSince=2020-04-01T18:00:00Z"
```

## Package 2

```
sf data create record
--subjecttype AppAnalyticsQueryRequest
--values "StartTime=2021-03-02T00:00:00Z
EndTime=2021-03-31T00:00:00Z
DataType=PackageUsageLog
PackageIds=0337XXXXXXXXXX
FileType=parquet
FileCompression=snappy
AvailableSince=2020-04-01T18:00:00Z"
```

## Where Do I Go for More Information About AppExchange App Analytics Queries?

Questions are natural when you start automating your queries and planning your query strategy. To find a good solution when you have questions, review your code base and the size and skill of your development team.

If you still need help, try these resources:

- If you have an assigned AppExchange Partner Account Manager (PAM) or AppExchange Technical Evangelist (TE), reach out to them.
- Otherwise, go to the Partner Community and post a question to the [ISV TE Experts - Partner Intelligence](#) Chatter group.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## Package Usage Summaries

Package usage summaries provide high-level metrics by calendar month. Discover how many users access your package and which operations they perform.

 **Note:** AppExchange App Analytics is subject to certain usage restrictions as described in the [AppExchange Program Policies](#).

AppExchange App Analytics tracks UI, API-based, Lightning-based, and Apex operations and logs each CRUD operation on components and custom objects in packages. Events from sandbox, scratch, and trial orgs aren't tracked in package usage summaries.

Partners and subscribers can access package usage data. Usage summaries become available at the beginning of the subsequent month. For example, you can get the usage summary for May at the beginning of June.

- AppExchange Partners can request monthly usage summaries using the AppAnalyticsQueryRequest in SOAP API from the license management org that owns the package.
- Subscribers can download usage summaries from Setup for any package that they installed that passed security review.

### [Package Usage Summary Schema](#)

Use the package usage summary to discover how many users access your package and which operations they perform.

## Package Usage Summary Schema

Use the package usage summary to discover how many users access your package and which operations they perform.

Package usage summaries contain aggregate data derived from related [package usage logs](#). ISV partners have access to package usage summaries by default, and they can activate access to package usage logs and subscriber snapshots. Subscribers only have access to package usage summaries.

Field	Description
custom_entity	The developer name of the component or custom object.
custom_entity_type	The type of component or custom object that the user viewed or manipulated. Examples: <ul style="list-style-type: none"> <li>• AnalyticsDashboard</li> <li>• AnalyticsLens</li> <li>• ApexClass</li> <li>• ApexTrigger</li> <li>• CustomInteractionLabel</li> <li>• CustomInteractionFailure</li> <li>• CustomObject</li> <li>• ExternalObject</li> <li>• LightningPage</li> <li>• LightningComponent</li> <li>• VisualforcePage</li> </ul>
managed_package_namespace	Namespace of the package.
month	The month that this usage summary applies to in YYYY-MM format. Example: 2019-03.

Field	Description
num_creates	The number of new records created from the package.
num_deletes	The number of deleted records associated with the package.
num_events	The number of log records associated with a <code>custom_entity</code> .
num_reads	The aggregate number of records associated with the <code>custom_entity</code> that was read.  The definition of <code>num_reads</code> changed with the Spring '23 release. Some data from February 2023 and all data dated January 2023 and earlier used this previous <code>num_reads</code> definition: the aggregate number of records associated with the package that were read, plus the number of <code>SOSL</code> and <code>SOQL</code> queries performed on the entity.
num_updates	The number of records associated with the package that were updated.
num_views	The number of times the component or page has been viewed.
organization_edition	The name of the Salesforce edition that the subscriber org is using. Examples: <ul style="list-style-type: none"> <li>• Developer Edition</li> <li>• Enterprise Edition</li> <li>• Unlimited Edition</li> </ul>
organization_id	The 15-character ID of the subscriber org.
organization_name	The name of the subscriber org. Example: <code>Acme, Inc.</code>
organization_status	The paid status of the subscriber org. Examples: <ul style="list-style-type: none"> <li>• Active</li> <li>• Demo</li> <li>• Free</li> <li>• Trial</li> </ul>
package_id	The ID of the package.
user_id_token	The hashed token representing the ID of the user who accessed the package. The token persists over time, even if a user's details change. The token also persists across any packages that the user interacts with.  The user ID token starts with the prefix <code>005-</code> . In compliance with privacy regulations, our systems can't access the actual user ID. Likewise, the hashed token can't be linked to the user ID.
user_type	The user license category of the user accessing Salesforce services through the UI or API. Examples: <ul style="list-style-type: none"> <li>• Guest</li> <li>• Partner</li> </ul>

Field	Description
	<ul style="list-style-type: none"> <li>Standard</li> </ul>

SEE ALSO:

[Package Usage Logs Schema](#)

## Package Usage Logs

Analyze adoption and user behavior, then make informed feature development decisions based on data from package usage logs. AppExchange App Analytics tracks UI, API-based, Lightning-based, and Apex operations, and it logs each CRUD operation on components and custom objects in packages. Events from sandbox and trial orgs are tracked in package usage logs. Events from scratch orgs aren't tracked.

 **Note:** AppExchange App Analytics is subject to certain usage restrictions as described in the [AppExchange Program Policies](#).

### [How to Read App Analytics Package Usage Log Data](#)


App Analytics package usage logs contain data about how subscribers interact with your managed package. Your managed package contains packaged components, and each package usage log line describes an interaction that a user has with one of your packaged components. To understand that interaction, analyze each log line—or record—and focus on: what packaged component was accessed, who interacted with that packaged component, and how that packaged component interaction occurred. Finally, analyze the specific interaction data.

### [Package Usage Logs Schema](#)

Make informed development decisions based on package usage log data. Analyze adoption, user behavior, company information, and Lightning app and page usage data. Package usage logs list activity during a 24-hour period, between 12:00 AM and 11:59 PM UTC.

## How to Read App Analytics Package Usage Log Data

App Analytics package usage logs contain data about how subscribers interact with your managed package. Your managed package contains packaged components, and each package usage log line describes an interaction that a user has with one of your packaged components. To understand that interaction, analyze each log line—or record—and focus on: what packaged component was accessed, who interacted with that packaged component, and how that packaged component interaction occurred. Finally, analyze the specific interaction data.

 **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.

### [Determine What Packaged Component Was Accessed](#)

To analyze a package usage log record, always start with your packaged component.

### [Identify Who Interacted with Your Packaged Component](#)

After you identify your packaged component, identify both the subscriber org and the user who triggered the interaction.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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



### Identify How a User Interacted with Your Packaged Component

After you identify your packaged component and who interacted with it, identify how the user interacted with your packaged component.

#### Custom Object and External Object Interactions

When a log record in your package usage log has `custom_entity_type` equal to `CustomObject` or `ExternalObject`, it means that a user performed an action that resulted in a create, read, update, or delete (CRUD) interaction on your object.

#### Lightning Interactions

Each record in your package usage log that has a `custom_entity_type` of `LightningComponent` or `LightningPage` describes an interaction with your packaged Lightning component or page.

#### Apex Interactions

Each record in your package usage log that has a `custom_entity_type` of `ApexClass` or `ApexTrigger` describes an interaction with your packaged Apex class or trigger.

#### Visualforce Interactions

Each record in your package usage log that has a `custom_entity_type` of `VisualforcePage` describes an interaction with your packaged Visualforce pages.

#### CRM Analytics Asset Interactions

Each record in your package usage log that has a `custom_entity_type` of `AnalyticsDashboard`, `AnalyticsLens`, or `AnalyticsRecipe` describes an interaction with your packaged CRM Analytics assets.

#### Custom Interactions

To understand which features and UI components a subscriber interacted with and how they flow through a user journey, create custom interactions with Apex enums and the `ISVPartners.AppAnalytics.logCustomInteraction` Apex method.

SEE ALSO:

[Package Usage Logs Schema](#)

## Determine What Packaged Component Was Accessed

To analyze a package usage log record, always start with your packaged component.

In App Analytics package usage logs, the name of each packaged component is represented by the `custom_entity` field and its type is represented by the `custom_entity_type` field. Your managed package likely contains multiple packaged components.

- To identify each packaged component uniquely, combine these fields.
  - `package_id`
  - `package_version_id`
  - `managed_package_namespace`
  - `custom_entity`
  - `custom_entity_type`

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## Identify Who Interacted with Your Packaged Component

After you identify your packaged component, identify both the subscriber org and the user who triggered the interaction.

- Identify the subscriber org with the `organization_id`. Some standard fields are always populated and provide you with info about the subscriber org. Some supplemental fields, when populated, add detail about that org. This table describes the subscriber org fields.

Standard Fields	Supplemental Fields
<ul style="list-style-type: none"> <li>- <code>organization_name</code></li> <li>- <code>organization_status</code></li> <li>- <code>organization_edition</code></li> <li>- <code>organization_type</code></li> </ul>	<ul style="list-style-type: none"> <li>- <code>organization_country_code</code></li> <li>- <code>organization_language_locale</code></li> <li>- <code>organization_time_zone</code></li> <li>- <code>organization_instance</code></li> <li>- <code>cloned_from_organization_id</code></li> </ul>

- Use the `user_id_token` to identify and describe the user associated with the interaction. This hashed token represents the ID of the user who accessed the package. The ID persists, even if a user's details change, across any packages that the user interacts with.

These supplemental fields, when populated, can provide you with more data about the user.

- `user_type`
- `user_agent`
- `user_country_code`
- `user_time_zone`
- `session_key`
- `login_key`

Because `user_id_token` can represent many different usage situations, we don't recommend using App Analytics for auditing customer license usage.

## Identify How a User Interacted with Your Packaged Component

After you identify your packaged component and who interacted with it, identify how the user interacted with your packaged component.

- Identify how the user interacted with your packaged component with `log_record_type`. Other common fields associated with each interaction are:
  - `request_id`
  - `timestamp_derived`

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## Custom Object and External Object Interactions

When a log record in your package usage log has `custom_entity_type` equal to `CustomObject` or `ExternalObject`, it means that a user performed an action that resulted in a create, read, update, or delete (CRUD) interaction on your object.

To determine the type and amount of CRUD that occurred on your packaged component, focus on:

- `operation_type`
- `operation_count`

Many user actions result in CRUD, such as platform events, Apex REST API requests, or scheduled job executions. Each action is related to a `log_record_type`, and each log record has some standard fields that are always populated with data. For example, an Apex REST API request with a `log_record_type` of `ApexRestApi` always has `url`, `api_version`, `http_method`, and `http_status_code` data. Many actions produce log records with supplemental fields that are often populated. For example, an Apex REST API request sometimes has `request_status`, `referrer_uri`, and `api_type` data.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## CRUD from Apex REST API Requests

To analyze an Apex REST API request that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `ApexRestApi`. Then use these fields to dig into the details of the Apex REST API interaction.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>url</code></li> <li>• <code>api_version</code></li> <li>• <code>http_method</code></li> <li>• <code>http_status_code</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>request_status</code></li> <li>• <code>referrer_uri</code></li> <li>• <code>api_type</code></li> <li>• <code>rows_processed</code></li> <li>• <code>request_size</code></li> <li>• <code>response_size</code></li> <li>• <code>num_fields</code></li> </ul>

## CRUD from Apex SOAP API Requests

To analyze an Apex SOAP API request that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `ApexSoap`. Then use these fields to explore the details of the Apex SOAP API interaction.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>api_version</code></li> <li>• <code>class_name</code></li> <li>• <code>method_name</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>url</code></li> <li>• <code>request_status</code></li> <li>• <code>referrer_uri</code></li> </ul>

### CRUD from REST API Requests

To analyze a REST API request that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `RestApi`. Then use these fields to understand the details of the REST API interaction.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>url</code></li> <li>• <code>api_version</code></li> <li>• <code>http_method</code></li> <li>• <code>http_status_code</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>request_status</code></li> <li>• <code>referrer_uri</code></li> <li>• <code>api_type</code></li> <li>• <code>rows_processed</code></li> <li>• <code>request_size</code></li> <li>• <code>response_size</code></li> <li>• <code>num_fields</code></li> </ul>

### CRUD from SOAP API Requests

To analyze a SOAP API request that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `API`. Then use these fields to uncover the details of the SOAP API interaction.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>api_type</code></li> <li>• <code>api_version</code></li> <li>• <code>request_size</code></li> <li>• <code>response_size</code></li> <li>• <code>method_name</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>url</code></li> <li>• <code>request_status</code></li> <li>• <code>request_uri</code></li> <li>• <code>rows_processed</code></li> <li>• <code>num_fields</code></li> </ul>

### CRUD from Bulk API Requests

To analyze a Bulk API request that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `BulkApiV1` or `BulkApiV2`. Then use these fields to discover the details of the Bulk API interaction.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>api_version</code></li> <li>• <code>bulk_job_id</code></li> <li>• <code>bulk_batch_id</code></li> <li>• <code>bulk_operation</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>api_type</code></li> <li>• <code>rows_processed</code></li> </ul>

### CRUD from Scheduled Job Executions

To analyze a scheduled job execution that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `CronJob`. There are no additional package usage log fields to describe scheduled job executions.

Standard Data	Supplemental Data
none	none

### CRUD from Platform Events

To analyze a platform event that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `PlatformEventConsumer`. Then use these fields to discover the details of the platform event.

Standard Data	Supplemental Data
none	<ul style="list-style-type: none"> <li>• <code>event</code></li> <li>• <code>event_subscriber</code></li> <li>• <code>event_count</code></li> </ul>

### CRUD from Queueable Apex Executions

To analyze a queueable Apex execution that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `QueuedExec`. There are no additional package usage log fields to describe Apex executions.

Standard Data	Supplemental Data
none	none

### CRUD from Standard User Interface Requests

To analyze a user interaction that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `URI`. Then use these fields to discover the details of the user interaction.

Standard Data	Supplemental Data
<code>url</code>	<ul style="list-style-type: none"> <li>• <code>request_status</code></li> <li>• <code>referrer_uri</code></li> </ul>

### CRUD from Visualforce Remoting Requests

To analyze a Visualforce Remoting request that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `VFRemoting`. Then use these fields to explore the details of the Visualforce Remoting request.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>class_name</code></li> <li>• <code>method_name</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>url</code></li> <li>• <code>request_status</code></li> <li>• <code>referrer_uri</code></li> <li>• <code>request_size</code></li> </ul>

Standard Data	Supplemental Data
	<ul style="list-style-type: none"> <li>• response_size</li> </ul>

### CRUD from Visualforce Requests

To analyze a Visualforce request that resulted in a CRUD operation on your packaged component, look for a `log_record_type` of `VisualforceRequest`. Then use these fields to explore the details of the Visualforce request.

Standard Data	Supplemental Data
url	<ul style="list-style-type: none"> <li>• request_status</li> <li>• referrer_uri</li> <li>• request_size</li> <li>• response_size</li> </ul>

### CRUD from All Other User Actions

To analyze any other user action that results in a CRUD operation on your packaged component, look for a `log_record_type` of `UnassociatedCRUD`. There are no additional package usage log fields to describe all other interactions.

Standard Data	Supplemental Data
none	none

 **Example:** Let's look at an example package usage log record and analyze the custom or external object interaction.

