# First-Generation Managed Packaging Developer Guide

Version 65.0, Winter ’26


Last updated: January 9, 2026

# CONTENTS


**Chapter 1:** First-Generation Managed Packages . . . . . 1

Why Switch to Second-Generation Managed Packaging? . . . . . 3
Move to 2GP: Migrate Your Managed Packages with Ease . . . . . 4
- About Package Conversion and Package Migration . . . . . 5
- Three Phases of Package Migration Development . . . . . 6
- Plan Your Package Migration . . . . . 8
- Before You Begin Package Migrations . . . . . 10
- Convert Your Managed 1GP Package to 2GP . . . . . 11
- Migrate Your Subscribers from 1GP to 2GP . . . . . 14
- Move to 2GP Package Development . . . . . 15
- Considerations for Package Migrations . . . . . 16
- Troubleshoot Package Conversion Failures . . . . . 16

Register a Namespace for a First-Generation Managed Package . . . . . 18
Create a First-Generation Managed Package Using a UI . . . . . 19
- What Are Beta Versions of Managed Packages? . . . . . 20
- Create a Beta Package for First-Generation Managed Packages . . . . . 20
- Create and Upload a First-Generation Managed Package . . . . . 22
- Publish Extensions to Managed Packages . . . . . 25
- View Package Details in First-Generation Managed Packages . . . . . 26
- Notifications for Package Errors . . . . . 29

Create a First-Generation Managed Package using Salesforce DX . . . . . 30
- Enable Dev Hub and Second-Generation Managed Packaging . . . . . 32
- Limited Access License for Package Developers . . . . . 32
- Add a Limited Access User to Your Dev Hub Org . . . . . 33
- Link a Namespace to a Dev Hub Org . . . . . 33
- Scratch Orgs and Package Development . . . . . 34
- Build and Release Your App with Managed Packages . . . . . 38
- Packaging Checklist . . . . . 38
- Deploy the Package Metadata to the Packaging Org . . . . . 39
- Create a Beta Version of Your App . . . . . 39
- Install the Package in a Target Org . . . . . 40
- Create a Managed Package Version of Your App . . . . . 41
- View Information About a Package . . . . . 41

Components Available in First-Generation Managed Packages . . . . . 43
- Account Plan Objective Measure Calculation Definition . . . . . 59
- Account Relationship Share Rule . . . . . 60
- Action Link Group Template . . . . . 61
- Action Plan Template . . . . . 61
- Actionable List Definition . . . . . 62

Actionable List Key Performance Indicator Definition . . . . . 63
Activation Platform . . . . . 64
AffinityScoreDefinition . . . . . 66
Agent Action . . . . . 67
Agent Topic . . . . . 69
AI Application . . . . . 70
AI Application Config . . . . . 71
AIUsecaseDefinition . . . . . 72
Analytics . . . . . 73
Analytics Dashboard . . . . . 74
Analytics Visualization . . . . . 75
Analytics Workspace . . . . . 75
Apex Class . . . . . 76
Apex Sharing Reason . . . . . 78
Apex Trigger . . . . . 79
App Framework Template Bundle . . . . . 80
Application Subtype Definition . . . . . 81
AssessmentConfiguration . . . . . 82
AssessmentQuestion . . . . . 83
AssessmentQuestionSet . . . . . 83
Aura Component . . . . . 84
Batch Calc Job Definition . . . . . 85
Batch Process Job Definition . . . . . 86
Benefit Action . . . . . 87
Bot Template . . . . . 88
Branding Set . . . . . 90
Briefcase Definition . . . . . 91
Building Energy Intensity Record Type Configuration . . . . . 92
Business Process . . . . . 93
Business Process Group . . . . . 94
Business Process Type Definition . . . . . 95
Care Benefit Verify Settings . . . . . 96
Care Limit Type . . . . . 97
Care Request Configuration . . . . . 98
Care System Field Mapping . . . . . 99
Channel Layout . . . . . 100
Chatter Extension . . . . . 101
Claim Financial Settings . . . . . 102
CommunicationChannelType . . . . . 103
Community Template Definition . . . . . 104
Community Theme Definition . . . . . 105
Compact Layout . . . . . 106
Conditional Formatting Ruleset . . . . . 107
Connected App . . . . . 107