```
{
  "timestamp_derived": "2022-12-15T05:47:35.945Z",
  "log_record_type": "VFRemoting",
  "request_id": "4mbhuJkvJ7Q83tlq2Z5aAk",
  "organization_id": "00Dxx0000006H21",
  "organization_name": "MyCustomer Inc.",
  "organization_status": "Demo",
  "organization_edition": "Enterprise Edition",
  "organization_country_code": "IN",
  "organization_language_locale": "en_US",
  "organization_time_zone": "Australia/Sydney",
  "organization_instance": "GS0",
  "organization_type": "Production",
  "user_id_token": "005-rBBA92863JO8GJN3pT75gp0cG8a9z1vpH6M0ti/359o=",
  "user_type": "Standard",
  "url": "uwlNmuT1+gH+xKq+xCoxiaAy00hw8B4WLeQXAbgx+mA=",
  "package_id": "033xx0000004FqD",
  "package_version_id": "04txx0000004Idi",
  "managed_package_namespace": "Acme",
  "custom_entity": "Insurance_Agent",
  "custom_entity_type": "CustomObject",
}
```

```

"operation_type": "INSERT",
"operation_count": 2,
"session_key": "9/uZ+soHD+0UqKYt",
"login_key": "5tjyGvX04w06xFgT",
"user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/IP_ADDRESS_REMOVED Safari/537.36",
"user_country_code": "IN",
"user_time_zone": "Asia/Kolkata",
"class_name": "shwGCoJjDrkhbw+CY4TFzVxFWypN07UGvtGkexbj/y4=",
"method_name": "3/UbV0E5yIW8a3c2Fb2XXjfWse1MUekEZWX44tp5TJs="
}
    
```

The Insurance\_Agent packaged component of type CustomObject had CRUD performed as a result of a user action from the subscriber org My Customer Inc. Specifically, two records were inserted during a Visualforce Remoting request that the user performed at 2022-12-15 at 05:47 am UTC.

The key data in this analysis are:

Question	Field	Value
What	<ul style="list-style-type: none"> <li>custom_entity</li> <li>custom_entity_type</li> <li>package_version_id</li> <li>managed_package_namespace</li> </ul>	<ul style="list-style-type: none"> <li>Insurance_Agent</li> <li>CustomObject</li> <li>04txx0000004Idi</li> <li>Acme</li> </ul>
Who	<ul style="list-style-type: none"> <li>organization_id</li> <li>user_id_token</li> </ul>	<ul style="list-style-type: none"> <li>00Dxx0000006H21</li> <li>05-mp26308Np7y0c891y40i/39=</li> </ul>
How	log_record_type	VFRemoting
How Much	<ul style="list-style-type: none"> <li>operation_type</li> <li>operation_count</li> </ul>	<ul style="list-style-type: none"> <li>INSERT</li> <li>2</li> </ul>
When	timestamp_derived	2022-12-15T05:47:35.945Z

In this example, the Visualforce Remoting code isn't owned by the package, so url, class\_name, and method\_name are tokenized.

```

"url": "uw1NmuT1+gH+xKq+xCoxiaAy00hw8B4WLeQXAbgx+mA=",
"class_name": "shwGCoJjDrkhbw+CY4TFzVxFWypN07UGvtGkexbj/y4=",
"method_name": "3/UbV0E5yIW8a3c2Fb2XXjfWse1MUekEZWX44tp5TJs="
    
```


If the Visualforce Remoting code is part of the package, you see actual values instead of tokens.

SEE ALSO:

[Package Usage Logs Schema](#)

## Lightning Interactions

Each record in your package usage log that has a `custom_entity_type` of `LightningComponent` or `LightningPage` describes an interaction with your packaged Lightning component or page.

 **Note:** We're continually improving the recording of Lightning interaction data in package usage logs. Many interactions with your packaged Lightning component or page are available in AppExchange App Analytics, but not all. To determine which interactions we capture for your specific package, compare your packaged components to your App Analytics package usage logs.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## Lightning User Interaction

When a user interacts with your `LightningPage` or `LightningComponent` packaged component, a `log_record_type` of `LightningInteraction` is created. Some standard fields are always populated with data. For example, a Lightning component interaction always has `app_name` and `ui_event_source` data. Lightning interactions have supplemental fields that are often populated. For example, a Lightning interaction sometimes also has `page_app_name` and `page_context` data.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>app_name</code></li> <li>• <code>ui_event_source</code></li> <li>• <code>ui_event_type</code></li> <li>• <code>target_ui_element</code></li> <li>• <code>parent_ui_element</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>page_app_name</code></li> <li>• <code>page_context</code></li> <li>• <code>related_list</code></li> <li>• <code>page_url</code></li> </ul>

## Lightning Page View

When a user views your Lightning page, a `log_record_type` of `LightningPageView` is created. Some standard fields are always populated with data. For example, a Lightning page view always has `app_name` and `page_app_name` data. Lightning page views have supplemental fields that are often populated. For example, a Lightning page view sometimes also has `page_entity_type` and `prevpager_url` data.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>app_name</code></li> <li>• <code>page_app_name</code></li> <li>• <code>page_context</code></li> <li>• <code>page_url</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>page_entity_type</code></li> <li>• <code>prevpager_url</code></li> <li>• <code>related_list</code></li> </ul>

 **Example:** Let's look at an example package usage log record and analyze the Lightning interaction.

```
{
  "timestamp_derived": "2022-11-22T06:17:39.167Z",
  "log_record_type": "LightningInteraction",
  "request_id": "TID:7635077000004b3035",
```



```

"organization_id": "00Dxx0000006H21",
"organization_name": "MyCustomer Inc.",
"organization_status": "Demo",
"organization_edition": "Enterprise Edition",
"organization_country_code": "IN",
"organization_language_locale": "en_US",
"organization_time_zone": "Australia/Sydney",
"organization_instance": "GS0",
"organization_type": "Production",
"user_id_token": "005-9BwnBWYO5FMn4cZ1sLw7F3LmTpoe8M77GrZOZHL6xQk=",
"user_type": "Standard",
"package_id": "033xx0000004FqD",
"package_version_id": "04txx0000004Idi",
"managed_package_namespace": "Acme",
"custom_entity": "Acme__Insurance_Agents",
"custom_entity_type": "LightningPage",
"session_key": "214YtFB/RmsRKVsS",
"login_key": "fGV6RgVOH3ZCgl2v",
"user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/IP_ADDRESS_REMOVED Safari/537.36",
"user_country_code": "US",
"user_time_zone": "America/Los_Angeles",
"app_name": "one:one",
"page_app_name": "Insurance_App",
"page_context": "app_flexipage:lwcAppFlexipageWrapper",
"ui_event_source": "click",
"ui_event_type": "user",
"ui_event_sequence_num": "10",
"target_ui_element": "setup-app-nav-menu-item-link",
"parent_ui_element": "global-setup",
"page_url": "/lightning/n/Acme__Insurance_Agents"
}

```

The Acme\_\_Insurance\_Agents Lightning page was interacted with as a result of a user action from subscriber org MyCustomer Inc. Specifically, a Lightning interaction took place on the page on 2022-11-22 at 6:17 am.

The key data in this analysis are:

Question	Field	Value
What	<ul style="list-style-type: none"> <li>custom_entity</li> <li>custom_entity_type</li> <li>package_version_id</li> <li>managed_package_namespace</li> </ul>	<ul style="list-style-type: none"> <li>Acme__Insurance_Agents</li> <li>LightningPage</li> <li>04txx0000004Idi</li> <li>Acme</li> </ul>
Who	<ul style="list-style-type: none"> <li>organization_id</li> <li>user_id_token</li> </ul>	<ul style="list-style-type: none"> <li>00Dxx0000006H21</li> <li>005-9BwnBWYO5FMn4cZ1sLw7F3LmTpoe8M77GrZOZHL6xQk=</li> </ul>
How	log_record_type	LightningInteraction
When	timestamp_derived	2022-11-22T06:17:39.167Z

 **Note:** Lightning interaction data is captured on an event by event basis.

SEE ALSO:

- [Package Usage Logs Schema](#)
- [Lightning Interaction Event Type](#)
- [Lightning Page View Event Type](#)

## Apex Interactions

Each record in your package usage log that has a `custom_entity_type` of `ApexClass` or `ApexTrigger` describes an interaction with your packaged Apex class or trigger.

Available in: both Salesforce Classic and Lightning Experience

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

## Apex Execution


When `log_record_type` is `ApexExecution`, the log record is associated with a user action that resulted in the execution of Apex code from an Apex class or trigger. Only the outermost Apex is captured.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>entry_point</code></li> <li>• <code>quiddity</code></li> </ul>	<code>num_soql_queries</code>

## Apex Unexpected Exception

When `log_record_type` is `ApexUnexpectedException`, the log record is associated with a user action that resulted in an Apex class or trigger throwing an unhandled exception. The `stack_trace` field provides detail about the Apex unexpected exceptions.

Standard Data	Supplemental Data
<code>stack_trace</code>	none

 **Example:** Let's look at an example package usage log record and analyze the Apex interaction.

```
{
  "timestamp_derived": "2022-11-22T06:19:33.990Z",
  "log_record_type": "ApexExecution",
  "request_id": "4mbhxFWBBXz83tlq2Z5aAk",
  "organization_id": "00Dxx0000006H21",
  "organization_name": "MyCustomer Inc.",
  "organization_status": "Demo",
  "organization_edition": "Enterprise Edition",
  "organization_country_code": "IN",
```

```

"organization_language_locale": "en_US",
"organization_time_zone": "Australia/Sydney",
"organization_instance": "GS0",
"organization_type": "Production",
"user_id_token": "005-9BwnBWYO5FMn4cZ1sLw7F3LmTpoe8M77GrZ0ZHL6xQk=",
"user_type": "Standard",
"package_id": "033xx0000004FqD",
"package_version_id": "04txx0000004Idi",
"managed_package_namespace": "Acme",
"custom_entity": "InsuranceDetailsBatchable",
"custom_entity_type": "ApexClass",
"session_key": "214YtFB/RmsRKVsS",
"login_key": "fGV6RgVOH3ZCgl2v",
"user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/IP_ADDRESS_REMOVED Safari/537.36",
"user_country_code": "US",
"user_time_zone": "America/Los_Angeles",
"entry_point": "Acme.InsuranceDetailsBatchable",
"num_soql_queries": "2",
"quiddity": "A"
}

```

The InsuranceAgentDetailsBatchable packaged component of type ApexClass was interacted with as a result of a user action from subscriber org MyCustomer Inc. Specifically, an execution of a batch Apex job occurred on 2022-11-22 at 6:19 am. The batch Apex job is represented by Quiddity = A.

The key data in this analysis are:

Question	Field	Value
What	<ul style="list-style-type: none"> <li>custom_entity</li> <li>custom_entity_type</li> <li>package_version_id</li> <li>managed_package_namespace</li> </ul>	<ul style="list-style-type: none"> <li>InsuranceDetailsBatchable</li> <li>ApexClass</li> <li>04txx0000004Idi</li> <li>Admc</li> </ul>
Who	<ul style="list-style-type: none"> <li>organization_id</li> <li>user_id_token</li> </ul>	<ul style="list-style-type: none"> <li>00Dxx0000006H21</li> <li>005-9BwnBWYO5FMn4cZ1sLw7F3LmTpoe8M77GrZ0ZHL6xQk=</li> </ul>
How	<ul style="list-style-type: none"> <li>log_record_type</li> <li>quiddity</li> </ul>	<ul style="list-style-type: none"> <li>ApexExecution</li> <li>A</li> </ul>
When	timestamp_derived	2022-11-22T06:19:33.990Z

SEE ALSO:

- [Package Usage Logs Schema](#)
- [Apex Developer Guide](#)

## Visualforce Interactions

Each record in your package usage log that has a `custom_entity_type` of `VisualforcePage` describes an interaction with your packaged Visualforce pages.

## Visualforce Requests

When a user performs an action that results in a request associated with your VisualForce page, `log_record_type` equals `VisualforceRequest`. One standard field is always populated with data: `url`.

Visualforce page requests also have supplemental fields that are often populated. For example, a Visualforce page request sometimes also has `request_status` and `referrer_uri` data.

Use these fields to explore the details of the Visualforce request.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

Standard Data	Supplemental Data
<code>url</code>	<ul style="list-style-type: none"> <li><code>request_status</code></li> <li><code>referrer_uri</code></li> <li><code>request_status</code></li> <li><code>response_size</code></li> </ul>



**Example:** Let's look at an example package usage log record and analyze the Visualforce request.

```
{
  "timestamp_derived": "2022-11-22T06:23:23.836Z",
  "log_record_type": "VisualforceRequest",
  "request_id": "4mbi9e1ZVef83tlq2Z5aAk",
  "organization_id": "00Dxx0000006H21",
  "organization_name": "MyCustomer Inc.",
  "organization_status": "Demo",
  "organization_edition": "Enterprise Edition",
  "organization_country_code": "IN",
  "organization_language_locale": "en_US",
  "organization_time_zone": "Australia/Sydney",
  "organization_instance": "GS0",
  "organization_type": "Production",
  "user_id_token": "005-9BwnBWYO5FMn4cZ1sLw7F3LmTpoe8M77GrZOZHL6xQk=",
  "user_type": "Standard",
  "url": "/apex/Acme__Agent_List",
  "package_id": "033xx0000004FqD",
  "package_version_id": "04txx0000004Idi",
  "managed_package_namespace": "Acme",
  "custom_entity": "/apex/Acme__Agent_List",
  "custom_entity_type": "VisualforcePage",
  "request_status": "S",
  "session_key": "214YtFB/RmsRkVsS",
  "login_key": "fGV6RgVOH3ZCg12v",
  "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/IP_ADDRESS_REMOVED Safari/537.36",
  "user_country_code": "US",
```

```

    "user_time_zone": "America/Los_Angeles",
    "request_size": "826",
    "response_size": "1830"
}
    
```

The Acme\_Agent\_List packaged component of type VisualforcePage was interacted with as a result of a user action from subscriber org MyCustomer Inc on 2022-11-22 at 6:23 am.

The key data in this analysis are:

Question	Field	Value
What	<ul style="list-style-type: none"> <li>• custom_entity</li> <li>• custom_entity_type</li> <li>• package_version_id</li> <li>• managed_package_namespace</li> </ul>	<ul style="list-style-type: none"> <li>• Acme__Agent_List</li> <li>• VisualforcePage</li> <li>• 04txx0000004Idi</li> <li>• Acme</li> </ul>
Who	<ul style="list-style-type: none"> <li>• organization_id</li> <li>• user_id_token</li> </ul>	<ul style="list-style-type: none"> <li>• 00Dxx0000006H21</li> <li>• 059BwOSM4ZiswE3rpe8M73Z0H6Q=</li> </ul>
How	log_record_type	VisualforceRequest
When	timestamp_derived	2022-11-22T06:23:23.836Z

SEE ALSO:

- [Package Usage Logs Schema](#)
- [Visualforce Developer Guide](#)

### CRM Analytics Asset Interactions

Each record in your package usage log that has a custom\_entity\_type of AnalyticsDashboard, AnalyticsLens, or AnalyticsRecipe describes an interaction with your packaged CRM Analytics assets.

#### Analytics Asset Runs

To analyze a run of your CRM Analytics asset, look for a log\_record\_type of AnalyticsAssetRun.

Standard Data	Supplemental Data
none	none

#### Analytics Asset Views

To analyze a view of your CRM Analytics asset, look for a log\_record\_type of AnalyticsAssetView.

#### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

Standard Data	Supplemental Data
none	none

 **Example:** Let's look at an example package usage log record and analyze the CRM Analytics asset interaction.

```
{
  "timestamp_derived": "2022-11-22T06:19:49.820Z",
  "log_record_type": "AnalyticsAssetView",
  "request_id": "4mbhvyfahFf83tlq2Z5aAk",
  "organization_id": "00Dxx0000006H21",
  "organization_name": "MyCustomer Inc.",
  "organization_status": "Demo",
  "organization_edition": "Enterprise Edition",
  "organization_country_code": "IN",
  "organization_language_locale": "en_US",
  "organization_time_zone": "Australia/Sydney",
  "organization_instance": "GS0",
  "organization_type": "Production",
  "user_id_token": "005-9BwnBWYO5FMn4cZ1sLw7F3LmTpoe8M77GrZOZHL6xQk=",
  "user_type": "Standard",
  "package_id": "033xx0000004FqD",
  "package_version_id": "04txx0000004Idi",
  "managed_package_namespace": "Acme",
  "custom_entity": "ClaimsDashboard",
  "custom_entity_type": "AnalyticsDashboard",
  "session_key": "214YtFB/RmsRKVsS",
  "login_key": "fGV6RgVOH3ZCg12v",
  "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/IP_ADDRESS_REMOVED Safari/537.36",
  "user_country_code": "US",
  "user_time_zone": "America/Los_Angeles"
}
```

The packaged Analytics dashboard, ClaimsDashboard, was interacted with by a standard user from the subscriber org MyCustom Inc. Specifically, the user performed a view of ClaimsDashboard on 2022-11-22 at 6:19am UTC.

The key data in this analysis are:

Question	Field	Value
What	<ul style="list-style-type: none"> <li>custom_entity</li> <li>custom_entity_type</li> <li>package_version_id</li> <li>managed_package_namespace</li> </ul>	<ul style="list-style-type: none"> <li>ClaimsDashboard</li> <li>AnalyticsDashboard</li> <li>04txx0000004Idi</li> <li>Acme</li> </ul>
Who	<ul style="list-style-type: none"> <li>organization_id</li> <li>user_id_token</li> </ul>	<ul style="list-style-type: none"> <li>00Dxx0000006H21</li> <li>005-9BwnBWYO5FMn4cZ1sLw7F3LmTpoe8M77GrZOZHL6xQk=</li> </ul>
How	log_record_type	AnalyticsAssetView

Question	Field	Value
When	timestamp_derived	2022-11-22T06:19:49.820Z

## SEE ALSO:

- [Package Usage Logs Schema](#)
- [CRM Analytics Developer Center](#)

## Custom Interactions

To understand which features and UI components a subscriber interacted with and how they flow through a user journey, create custom interactions with Apex enums and the `IsvPartners.AppAnalytics.logCustomInteraction` Apex method.

Standard Data	Supplemental Data
<ul style="list-style-type: none"> <li>• <code>class_name</code></li> <li>• <code>method_name</code></li> <li>• <code>line_number</code></li> <li>• <code>interaction_id_token</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>api_version</code></li> </ul>


### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## Successful Custom Interactions

To analyze a custom interaction with your packaged components, look for a `log_record_type` of `CustomInteraction` and a `custom_entity_type` of `CustomInteractionLabel`. The `custom_entity` contains a custom interaction label that you created and that was logged.

 **Note:** `interaction_id_token` is included only if an `interaction_id` was provided to the associated `IsvPartners.AppAnalytics.logCustomInteraction` call. `interaction_id_token` is a hashed, tokenized version of the raw interaction id that was provided.

## Unsuccessful Custom Interactions

When `custom_entity_type` is equal to `CustomInteractionFailure` then the custom interaction couldn't be logged. To determine the reason for the failed logging, review the reason code provided by the `custom_entity` value.

custom_entity	Message
LABEL_NO_NAMESPACE	We couldn't log the custom interaction with App Analytics. The interaction label provided to <code>IsvPartners.AppAnalytics.logCustomInteraction</code> must have a namespace.
LABEL_NOT_ENUM	We couldn't log the custom interaction with App Analytics. The interaction label provided to

custom_entity	Message
	IsvPartners.AppAnalytics.logCustomInteraction must be an Apex enum.
LABEL_WRONG_NAMESPACE	We couldn't log the custom interaction with App Analytics. The interaction label provided to IsvPartners.AppAnalytics.logCustomInteraction must have the same namespace as the Apex code that called the method.
OVER_CALL_LIMIT	IsvPartners.AppAnalytics.logCustomInteraction was called too many times in a single user request. This custom interaction and subsequent ones for this user request weren't logged with App Analytics.

 **Example:** Let's look at an example package usage log record and analyze a successful Apex interaction.