Context Definition . . . . . 109
Contract Type . . . . . 110
Conversation Channel Definition . . . . . 111
Conversation Vendor Info . . . . . 112
CORS Allowlist . . . . . 113
CSP Trusted Site . . . . . 114
Custom Application . . . . . 116
Custom Button or Link . . . . . 117
Custom Console Components . . . . . 118
Custom Field on Standard or Custom Object . . . . . 119
Custom Field on Custom Metadata Type . . . . . 121
Custom Field Display . . . . . 121
Custom Help Menu Section . . . . . 122
Custom Index . . . . . 123
Custom Label . . . . . 123
Custom Metadata Type Records . . . . . 124
Custom Metadata Type . . . . . 125
Custom Notification Type . . . . . 126
Custom Object . . . . . 127
Custom Object Translation . . . . . 129
Custom Permission . . . . . 130
Custom Tab . . . . . 131
Dashboard . . . . . 132
DataCalcInsightTemplate . . . . . 133
Data Connector Ingest API . . . . . 134
Data Connector S3 . . . . . 135
Data Kit Object Dependency . . . . . 136
Data Kit Object Template . . . . . 138
DataObjectBuildOrgTemplate . . . . . 139
Data Package Kit Definition . . . . . 140
Data Package Kit Object . . . . . 141
Data Source . . . . . 142
Data Source Bundle Definition . . . . . 143
Data Source Object . . . . . 144
Data Src Data Model Field Map . . . . . 145
Data Stream Definition . . . . . 147
Data Stream Template . . . . . 148
DataWeaveResource . . . . . 149
Decision Matrix Definition . . . . . 150
Decision Matrix Definition Version . . . . . 152
Decision Table . . . . . 153
Decision Table Dataset Link . . . . . 154
Digital Experience . . . . . 155
Digital Experience Bundle . . . . . 156

Decision Table . . . 157
Disclosure Definition . . . 158
Disclosure Definition Version . . . 159
Disclosure Type . . . 160
Discovery AI Model . . . 161
Discovery Goal . . . 162
Discovery Story . . . 163
Document . . . 164
Document Generation Setting . . . 165
Eclair GeoData . . . 166
Email Template (Classic) . . . 166
Email Template (Lightning) . . . 167
Embedded Service Config . . . 168
Embedded Service Menu Settings . . . 169
Enablement Measure Definition . . . 170
Enablement Program Definition . . . 171
Enablement Program Task Subcategory . . . 173
Entitlement Template . . . 174
ESignature Config . . . 175
ESignature Envelope Config . . . 176
Event Relay . . . 177
Explainability Action Definition . . . 178
Explainability Action Version . . . 179
Explainability Message Template . . . 179
Expression Set Definition . . . 180
Expression Set Definition Version . . . 182
Expression Set Object Alias . . . 183
Expression Set Message Token . . . 184
External Auth Identity Provider . . . 185
External Client App Header . . . 186
External Client App Notification Settings . . . 187
External Client App OAuth Settings . . . 188
External Client App Push Settings . . . 190
External Credential . . . 191
External Data Connector . . . 193
External Data Source . . . 194
External Data Transport Field Template . . . 195
External Data Transport Field . . . 196
External Data Transport Object Template . . . 197
External Data Transport Object . . . 198
External Document Storage Configuration . . . 200
External Services . . . 201
Feature Parameter Boolean . . . 202
Feature Parameter Date . . . 203

Feature Parameter Integer . . . . . 204
Field Set . . . . . 206
Field Source Target Relationship . . . . . 207
Flow . . . . . 208
Flow Category . . . . . 211
Flow Definition . . . . . 212
Flow Test . . . . . 213
Folder . . . . . 214
Fuel Type . . . . . 215
Fuel Type Sustainability Unit of Measure . . . . . 216
Fundraising Config . . . . . 217
Gateway Provider Payment Method Type . . . . . 218
Gen Ai Planner Bundle . . . . . 219
Generative AI Prompt Template . . . . . 220
Global Picklist . . . . . 221
Home Page Component . . . . . 222
Home Page Layout . . . . . 223
Identity Verification Proc Def . . . . . 224
Inbound Network Connection . . . . . 225
IntegrationProviderDef . . . . . 226
LearningAchievementConfig . . . . . 227
Learning Item Type . . . . . 228
Letterhead . . . . . 229
Life Science Config Category . . . . . 230
Life Science Config Record . . . . . 231
Lightning Bolt . . . . . 233
Lightning Message Channel . . . . . 233
Lightning Page . . . . . 234
Lightning Type . . . . . 235
Lightning Web Component . . . . . 236
List View . . . . . 238
Live Chat Sensitive Data Rule . . . . . 239
Loyalty Program Setup . . . . . 240
Managed Content Type . . . . . 241
Marketing App Extension . . . . . 242
Marketing App Extension Activity . . . . . 243
Market Segment Definition . . . . . 245
MktCalculatedInsightsObjectDef . . . . . 246
MktDataConnection . . . . . 247
MktDataTranObject . . . . . 248
Named Credential . . . . . 249
Object Source Target Map . . . . . 251
OcrSampleDocument . . . . . 252
OcrTemplate . . . . . 254

Outbound Network Connection . . . . . 255
Page Layout . . . . . 256
Path Assistant . . . . . 257
Payment Gateway Provider . . . . . 258
Permission Set . . . . . 259
Permission Set Groups . . . . . 260
Platform Cache . . . . . 261
Platform Event Channel . . . . . 262
Platform Event Channel Member . . . . . 262
Platform Event Subscriber Configuration . . . . . 263
Pricing Action Parameters . . . . . 264
Pricing Recipe . . . . . 265
Procedure Output Resolution . . . . . 266
Process . . . . . 267
Process Flow Migration . . . . . 267
Product Attribute Set . . . . . 268
Product Specification Type . . . . . 269
Product Specification Record Type . . . . . 270
Prompts (In-App Guidance) . . . . . 271
Quick Action . . . . . 272
Recommendation Strategy . . . . . 273
Record Action Deployment . . . . . 274
Record Alert Data Source Expression Set Definition . . . . . 275
Record Type . . . . . 276
RedirectWhitelistUrl . . . . . 277
Referenced Dashboard . . . . . 278
Registered External Service . . . . . 279
RelationshipGraphDefinition . . . . . 280
Remote Site Setting . . . . . 281
Report . . . . . 282
Report Type . . . . . 283
ServiceProcess . . . . . 284
Slack App (Beta) . . . . . 285
Service Catalog Category . . . . . 286
Service Catalog Filter Criteria . . . . . 287
Service Catalog Item Definition . . . . . 288
Service Catalog Fulfillment Flow . . . . . 289
Stationary Asset Environmental Source Record Type Configuration . . . . . 290
Static Resource . . . . . 291
Streaming App Data Connector . . . . . 292
Sustainability UOM . . . . . 293
Sustainability UOM Conversion . . . . . 294
Timeline Object Definition . . . . . 295
Timesheet Template . . . . . 297

Translation . . . 297
UI Object Relation Config . . . 298
User Access Policy . . . 299
Validation Rule . . . 301
Vehicle Asset Emissions Source Record Type Configuration . . . 302
View Definition (Beta) . . . 303
Virtual Visit Config . . . 304
Visualforce Component . . . 305
Visualforce Page . . . 306
Wave Analytic Asset Collection . . . 307
Wave Application . . . 308
Wave Component . . . 309
Wave Dataflow . . . 310
Wave Dashboard . . . 311
Wave Dataset . . . 312
Wave Lens . . . 313
Wave Recipe . . . 314
Wave Template Bundle . . . 315
Wave Xmd . . . 316
Web Store Template . . . 318
Workflow Alert . . . 318
Workflow Field Update . . . 319
Workflow Knowledge Publish . . . 320
Workflow Outbound Message . . . 321
Workflow Rule . . . 322
Workflow Task . . . 324
Behavior of Specific Metadata in First-Generation Managed Packages . . . 325
Get Access to Agentforce in Your 1GP Packaging Org . . . 326
Components Automatically Added to First-Generation Managed Packages . . . 327
Protected Components in Managed Packages . . . 330
Set Up a Platform Cache Partition with Provider Free Capacity . . . 331
Package Dependencies in First-Generation Managed Packages . . . 332
Metadata Access in Apex Code . . . 333
Permission Sets and Profile Settings in Packages . . . 333
Permission Set Groups . . . 336
Custom Profile Settings . . . 336
Protecting Your Intellectual Property . . . 337
Call Salesforce URLs Within a Package . . . 337
Develop App Documentation . . . 339
API and Dynamic Apex Access in Packages . . . 339
Connected Apps . . . 346
Package and Test Your First-Generation Managed Package . . . 346
Install a Managed Package . . . 347
Install First-Generation Managed Packages Using Metadata API . . . 350