```
{
  "timestamp_derived": "2023-09-20T06:17:39.167Z",
  "log_record_type": "CustomInteraction",
  "request_id": "TID:7635077000004b3035",
  "organization_id": "00Dxx0000006H21",
  "organization_name": "MyCustomer Inc.",
  "organization_status": "Demo",
  "organization_edition": "Enterprise Edition",
  "organization_country_code": "IN",
  "organization_language_locale": "en_US",
  "organization_time_zone": "Australia/Sydney",
  "organization_instance": "GS0",
  "organization_type": "Production",
  "user_id_token": "005-9BwnBWY05FMn4cZ1sLw7F3LmTpoe8M77GrZOZHL6xQk=",
  "user_type": "Standard",
  "package_id": "033xx0000004FqD",
  "package_version_id": "04txx0000004Idi",
  "managed_package_namespace": "Acme",
  "custom_entity": "MyInteractionLabels.LoginButtonClicked",
  "custom_entity_type": "CustomInteractionLabel",
  "session_key": "214YtFB/RmsRkVsS",
  "login_key": "fGV6RgVOH3ZCgl2v",
  "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/IP_ADDRESS_REMOVED Safari/537.36",
  "user_country_code": "US",
  "user_time_zone": "America/Los_Angeles",
  "class_name": "Acme.MyController",
  "method_name": "loginButtonCallback",
  "line_number": 56,
  "interaction_id_token": "7NDe8HM8ZgPdBL+jiOpTW3/xKTwwL30dyxmKNxtyzi8="
}
```

The `MyInteractionLabels.LoginButtonClicked` custom interaction label was logged as a custom interaction as a result of a user action from subscriber org `MyCustomer Inc` on 2023-09-20 at 6:17 am. Specifically, the user interaction resulted in logging a custom interaction from line number 56 of the `loginButtonCallback` method found in the



Acme.MyController Apex class. In addition to the InteractionLabels.LoginButtonClicked label, an interaction ID was provided to the log call resulting in an interaction token id value of 7NDe8HM8ZgPdBL+jiOpTW3/xKTwwL30dyxmKNxtyzi8=.

The key data in this analysis are:

Question	Field	Value
What	<ul style="list-style-type: none"> <li>class_name</li> <li>custom_entity</li> <li>custom_entity_type</li> <li>package_version_id</li> <li>managed_package_namespace</li> <li>class_name</li> <li>method_name</li> <li>line_number</li> <li>interaction_id_token</li> </ul>	<ul style="list-style-type: none"> <li>Acme.MyController</li> <li>MyInteractionLabels.LoginButtonClicked</li> <li>CustomInteractionLabel</li> <li>04txx0000004Idi</li> <li>Acme</li> <li>loginButtonCallback</li> <li>56</li> <li>7NDe8HM8ZgPdBL+jiOpTW3/xKTwwL30dyxmKNxtyzi8=</li> </ul>
Who	<ul style="list-style-type: none"> <li>organization_id</li> <li>user_id_token</li> </ul>	<ul style="list-style-type: none"> <li>00Dxx0000006H21</li> <li>059BwVOSM4ZiswE3rpo8M73ZOH6Q=</li> </ul>
How	log_record_type	CustomInteraction
When	timestamp_derived	2023-09-20T06:17:39.167Z

SEE ALSO:

- [Download Package Usage Logs, Package Usage Summaries, and Subscriber Snapshots](#)
- [Considerations for Custom Interactions](#)

## Package Usage Logs Schema

Make informed development decisions based on package usage log data. Analyze adoption, user behavior, company information, and Lightning app and page usage data. Package usage logs list activity during a 24-hour period, between 12:00 AM and 11:59 PM UTC.

Field	Description
api_type	The type of API request. Examples: <ul style="list-style-type: none"> <li>BULK_API</li> <li>E: SOAP Enterprise</li> <li>P: SOAP Partner</li> <li>REST</li> </ul>
api_version	The version of the API that's used. Example: 45.0.

Field	Description
app_name	The name of the Lightning application the user accessed. Examples: <ul style="list-style-type: none"> <li>• one:one</li> <li>• FieldServiceApp</li> <li>• Chatter</li> </ul>
bulk_batch_id	The batch ID for the Bulk API job.
bulk_job_id	The ID for the Bulk API job.
bulk_operation	The operation for the Bulk API job. Examples: <ul style="list-style-type: none"> <li>• delete</li> <li>• hardDelete</li> <li>• insert</li> <li>• query</li> <li>• queryAll</li> <li>• update</li> <li>• upsert</li> </ul>
class_name	The name of the Apex class. Examples: <ul style="list-style-type: none"> <li>• Help_HomeController</li> <li>• ROAppController_v2</li> <li>• FSL</li> </ul>
cloned_from_organization_id	The ID of the org from which this subscriber org was cloned. Applies to sandbox orgs only. Example: 00Dxx00000000000
custom_entity	The developer name of the component or custom object.
custom_entity_type	The type of component or custom object that the user viewed or manipulated. Examples: <ul style="list-style-type: none"> <li>• AnalyticsDashboard</li> <li>• AnalyticsLens</li> <li>• AnalyticsRecipe</li> <li>• ApexClass</li> <li>• ApexTrigger</li> <li>• CustomInteractionFailure</li> <li>• CustomInteractionLabel</li> <li>• CustomObject</li> <li>• ExternalObject</li> </ul>

Field	Description
	<ul style="list-style-type: none"> <li>• LightningComponent</li> <li>• LightningPage</li> <li>• VisualforcePage</li> </ul>
entry_point	<p>The entry point of the executed Apex event.</p> <ul style="list-style-type: none"> <li>• GeneralCloner.cloneAndInsertRecords</li> <li>• VF- /apex/CloneUser</li> </ul>
event	<p>The name or ID of the platform event.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• /event/011xx0000005akx</li> <li>• SomeCustomEvent</li> </ul>
event_count	The number of platform events consumed by the subscriber. Example: 2.
event_subscriber	The ID of the platform event subscriber. Example: 01qxx0000004Coy.
http_method	The type of HTTP request method. Example: GET.
http_status_code	The HTTP response status code. Example: 404.
interaction_id_token	A hashed token representing the interaction ID provided when the custom interaction was logged. In compliance with privacy regulations, Salesforce can't store an actual user interaction ID.
line_number	The line number in the Apex file.
login_key	The hashed string that ties together all events in a given user's login session. The session starts with a login event and ends with either a logout event or the session expiring. All log lines with the same login key occurred during the same user login session.
log_record_type	<p>Type of log record.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• AnalyticsAssetView</li> <li>• AnalyticsAssetRun</li> <li>• API</li> <li>• ApexExecution</li> <li>• ApexRestApi</li> <li>• ApexSoap</li> <li>• ApexUnexpectedException</li> <li>• BulkApiV1</li> <li>• BulkApiV2</li> <li>• CronJob</li> <li>• CustomInteraction</li> <li>• LightningInteraction</li> </ul>

Field	Description
	<ul style="list-style-type: none"> <li>• LightningPageView</li> <li>• PlatformEventConsumer</li> <li>• QueuedExec</li> <li>• RestApi</li> <li>• UnassociatedCRUD</li> <li>• URI</li> <li>• VFRemoting</li> <li>• VisualforceRequest</li> </ul> <p>A <code>log_record_type</code> value of <code>UnassociatedCRUD</code> is assigned when a create, read, update, or delete (CRUD) event occurs on a custom object that isn't associated with a log record type that App Analytics captures or that is associated with unknown log records.</p>
<code>managed_package_namespace</code>	Namespace of the package.
<code>method_name</code>	<p>The name of the Apex method.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• <code>getUserAccessLevelBean</code></li> <li>• <code>getCurrentDocumentsRates</code></li> <li>• <code>getAdditionalHelpTemplate</code></li> </ul>
<code>num_fields</code>	The number of fields accessed by the user in this transaction.
<code>num_soql_queries</code>	The number of SOQL queries completed during the executed Apex event.
<code>operation_count</code>	<p>The definition of <code>operation_count</code> depends on the <code>operation_type</code> performed.</p> <ul style="list-style-type: none"> <li>• When <code>operation_type</code> is <code>INSERT</code>, <code>READ</code>, <code>UPDATE</code>, or <code>DELETE</code>, <code>operation_count</code> is the number of records associated with the <code>custom_entity</code> affected by the operation in this transaction.</li> <li>• When <code>operation_type</code> is <code>SOQL_QUERY</code>, <code>operation_count</code> is the number of SOQL queries associated with the <code>custom_entity</code> performed in this transaction.</li> <li>• When <code>operation_type</code> is <code>SOSL_QUERY</code>, <code>operation_count</code> is the number of SOSL queries associated with the <code>custom_entity</code> performed in this transaction.</li> </ul>
<code>operation_type</code>	<p>The operation performed on the component or custom object.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• <code>INSERT</code></li> <li>• <code>READ</code></li> <li>• <code>UPDATE</code></li> <li>• <code>DELETE</code></li> </ul>

Field	Description
	<ul style="list-style-type: none"> <li>• SOSL_QUERY</li> <li>• SOQL_QUERY</li> </ul>
organization_country_code	<p>The ISO-3166 two-character country code corresponding to the subscriber org's address at the time of sign-up.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• US</li> <li>• CA</li> <li>• FR</li> </ul>
organization_edition	<p>The name of the Salesforce edition the subscriber org is using.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• Developer Edition</li> <li>• Enterprise Edition</li> <li>• Unlimited Edition</li> </ul>
organization_id	The 15-character ID of the subscriber org.
organization_instance	<p>The name of the subscriber org's instance.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• AP2</li> <li>• EU7</li> <li>• NA44</li> </ul>
organization_language_locale	<p>The 2–5 character code that represents the language and locale ISO-639 code of the subscriber org. This code controls the language for the labels displayed in an application.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• de-DE</li> <li>• en-US</li> <li>• fr-CA</li> </ul>
organization_name	The name of the subscriber org. Example: Acme, Inc.
organization_status	<p>The paid status of the subscriber org.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• Active</li> <li>• Demo</li> <li>• Free</li> <li>• Trial</li> </ul>
organization_time_zone	<p>The default time zone of the subscriber org.</p> <p>Examples:</p>

Field	Description
	<ul style="list-style-type: none"> <li>America/New York</li> <li>America/Los Angeles</li> <li>Europe/Paris</li> </ul>
organization_type	<p>The subscriber org environment type.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>Production</li> <li>Sandbox</li> </ul>
package_id	The ID of the package.
package_version_id	The ID of the package version.
page_app_name	<p>The internal name of the Lightning application that the user accessed from the App Launcher.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>LightningSales</li> <li>Chatter</li> </ul>
page_context	The context of the Lightning page where the event occurred. Example: <code>clients:cardContainer</code> .
page_entity_type	<p>The Lightning entity type of the event.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>Contact</li> <li>Task</li> </ul>
page_url	<p>The relative URL of the top-level Lightning Experience or Salesforce mobile app page that the user accessed. The page can contain one or more Lightning components. Multiple record IDs can be associated with page_url. Example: <code>/sObject/0064100000JXITSASS/view</code></p>
parent_ui_element	The parent scope of the Lightning page element where the event occurred. Example: <code>ChatterFeed</code> .
prevpage_url	The relative URL of the previous Lightning Experience or Salesforce mobile app page that the user opened. Example: <code>/sObject/0064100000</code>
quiddity	<p>The type of outer execution associated with the executed Apex event.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>A: QueryLocator Batch Apex</li> <li>B: Bulk API and Bulk API 2.0</li> <li>BA: Batch Apex</li> <li>C: Scheduled Apex</li> <li>E: Inbound Email Service</li> </ul>

Field	Description
	<ul style="list-style-type: none"> <li>• F: Future</li> <li>• H: Apex REST</li> <li>• I: Invocable Action</li> <li>• K: Quick Action</li> <li>• L: Lightning</li> <li>• M: Remote Action</li> <li>• Q: Queuable</li> <li>• R: Synchronous Uncategorized</li> <li>• S: Serial Batch Apex</li> <li>• TA: Tests Async</li> <li>• TD: Tests Deployment</li> <li>• TS: Tests Synchronous</li> <li>• V: Visualforce</li> <li>• W: SOAP Webservices</li> <li>• X: Execute Anonymous</li> </ul>
referrer_uri	<p>The referring URI from the HTTP request. URIs are redacted in these ways.</p> <ul style="list-style-type: none"> <li>• Query strings are removed.</li> <li>• User IDs display as hashed tokens.</li> <li>• Subscriber-created URIs, such as VisualForce pages, are removed.</li> </ul>
related_list	<p>A section of a record or other detail page that lists items related to that record.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• Open Activities</li> <li>• Stage History</li> </ul>
request_id	<p>The ID of the HTTP request made to the server by the browser. If multiple log lines have the same request ID, they all occurred as part of the same user interaction.</p>
request_size	<p>The size of the callout request body in bytes.</p>
request_status	<p>The status of the HTTP request for the page or action that accesses a component or custom object in a package.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• A = Auth Error</li> <li>• F = Failure</li> <li>• N = 404 error</li> <li>• R = Redirect</li> <li>• S = Success</li> <li>• U = Undefined</li> </ul>

Field	Description
<code>response_size</code>	The size of the callout response in bytes.
<code>rows_processed</code>	The number of rows that were processed in the request.
<code>session_key</code>	The HTTP session ID for the HTTP request to access a component or custom object in a package. The session ID is hashed.
<code>stack_trace</code>	The stack trace associated with the Apex exception.
<code>target_ui_element</code>	The Lightning target page element where the event occurred. Examples: <ul style="list-style-type: none"> <li><code>label body truncate</code></li> <li><code>tabitem-link</code></li> </ul>
<code>timestamp_derived</code>	The access time of a component or custom object in a package in ISO8601-compatible format (YYYY-MM-DDTHH:MM:SS.sssZ). Example: 2018-07-27T11:32:59.555Z.
<code>ui_event_sequence_num</code>	An auto-incremented sequence number of the current Lightning event since the session started.
<code>ui_event_source</code>	The user action on the Lightning record or records. This value indicates whether the user's action was on a single record or multiple records. For example, <code>read</code> indicates that one record was read, such as on a record detail page. In contrast, <code>reads</code> indicates that multiple records were read, such as in a list view. Examples: <ul style="list-style-type: none"> <li><code>click</code></li> <li><code>create</code></li> <li><code>delete</code></li> <li><code>hover</code></li> <li><code>read</code></li> <li><code>update</code></li> </ul>
<code>ui_event_type</code>	The type of Lightning event. Examples: <ul style="list-style-type: none"> <li><code>crud</code></li> <li><code>system</code></li> <li><code>user</code></li> </ul>
<code>url</code>	The redacted URL of the request to access a component or custom object in a package. URLs are redacted in these ways. <ul style="list-style-type: none"> <li>Query strings are removed.</li> <li>User IDs display as hashed tokens.</li> <li>Subscriber-created URIs, such as VisualForce pages, are removed.</li> </ul>



Field	Description
	For Lightning-based URLs, only <code>/aura</code> is displayed. For Visualforce-based URLs that aren't pages owned by the managed package, either <code>/apex</code> or <code>/apexrest</code> is displayed.
<code>user_agent</code>	The browser and operating system of the device used to make the request. Examples: <ul style="list-style-type: none"> <li>• Mozilla/5.0 (iPhone; CPU iPhone // 12_0 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) CriOS/69.0.3497.105 Mobile/15E148 Safari/605.1</li> <li>• Mozilla/5.0 (Linux; Android 8.0.0; SM-G960F Build/R16NW) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.84 Mobile Safari/537.36</li> </ul>
<code>user_country_code</code>	The default ISO-3166 two-character country code of the user. Examples: <ul style="list-style-type: none"> <li>• CA</li> <li>• FR</li> <li>• US</li> </ul>
<code>user_id_token</code>	The hashed token representing the ID of the user who accessed the package. The ID persists, even if a user's details change. The token also persists across any packages that the user interacts with.  The user ID token starts with the prefix 005-. In compliance with privacy regulations, our systems can't access the actual user ID. Likewise, the hashed token can't be linked to the user ID.
<code>user_time_zone</code>	The default time zone of the user. Example: <code>America/New_York</code> .
<code>user_type</code>	The user license category of the user accessing Salesforce services through the UI or API. Examples: <ul style="list-style-type: none"> <li>• Guest</li> <li>• Partner</li> <li>• Standard</li> </ul>

## Subscriber Snapshots

Subscriber snapshots give you a point-in-time summary of subscriber activity. Use subscriber snapshots to see usage trends by org and package.


 **Note:** AppExchange App Analytics is subject to certain usage restrictions, as described in the [AppExchange Program Policies](#).

AppExchange App Analytics takes a daily snapshot of org, package, and custom entity data. Snapshots are captured daily at 00:00 UTC and become available for download immediately. You request a date and time, or range of dates and times, and you receive one snapshot per valid date and time requested. For example, if on April 7, 2023 you request a date and time range of `StartTime=2023-04-04T00:00:00Z EndTime=2020-04-07T00:00:00Z`, you receive three snapshots, one for each completed day.

Field	Description
attribute_name	Represents a characteristic of a custom entity, managed package, package version, or org. Example: <code>UsersWithMFA</code>
attribute_value	A string that represents a characteristic or measure of an <code>attribute_name</code> . Examples: <ul style="list-style-type: none"> <li>• 0.570</li> <li>• 1.000</li> <li>• Acme, Inc.</li> <li>• Active</li> <li>• Deprecated</li> </ul>
count	Total records for the custom entity in that org on the specified snapshot date.
custom_entity	The developer name of the component or custom object. Examples: <ul style="list-style-type: none"> <li>• Amount</li> <li>• Travel_Expense</li> </ul>
date	The subscriber snapshot date requested, in <code>YYYY-MM-DDT00:00:00Z</code> format. Each point-in-time snapshot is captured at <code>00:00 UTC</code> on the date specified. Example: <code>2023-04-04T00:00:00Z</code>
managed_package_namespace	Namespace of the package. Example: <code>sfdx_isv_pkg001</code>
organization_edition	The name of the Salesforce edition the subscriber org is using. Examples: <ul style="list-style-type: none"> <li>• Developer Edition</li> <li>• Enterprise Edition</li> <li>• Unlimited Edition</li> </ul>
organization_id	The 15-character ID of the subscriber org. Example: <code>00D4m000000Td8Y.</code>
organization_name	The name of the subscriber org. Example: <code>My_Org.</code>
organization_status	The paid status of the subscriber org. Examples: <ul style="list-style-type: none"> <li>• ACTIVE</li> <li>• DEMO</li> <li>• FREE</li> <li>• TRIAL</li> </ul>
package_id	The ID of the managed package. Example: <code>033xx00000000CI.</code>
package_version_id	The ID of the managed package version. Example: <code>04t6A0000004eytQAA.</code>

The `attribute_name` and `attribute_value` fields are a key-value pair. Each pair has a specific scope. Some pairs provide org-level metadata, and others provide custom entity, managed package, or package version metadata,

Interpret these two fields in tandem using the information in this table.

 **Note:** As of Spring '25, trial orgs aren't included in subscriber snapshot MFA data.

<code>attribute_name</code>	Description	<code>attribute_value</code>	Scope
<code>UsersWithMFA</code>	<p>Represents the percentage of your unique, standard users who enabled multi-factor authentication (MFA) using one of these methods.</p> <ul style="list-style-type: none"> <li>User permission sets</li> <li>Profile permission sets</li> <li>High-assurance session security level</li> </ul> <p>The corresponding <code>attribute_value</code> is always between 0 and 1, where 0 represents 0% and 1 represents 100%.</p>	<p>Percent</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>0.060</li> <li>0.940</li> </ul>	<p>Org-level.</p> <p>For all packages installed into an active or demo org, the same org-level <code>UsersWithMFA</code> percent repeats on every package version row.</p>

## Test Custom Integrations

To test your custom integrations in a nonproduction environment, use AppExchange App Analytics Simulation Mode. Submit an App Analytics query request and receive sample usage data.

 **Note:** AppExchange App Analytics is subject to certain usage restrictions as described in the [AppExchange Program Policies](#).

To receive sample usage data, enable simulation mode, then submit a query request that includes a simulation mode package ID.

1. Enable simulation mode in your test org using the Metadata API [AppAnalyticsSettings](#) `enableSimulationMode` org preference.
2. To simulate package usage log, usage summary, or subscriber snapshot downloads, complete the required fields in your SOAP API [AppAnalyticsQueryRequest](#). Include `DataType`, and leave `OrganizationIDs` blank. For `PackageIDs`, include at least one simulation mode package ID that matches the scenario you're testing.


### USER PERMISSIONS

To enable simulation mode:

- `ModifyMetadata`

Package Dataset	Simulation Mode Package ID	Description
Small Dataset	033xx00SIMsmall	Contains a small amount of data. For use with all query types. Use this package ID to download data for any query-allowed timespan.

Package Dataset	Simulation Mode Package ID	Description
Large Dataset	033xx00SIM1arge	Contains a large amount of data for two org IDs (00Dxx00SIM00f00 and 00Dxx00SIM00bar). For use only with package usage log queries.
Empty Dataset	Use any other 15-character package ID prefixed with 033xx00SIM.  Examples: <ul style="list-style-type: none"> <li>• 033xx00SIMempty</li> <li>• 033xx00SIM44444</li> </ul>	Contains no data. For use with all query types. Use one of these package IDs to return an empty dataset.

3. Submit your query.
  4. Check your API request.
    - a. If successful, retrieve the App Analytics Query Request object created in the API request. The `DownloadURL` field populates when the request is completed.
    - b. If you get an error, edit your query. Use a smaller time window, such as a 14 days, or specify one org ID. Then resubmit your query.
  5. Download the comma-separated value (.csv) file containing sample usage data from the `DownloadURL` field in the App Analytics Query Request object.
-  **Important:** When simulation mode is enabled, you can only access our sample usage data. Disable simulation mode to access your production data.

## AppExchange App Analytics Developer Cookbook

Delve deeper into your AppExchange App Analytics managed package usage data by creating key performance indicators (KPIs). First, complete some prerequisites and retrieve your App Analytics data. Next, prepare your CRM Analytics environment. Finally, to build your KPIs, complete App Analytics recipes.

 **Note:** AppExchange App Analytics is subject to certain usage restrictions as described in the [AppExchange Program Policies](#).

### 1. [What Are Recipes?](#)

The AppExchange App Analytics Developer Cookbook uses two distinct types of recipes: CRM Analytics recipes and App Analytics recipes. The CRM Analytics recipes are foundational work that you must complete before creating App Analytics recipes. App Analytics recipes build on your CRM Analytics recipe analytics environment and result in key performance indicators (KPIs).

### 2. [Before You Begin](#)

Complete these prerequisites before you create App Analytics recipes.

#### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### 3. CRM Analytics Recipes

Set up your org to create AppExchange App Analytics recipes by building your CRM Analytics environment. You first create a country-codes dataset. Then you create two CRM Analytics recipes to produce a dataset of your subscriber info, and an aggregate dataset of all of your daily data.

### 4. App Analytics Recipes

To understand how your customers are using your managed packages and components, create App Analytics recipes. Each App Analytics recipe produces a CRM Analytics lens and is a key performance indicator (KPI). Use CRM Analytics dashboards to visualize your KPIs and gain deeper insights.

## What Are Recipes?

The AppExchange App Analytics Developer Cookbook uses two distinct types of recipes: CRM Analytics recipes and App Analytics recipes. The CRM Analytics recipes are foundational work that you must complete before creating App Analytics recipes. App Analytics recipes build on your CRM Analytics recipe analytics environment and result in key performance indicators (KPIs).

You can use any reporting tool to create KPIs, but we recommend our analytics powerhouse, CRM Analytics. With CRM Analytics, you can easily integrate your License Management App (LMA) data with your App Analytics data using datasets and CRM Analytics recipes.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## CRM Analytics Recipes

If you're familiar with CRM Analytics, you're familiar with dataflows and CRM Analytics recipes. Dataflows are great for combining data from multiple sources, while CRM Analytics recipes are great for performing transformations on a single dataset. To set up your App Analytics recipe environment, create CRM Analytics recipes that combine a country code dataset, your LMA data, and your App Analytics data. These CRM Analytics recipes are required to create App Analytics recipes.

## App Analytics Recipes

App Analytics recipes are CRM Analytics lens formulas with SAQL code provided. Each App Analytics recipe results in a KPI that you can use to visualize your data on a dashboard. Some examples include Daily and Monthly Active Users, and Custom Object Reads Per Day. Complete your CRM Analytics recipes before starting with App Analytics recipes.

## Before You Begin

Complete these prerequisites before you create App Analytics recipes.

To brush up on your AppExchange App Analytics or CRM Analytics skills, we recommend completing these Trailhead modules.

- [AppExchange Partner Intelligence Basics](#)
- [CRM Analytics Data Integration Basics](#)

1. Set up your [License Management Org](#) (LMO).

Use your LMO to track all Salesforce users who install your managed package. The LMO receives a notification in the form of a lead record when a user installs or uninstalls your package. It also tracks each package upload on AppExchange. Typically, as an AppExchange partner, you use your [Partner Business Org](#) (PBO) as your LMO.

2. Register your security-reviewed managed package with your LMO. Follow the directions in [Link a Package with Your License Management Organization](#).

3. If you're not using your PBO as your LMO, install the License Management App (LMA) in your LMO. The LMA lets you manage leads and licenses for your AppExchange offerings. To install the LMA, read [Get Started with the License Management App](#).



**Note:** If you're using your PBO as your LMO, you're all set. The LMA is automatically installed for you.

4. Create an App Analytics Admin permission set that includes create and read access on the AppAnalyticsQueryRequest object. Assign this permission to any non-Admin users so that they can create App Analytics requests. Read [Create Permission Sets](#) in Salesforce Help if you need instructions.

5. Set up the CLI using the [Salesforce CLI Setup Guide](#). If you need a CLI refresher, take the [App Development with Salesforce DX](#) Trailhead module.

6. [Enable CRM Analytics](#) in your Salesforce org.

7. Create a CRM Analytics app named PartnerIntelligence.

8. To request and retrieve a sample package usage log, create an AppExchange App Analytics query request using the CLI. Save the CSV data file as `RawPackageLogFile.csv`.

9. To request and retrieve package usage logs automatically, create an automation. Which automation method you choose depends on your business specifications and which data volume you're automating.

- For smaller datasets, such as package usage summaries, Apex scales well for automation. This [GitHub repo](#) has the details.
- For larger datasets, such as package usage logs, automate using an [Amazon Web Services \(AWS\) stack](#).
- You can also use the free Salesforce Labs app, [App Analytics](#). It offers great functionality to retrieve and automate data collection and to get started with recipes and dashboards. Salesforce Labs apps are developed by Salesforce employees and are unsupported.

### Get Help with Prerequisites

If you need help with setting up your solution, you can request a consultation with a Platform Expert.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### USER PERMISSIONS

To access License Management App data, packages, and package versions:

- Read on Licenses, Packages, Package Versions

To request and retrieve AppExchange App Analytics data:


- Create, Read, Edit, Delete, View All, and Modify All on the AppAnalyticsQueryRequest object

To use CRM Analytics:

- CRM Analytics Plus Admin user

## Get Help with Prerequisites

If you need help with setting up your solution, you can request a consultation with a Platform Expert.

1. Log in to the [Salesforce Partner Community](#).
2. Click the question icon  and then click **Log a Case for Help**.
3. Provide any required details, and then click **Create Case**.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## CRM Analytics Recipes

Set up your org to create AppExchange App Analytics recipes by building your CRM Analytics environment. You first create a country-codes dataset. Then you create two CRM Analytics recipes to produce a dataset of your subscriber info, and an aggregate dataset of all of your daily data.

- The first CRM Analytics recipe, LMAJoin, combines package and license data from your LMA with your accounts and leads. It produces a dataset of your subscribers.
- The second CRM Analytics recipe, DailyAggregation, joins the LMAJoin dataset with your App Analytics data. It produces the DailyAggregation dataset. All your App Analytics recipes are built on top of your DailyAggregation dataset.

### 1. [Create the Country-Codes Dataset](#)

To create visualizations of your country-based data in map format, you normalize the LMA country-code data to CRM Analytics country-code format.

### 2. [Connect to Your License Management App Data](#)

Create an SFDC\_Local connection to your License Management App (LMA) data.

### 3. [Create the LMAJoin CRM Analytics Recipe](#)

Create a CRM Analytics recipe that contains your License Management App (LMA) data.

### 4. [Create Your App Analytics Dataset](#)

Create a RawPackageLogFile App Analytics dataset using your `RawPackageLogFile.csv` file.

### 5. [Create Your DailyAggregation CRM Analytics Recipe](#)

You join your raw package log file data with your License Management App (LMA) data to create the DailyAggregation CRM Analytics recipe. The recipe produces a dataset called DailyAggregation that you use to create App Analytics recipes.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### SEE ALSO:

[Explore Data and Take Action with CRM Analytics](#)

## Create the Country-Codes Dataset

To create visualizations of your country-based data in map format, you normalize the LMA country-code data to CRM Analytics country-code format.

1. Click [country-codes.csv](#) to download standardized country code data.
2. Right-click **Raw** and click **Save Link As**.
3. Name the file `country-codes.txt`, and save it to your desktop.
4. In Analytics Studio in CRM Analytics, click **Create**.
5. Click **Dataset**.
6. Click **CSV File**.
7. Select your `country-codes.txt` file.
8. Click **Next**.
9. Name your dataset `country-codes`.
10. Select your **PartnerIntelligence** app.
11. Click **Next**.
12. Click **Upload File**.

## Connect to Your License Management App Data

Create an SFDC\_Local connection to your License Management App (LMA) data.

In your org in Analytics Studio in CRM Analytics:

1. Click **Data Manager**.
2. Click **Connect**.
3. Click **Connect to Data**.
4. Click **SFDC\_LOCAL**.
5. Click **Account**.
6. Click **Continue**.
7. Select all fields.
8. Click **Continue**.
9. Click **Save**.
10. Repeat steps 2 through 8 to connect to these objects.
  - **Lead**
  - **sfLma\_\_License\_\_c**
  - **sfLma\_\_Package\_\_c**
  - **sfLma\_\_Package\_Version\_\_c**
11. Next to Account, click the down arrow.
12. Click **Run Data Sync**.
13. Repeat step 11 for these objects in your Connect window.
  - **Lead**

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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



- `sfLma__License__c`
- `sfLma__Package__c`
- `sfLma__Package_Version__c`

## Create the LMAJoin CRM Analytics Recipe

Create a CRM Analytics recipe that contains your License Management App (LMA) data.

In your org in Analytics Studio in CRM Analytics:

1. Click **Data Manager**.
2. In Dataflows & Recipes on the Recipes tab, click **Create Recipe**.
3. Click **Add Input Data**.
4. Select `sfLma__License__c`, and select all columns.
5. Create a transform named *License* with these specifications.
  - Custom Formula: `string(Id)`
  - Output Type: **Text**
  - Length: `255`
  - Default Value: `blank`
  - Show Results In: **New Column (and Keep Original)**
  - Column Label: `LicenseRecordId`
6. Add a join to Lead with these specifications.
  - Select Input Data to Join: **Lead**
  - Columns to Select: **Company, First Name, Id, Last Name**
  - Join Type: **Lookup**
  - Join Keys: **License: Record ID = Lead ID**
  - API Name Prefix for Right Columns: `Lead`
7. Add a join to Account with these specifications.
  - Select Input Data to Join: **Account**
  - Columns to Select: **Name**
  - Join Type: **Lookup**
  - Join Keys: **Account Name = Account Name**
  - API Name Prefix for Right Columns: `Account`
8. Add a join to `sfLma__Package__c` with these specifications.
  - Select Input Data to Join: `sfLma__Package__c`
  - Columns to Select: `All fields`
  - Join Type: **Lookup**
  - Join Keys: **Package = Record ID**
  - API Name Prefix for Right Columns: `Package`
9. Create a transform between the join and `sfLma__Package__c` with these specifications.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

- Custom Formula: `substr(sfLma__Package_ID__c, 1, 15)`
- Output Type: **Text**
- Length: 255
- Default Value: none
- Show Results in: **New Column (and Keep Original)**
- Column Label: `PackageID15`

10. Create another join with these specifications.

- Select Input Data to Join: **sfLma\_\_Package\_Verzion\_\_c**
- Columns to Select: *All fields*
- Join Type: **Lookup**
- Join Keys: **Package Version = Record ID**
- API Name Prefix for Right Columns: `PackageVersion`

11. Create an output with these specifications.

- Write To: **Dataset**
- Dataset Display Label: `LMAJoin`
- App Location: **PartnerIntelligence**
- Sharing Source: default
- Security Predicate: **Apply row-level security to the target dataset by adding a predicate filter condition**

12. Click **Apply**.

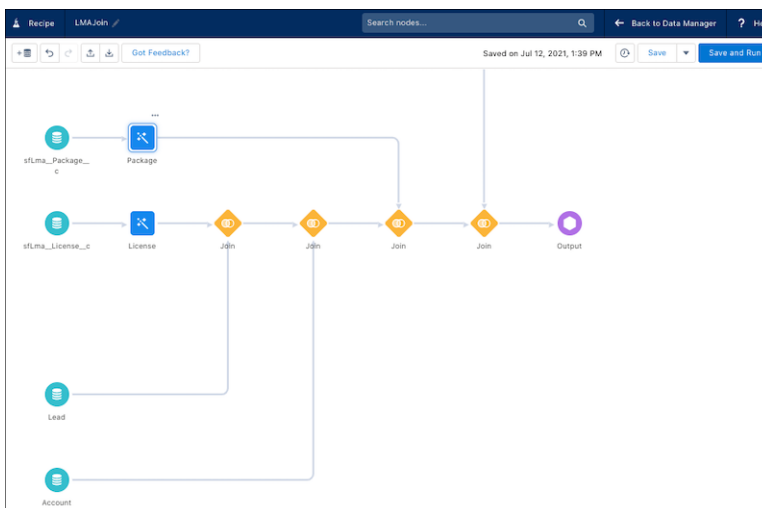
13. Click **Save**.

14. Save your recipe as `LMAJoin`.

15. Click **Save and Run**.

16. To monitor the status of your job, click **Go to Data Monitor**.

 **Example:** When complete, your LMAJoin CRM Analytics recipe looks like this.



1. [Monitor Your LMAJoin CRM Analytics Recipe](#)

CRM Analytics recipes can take a while to complete. Use these steps to monitor the status of your LMAJoin recipe.

2. [Run the LMAJoin CRM Analytics Recipe](#)

To create a reusable dataset, schedule your LMAJoin CRM Analytics recipe to run on a regular basis. We recommend daily at midnight.

## Monitor Your LMAJoin CRM Analytics Recipe

CRM Analytics recipes can take a while to complete. Use these steps to monitor the status of your LMAJoin recipe.

In your org in Analytics Studio in CRM Analytics:

1. Click **Data Manager**.
2. Click **Monitor**.
3. On the Jobs tab, locate your LMAJoin job.
4. When your job is Successful, click **Data** to view your completed LMAJoin dataset.

## Run the LMAJoin CRM Analytics Recipe

To create a reusable dataset, schedule your LMAJoin CRM Analytics recipe to run on a regular basis. We recommend daily at midnight.

In your org in Analytics Studio in CRM Analytics:

1. Click **Data Manager**.
2. Click **Dataflows & Recipes**.
3. Click the **Recipes** tab.
4. Next to your LMAJoin CRM Analytics recipe, click the arrow.
5. Click **Schedule**, and set up your schedule.

## Create Your App Analytics Dataset

Create a RawPackageLogFile App Analytics dataset using your `RawPackageLogFile.csv` file.

In your org in Analytics Studio in CRM Analytics:

1. Click **Create** and select **Dataset**.
2. Click **CSV File** and select your `RawPackageLogFile.csv` file.
3. Click **Next**.
4. Name your dataset `RawPackageLogFile` and select your **PartnerIntelligence** app.
5. Click **Next**.
6. For **event\_count**, **num\_fields**, **num\_soql\_queries**, **operation\_count**, and **rows\_processed** fields, change the field type from **Dimension** to **Measure** and add these specifications.
  - Default value: `0`
  - Scale: `0`
  - Precision: `18`

### EDITIONS

Available in: both **Salesforce Classic** and **Lightning Experience**

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

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

7. Search for **timestamp\_derived** and make sure that its field type is **Date**.
8. Click **Upload File**.

## Create Your DailyAggregation CRM Analytics Recipe

You join your raw package log file data with your License Management App (LMA) data to create the DailyAggregation CRM Analytics recipe. The recipe produces a dataset called DailyAggregation that you use to create App Analytics recipes.

In your org in Analytics Studio in CRM Analytics:

1. Click **Data Manager**.
2. Click **Dataflows & Recipes**.
3. On the Recipes tab, click **Create Recipe**.
4. Click **Add Input Data**.
5. Select **RawPackageLogFile**.
6. Select all the columns.
7. Create an aggregate with these specifications.

Field	Aggregate By
event_count	Sum
login_key	Unique
num_fields	Sum
num_soql_queries	Sum
operation_count	Sum
rows_processed	Sum
session_key	Unique

8. In the aggregate, in Group Rows, click **+**, and select **timestamp\_derived**.
  - a. Select **Year, Month, and Day**.
  - b. Click **Add**.
9. In the aggregate, in Group Rows, create a group for each of these fields.
  - **api\_type**
  - **api\_version**
  - **app\_name**
  - **class\_name**
  - **cloned\_from\_organization**
  - **custom\_entity**
  - **custom\_entity\_type**

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

- **entry\_point**
- **event**
- **event\_subscriber**
- **http\_method**
- **http\_status\_code**
- **log\_record\_type**
- **managed\_package\_namespace**
- **method\_name**
- **operation\_type**
- **organization\_country\_code**

10. Create a transform named *Create DMY Field* with this

```
formula_to_date(concat(timestamp_derived_DAY,"/",timestamp_derived_MONTH,"/",timestamp_derived_YEAR),"dd/MM/yyyy"))
```


11. Join your RawPackageLogFile dataset to your LMADData dataset using this information.

- Select Input Data to Join: **LMADData**
- Columns to Select: *All fields*
- Join Type: **Lookup**
- Join Keys: **organization\_id = Subscriber Org ID** and **package\_id = PackageID15**
- API Name Prefix for Right Columns: *LMADData*

12. Join your country-codes dataset to your LMADData dataset using this information.

- Select Input Data to Join: **country-codes**
- Columns to Select: *All fields*
- Join Type: **Lookup**
- Join Keys: **user\_country\_code = ISO3166-1-Alpha-2**
- API Name Prefix for Right Columns: *UserCountry*

13. Create a transform named *Feature Name*.

- Create as many CRM Analytics buckets as you need for your features, such as Inventory, Orders, and a catch-all bucket called Other.
-  **Note:** A CRM Analytics bucket represents a category that you use to group your data. For example, say your app contains multiple features, such as an inventory tracking feature and an order processing feature. Create a CRM Analytics bucket for each feature. Each bucket contains the custom objects, pages, Lightning components, and Apex classes that pertain to that feature. You can use these buckets to create Feature Adoption App Analytics recipes

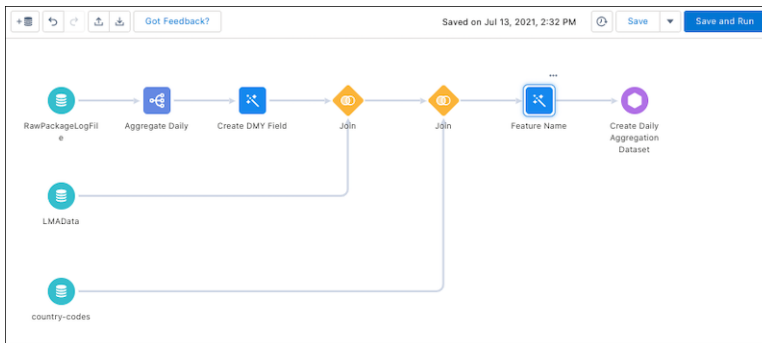
Add your custom entities to the appropriate bucket.

14. Select **Output** and use these settings.

- Write To: **Dataset**
- Dataset Display Label: *DailyAggregation*
- App Location: **PartnerIntelligence**
- Sharing Source: default
- Security Predicate: **Apply row-level security to the target dataset by adding a predicate filter condition**
- Name: *Create Daily Aggregation Dataset*

15. Click **Apply**.
16. Click **Save**.
17. Name your recipe *DailyAggregation*.
18. Click **Save and Run**.

 **Example:** When complete, your DailyAggregation recipe looks like this.



1. [Monitor the DailyAggregation CRM Analytics Recipe](#)  
CRM Analytics recipes can take a while to complete. Use these steps to monitor the status of your DailyAggregation recipe.
2. [Run the DailyAggregation CRM Analytics Recipe](#)  
To create a reusable dataset, schedule your DailyAggregation CRM Analytics recipe to run on a regular basis. We recommend daily at midnight.

### Monitor the DailyAggregation CRM Analytics Recipe

CRM Analytics recipes can take a while to complete. Use these steps to monitor the status of your DailyAggregation recipe.

In your org in Analytics Studio in CRM Analytics:

1. Click **Data Manager**.
2. Click **Monitor**.
3. On the Jobs tab, locate your DailyAggregation job.
4. When your job is Successful, click **Data** to view your completed DailyAggregation dataset.

### Run the DailyAggregation CRM Analytics Recipe

To create a reusable dataset, schedule your DailyAggregation CRM Analytics recipe to run on a regular basis. We recommend daily at midnight.

In your org in Analytics Studio in CRM Analytics:

1. Click **Data Manager**.
2. Click **Dataflows & Recipes**.
3. Click the **Recipes** tab.
4. Next to your DailyAggregation CRM Analytics recipe, click the arrow.

#### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

#### EDITIONS


Available in: both Salesforce Classic and Lightning Experience

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

5. Click **Schedule**, and set up your schedule.

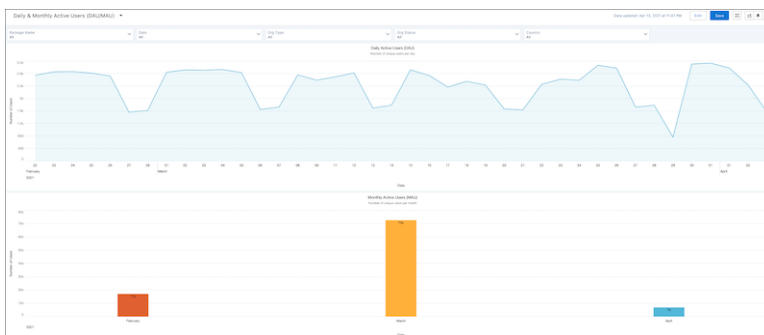
## App Analytics Recipes

To understand how your customers are using your managed packages and components, create App Analytics recipes. Each App Analytics recipe produces a CRM Analytics lens and is a key performance indicator (KPI). Use CRM Analytics dashboards to visualize your KPIs and gain deeper insights.

 **Note:** AppExchange App Analytics is subject to certain usage restrictions as described in the [AppExchange Program Policies](#). To request and retrieve package usage logs and subscriber snapshots, activate App Analytics on your security-reviewed managed package by logging a support case in the [Salesforce Partner Community](#). For product, specify **Partner Programs & Benefits**. For topic, specify **ISV Technology Request**. You can access package usage summaries without activation.

For example, to analyze a wide range of daily and monthly package usage metrics, build Daily and Monthly Active User App Analytics recipes.

 **Example:**



### Customer Success Recipes

Customer success is a relationship-focused method of ensuring that your customers achieve their desired outcomes while using your managed packages.

### Custom Object Usage Recipes

Understanding how your customers use your custom objects is critical to managing the lifecycle of your managed package and its custom objects. Start by measuring custom object usage by create, read, update, and delete (CRUD) operations.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

## Customer Success Recipes

Customer success is a relationship-focused method of ensuring that your customers achieve their desired outcomes while using your managed packages.

To measure customer success, you can create metrics that help you understand:

- Overall managed package usage
- Depth of managed package usage
- Growth
- Length of time as a customer
- Number of renewals
- Number of upsells
- Overall relationship

As you learn more about your customers and how they use your managed packages, your list of customer success metrics expands.

To analyze user behavior, we rely on user-related and CRUD (create, read, update, and delete) App Analytics data fields to calculate metrics. All user behavior calculations rely on how a unique user is defined.

- An individual that has used your managed package and its components
- Measured for a specified time period, such as a day, month, or year

An active user can be defined as: A user who has logged some type of package usage, such as CRUD activity, page views, or Lightning interactions, during a specified time period.

Segment the unique and active users by time period, such as day, month, or quarter.

### [Create a Daily Unique Users Recipe](#)

This recipe produces a unique count of users by day.

### [Create a Weekly Unique Users Recipe](#)

This recipe produces a unique count of users by week.

### [Create a Monthly Unique Users Recipe](#)

This recipe produces a unique count of users by month.

## Create a Daily Unique Users Recipe

This recipe produces a unique count of users by day.

In your org in Analytics Studio in CRM Analytics:

1. In All items on the Datasets tab, select your DailyAggregation dataset.
2. Under Bar Length, click **Count of Rows**.
3. Click **Unique**.
4. Select **user\_id\_token**.
5. Select **Charts**.
6. Click **Column**.
7. Under Bars, click **+** and search for *timestamp\_DMY*.
8. Select **Year-Month-Day**.
9. Click **Save**.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

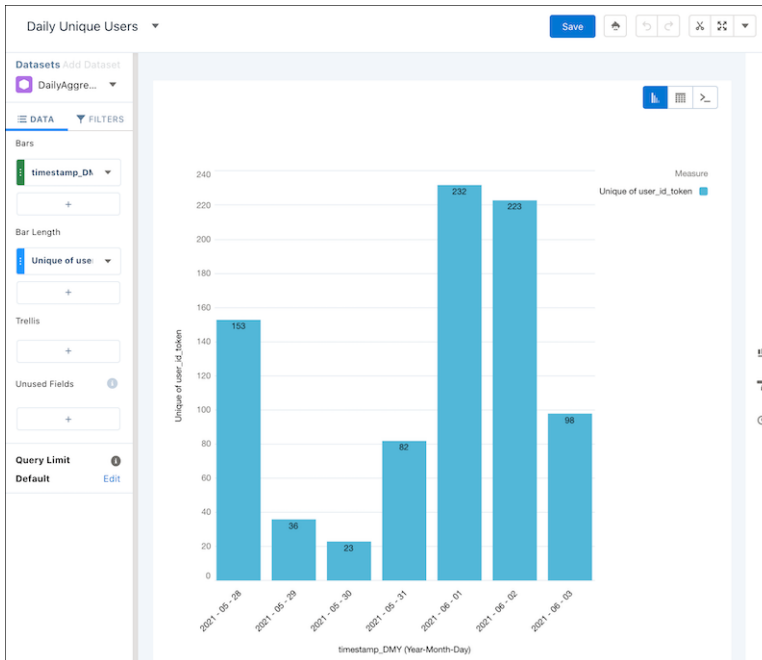


10. Name your lens *Daily Unique Users*.

11. Select your PartnerIntelligence app.

12. Click **Save**.

 **Example:**



SAQL:

```
q = load "DailyAggregation";
q = group q by ('timestamp_derived_DAY_formula_Year',
'timestamp_derived_DAY_formula_Month', 'timestamp_derived_DAY_formula_Day');
q = foreach q generate 'timestamp_derived_DAY_formula_Year' + "~~~" +
'timestamp_derived_DAY_formula_Month' + "~~~" + 'timestamp_derived_DAY_formula_Day'
as
'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month~~~timestamp_derived_DAY_formula_Day',
unique('user_id_token') as 'unique_user_id_token';
q = order q by
'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month~~~timestamp_derived_DAY_formula_Day'
asc;
q = limit q 2000;
```

## Create a Weekly Unique Users Recipe

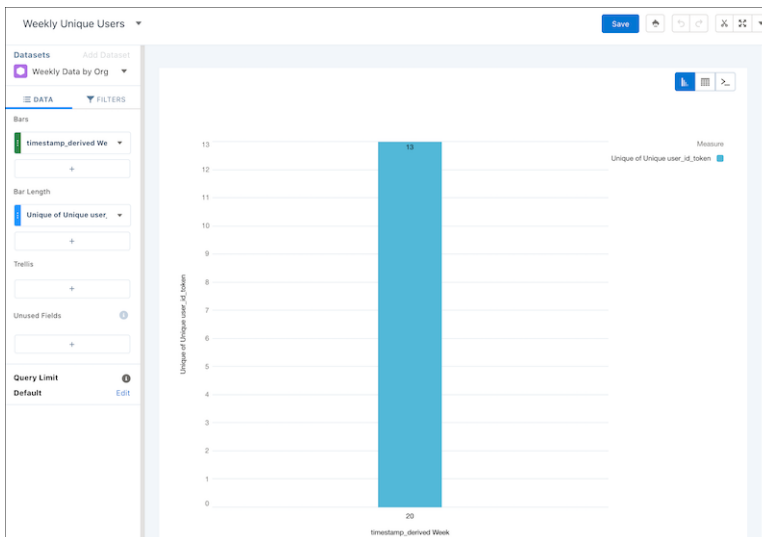
This recipe produces a unique count of users by week.

In your org in Analytics Studio in CRM Analytics:

1. In All items on the Datasets tab, select your DailyAggregation dataset.
2. Under Bar Length, click **Count of Rows**.
3. Click **Unique**.

4. Select **user\_id\_token**.
5. Select **Charts**.
6. Click **Column**.
7. Under Bars, click **+** and search for *timestamp\_DMY*.
8. Select **Year-Week**.
9. Click **Save**.
10. Name your lens *Weekly Unique Users*.
11. Select your PartnerIntelligence app.
12. Click **Save**.

 **Example:**



SAQL:

```
q = load "DailyAggregation";
q = group q by ('timestamp_derived_DAY_formula_Year',
'timestamp_derived_DAY_formula_Week');
q = foreach q generate 'timestamp_derived_DAY_formula_Year' + "~~~" +
'timestamp_derived_DAY_formula_Week' as
'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Week',
unique('user_id_token') as 'unique_user_id_token';
q = order q by 'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Week'
asc;
q = limit q 2000;
```

### Create a Monthly Unique Users Recipe

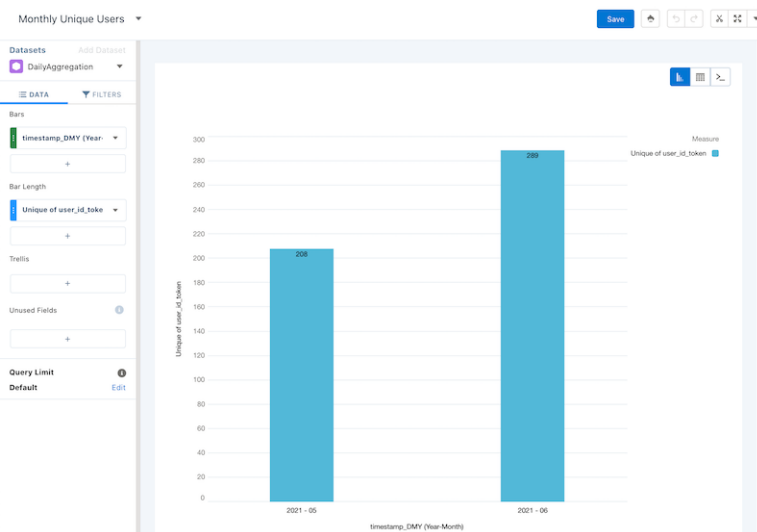
This recipe produces a unique count of users by month.

In your org in Analytics Studio in CRM Analytics:

1. In All items on the Datasets tab, select your DailyAggregation dataset.

2. Under Bar Length, click **Count of Rows**.
3. Click **Unique**.
4. Select **user\_id\_token**.
5. Select **Charts**.
6. Click **Column**.
7. Under Bars, click + and search for **timestamp\_DMY**.
8. Select **Year-Month**.
9. Click **Save**.
10. Name your lens *Monthly Unique Users*.
11. Select your PartnerIntelligence app.
12. Click Save.

 **Example:**



SAQL:

```
q = load "DailyAggregation";
q = group q by ('timestamp_derived_DAY_formula_Year',
'timestamp_derived_DAY_formula_Month');
q = foreach q generate 'timestamp_derived_DAY_formula_Year' + "~~~" +
'timestamp_derived_DAY_formula_Month' as
'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month',
unique('user_id_token') as 'unique_user_id_token';
q = order q by 'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month'
asc;
q = limit q 2000;
```

## Custom Object Usage Recipes

Understanding how your customers use your custom objects is critical to managing the lifecycle of your managed package and its custom objects. Start by measuring custom object usage by create, read, update, and delete (CRUD) operations.

### [Create a Custom Object Creates Per Day Recipe](#)

This recipe produces a unique count of how many times per day a custom object was created.

### [Create a Custom Object Updates Per Day Recipe](#)

This recipe produces a unique count of how many times per day a custom object was created.

### [Create a Custom Object Reads Per Day Recipe](#)

This recipe produces a unique count of how many times per day a custom object was read.

## EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

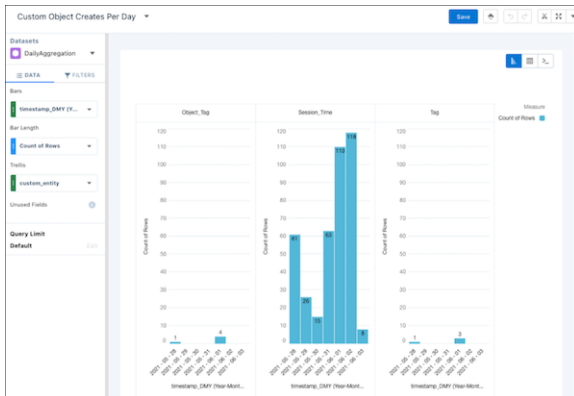
## Create a Custom Object Creates Per Day Recipe

This recipe produces a unique count of how many times per day a custom object was created.

In your org in Analytics Studio in CRM Analytics:

1. In All items on the Datasets tab, select your DailyAggregation dataset.
2. Select **Charts**.
3. Click **Column** and leave Bar Length as **Count of Rows**.
4. Under Bars, click + and search for **timestamp\_DMY**.
5. Select **Year-Month-Day**.
6. Click **Filters**.
7. Click +.
8. Select **custom\_entity\_type** equals **CustomObject**.
9. Click **Apply**.
10. Click +.
11. Select **operation\_type** Equals **INSERT**.
12. Click **Apply**.
13. Click **Data**.
14. Under Trellis, click +.
15. Select **custom\_entity**.
16. Click **Save**.
17. Name your lens *Custom Object Creates Per Day*.
18. Select your PartnerIntelligence app.
19. Click **Save**.

 Example:



SAQL:

```
q = load "DailyAggregation";
q = filter q by 'custom_entity_type' == "CustomObject";
q = filter q by 'operation_type' == "INSERT";
q = group q by ('timestamp_derived_DAY_formula_Year',
'timestamp_derived_DAY_formula_Month', 'timestamp_derived_DAY_formula_Day',
'custom_entity');
q = foreach q generate 'timestamp_derived_DAY_formula_Year' + "~~~" +
'timestamp_derived_DAY_formula_Month' + "~~~" + 'timestamp_derived_DAY_formula_Day'
as
'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month~~~timestamp_derived_DAY_formula_Day',
'custom_entity' as 'custom_entity', count() as 'count';
q = order q by
('timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month~~~timestamp_derived_DAY_formula_Day'
asc, 'custom_entity' asc);
q = limit q 2000;
```

## Create a Custom Object Updates Per Day Recipe

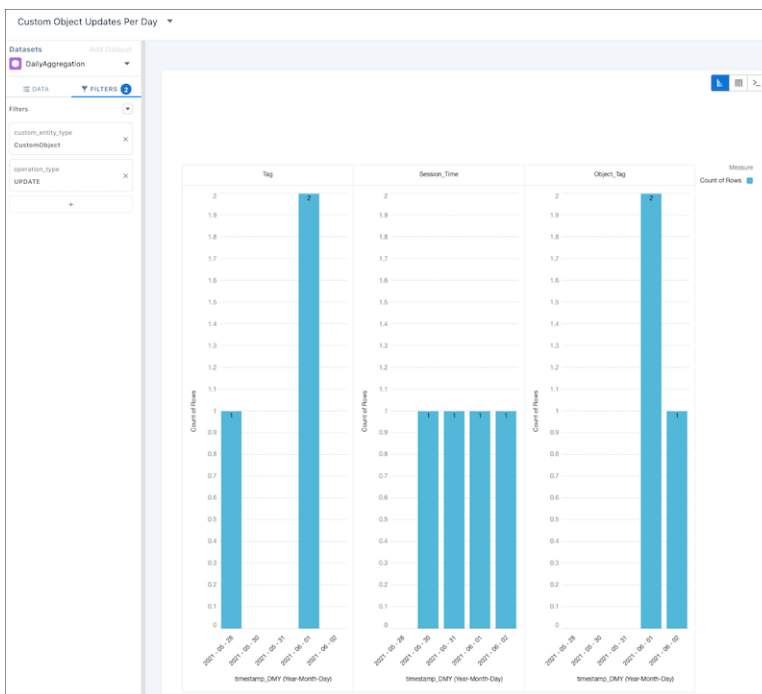
This recipe produces a unique count of how many times per day a custom object was created.

In your org in Analytics Studio in CRM Analytics:

1. In All items on the Datasets tab, select your DailyAggregation dataset.
2. Select **Charts**.
3. Click **Column** and leave Bar Length as **Count of Rows**.
4. Under Bars, click + and select **timestamp\_DMY**.
5. Select **Year-Month-Day**.
6. Click the **Filters** tab.
7. Click +.
8. Select **custom\_entity\_type** Equals **CustomObject**
9. Click **Apply**.
10. Click +.

11. Select **operation\_type** Equals **UPDATE**.
12. Click **Apply**.
13. Click the **Data** tab.
14. Under Trellis, click **+**.
15. Select **custom\_entity**.
16. Click **Save**.
17. Name your lens *Custom Object Creates Per Day*.
18. Select your PartnerIntelligence app.
19. Click **Save**.

 **Example:**



SAQL:

```
q = load "DailyAggregation";
q = filter q by 'custom_entity_type' == "CustomObject";
q = filter q by 'operation_type' == "UPDATE";
q = group q by ('timestamp_derived_DAY_formula_Year',
'timestamp_derived_DAY_formula_Month', 'timestamp_derived_DAY_formula_Day',
'custom_entity');
q = foreach q generate 'timestamp_derived_DAY_formula_Year' + "~~~" +
'timestamp_derived_DAY_formula_Month' + "~~~" + 'timestamp_derived_DAY_formula_Day'
as
'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month~~~timestamp_derived_DAY_formula_Day',
'custom_entity' as 'custom_entity', count() as 'count';
q = order q by
('timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month~~~timestamp_derived_DAY_formula_Day'
```

```
asc, 'custom_entity' asc);
q = limit q 2000;
```

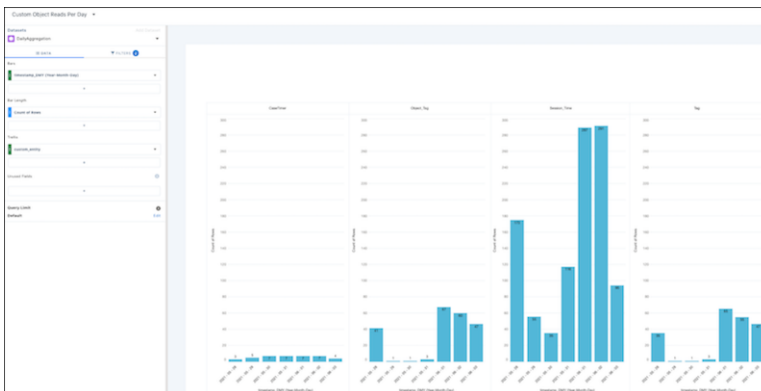
## Create a Custom Object Reads Per Day Recipe

This recipe produces a unique count of how many times per day a custom object was read.

In your org in Analytics Studio in CRM Analytics:

1. In All items on the Datasets tab, select your DailyAggregation dataset.
2. Select **Charts**.
3. Click **Column** and leave Bar Length as **Count of Rows**.
4. Under Bars, click + and search for **timestamp\_DMY**.
5. Select **Year-Month-Day**.
6. Click **Filters**.
7. Click +.
8. Select **custom\_entity\_type** Equals **CustomObject**
9. Click **Apply**.
10. Click +.
11. Select **operation\_type** Equals **READ**.
12. Click **Apply**.
13. Click **Data**.
14. Under Trellis, click +.
15. Select **custom\_entity**.
16. Click **Save**.
17. Name your lens *Custom Object Reads Per Day*.
18. Select your PartnerIntelligence app.
19. Click **Save**.

### Example:



SAQL:

```
q = load "DailyAggregation";
q = filter q by 'custom_entity_type' == "CustomObject";
q = filter q by 'operation_type' == "READ";
q = group q by ('timestamp_derived_DAY_formula_Year',
'timestamp_derived_DAY_formula_Month', 'timestamp_derived_DAY_formula_Day',
'custom_entity');
q = foreach q generate 'timestamp_derived_DAY_formula_Year' + "~~~" +
'timestamp_derived_DAY_formula_Month' + "~~~" + 'timestamp_derived_DAY_formula_Day'
as
'timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month~~~timestamp_derived_DAY_formula_Day',
'custom_entity' as 'custom_entity', count() as 'count';
q = order q by
('timestamp_derived_DAY_formula_Year~~~timestamp_derived_DAY_formula_Month~~~timestamp_derived_DAY_formula_Day'
asc, 'custom_entity' asc);
q = limit q 2000;
```

## Gaps Between First-Generation and Second-Generation Managed Packaging

---

The following functionality is supported in first-generation managed packaging, and not yet supported in second-generation managed packaging. We're working to address these feature gaps.

- Package versions can't be deprecated.
- [Apex VersionProvider](#) isn't supported.
- A default language for labels in packages can't be specified.

See the [Metadata Coverage Report](#), for the latest information on supported metadata types.