Component Availability After Deployment . . . 351
Install Notifications for Unauthorized Managed Packages . . . 351
Resolve Apex Test Failures . . . 352
Run Apex on Package Install/Upgrade . . . 352
Run Apex on Package Uninstall . . . 356
Uninstall a First-Generation Managed Package . . . 357
Update Your First-Generation Managed Package . . . 358
Package Versions in First-Generation Managed Packages . . . 359
Create and Upload Patches in First-Generation Managed Packages . . . 360
Work with Patch Versions . . . 361
Versioning Apex Code . . . 363
Apex Deprecation Effects for Subscribers . . . 363
Publish Upgrades to First-Generation Managed Packages . . . 364
Plan the Release of First-Generation Managed Packages . . . 365
Remove Components from First-Generation Managed Packages . . . 366
Delete Components from First-Generation Managed Packages . . . 368
Modifying Custom Fields after a Package Is Released . . . 369
Manage Versions of First-Generation Managed Packages . . . 369
View Unused Components in a Managed Package . . . 370
Push Package Upgrades to Subscribers . . . 371
Manage Licenses for Managed Packages . . . 377
Get Started with the License Management App . . . 378
Lead and License Records in the License Management App . . . 381
Modify a License Record . . . 382
Refresh Licenses for a Managed Package . . . 383
Extending the License Management App . . . 383
Move the License Management App to Another Salesforce Org . . . 386
Troubleshoot the License Management App . . . 386
Best Practices for the License Management App . . . 388
Troubleshoot Subscriber Issues . . . 388
Manage Features in First-Generation Managed Packages . . . 390
Feature Parameter Metadata Types and Custom Objects . . . 391
Set Up Feature Parameters . . . 392
Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features . . . 393
Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters . . . 394
Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs . . . 395
Best Practices for Feature Management . . . 395
Considerations for Feature Management . . . 396
AppExchange App Analytics for First-Generation Managed Packages . . . 396
Enable App Analytics on Your First-Generation Managed Package . . . 397
Developing and Distributing Unmanaged Packages . . . 397
Create and Upload an Unmanaged Package . . . 397
Components Available in Unmanaged Packages . . . 398
Convert Unmanaged Packages to Managed . . . 401

# CHAPTER 1 First-Generation Managed Packages

In this chapter ...


- Why Switch to Second-Generation Managed Packaging?
- Move to 2GP: Migrate Your Managed Packages with Ease
- Register a Namespace for a First-Generation Managed Package
- Create a First-Generation Managed Package Using a UI
- Create a First-Generation Managed Package using Salesforce DX
- Components Available in First-Generation Managed Packages
- Behavior of Specific Metadata in First-Generation Managed Packages
- Package and Test Your First-Generation Managed Package
- Update Your First-Generation Managed Package
- Publish Upgrades to First-Generation Managed Packages
- Manage Licenses for Managed Packages
- Manage Features in First-Generation Managed Packages


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

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

- AppExchange App Analytics for First-Generation Managed Packages
- Developing and Distributing Unmanaged Packages

# Why Switch to Second-Generation Managed Packaging?

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

## Source-Driven Development

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

## Minimal Interaction with Salesforce Orgs

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

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

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

## API- and CLI-first Model

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

## Flexible Versioning

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

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

## One Namespace Shared Across Multiple Packages

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

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

## Declarative Dependencies

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

## Simplified Patch Versioning

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

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

## Avoid Having to Migrate Customers in the Future

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

SEE ALSO:

Second-Generation Managed Packaging Developer Guide

# Move to 2GP: Migrate Your Managed Packages with Ease

Are you still using first-generation managed packaging (1GP) for your package development work? If so, you're not alone. Many ISVs like you are looking to move to second-generation managed packaging (2GP) and take advantage of its many benefits.

If you don't have customers who are still using 1GP packages you created, you can skip this chapter and move straight to learning how to build new 2GP packages.

Package migration involves two main steps: converting an existing 1GP package into a 2GP package, and then migrating the 2GP package into a subscriber org. Package migration won't change your package's metadata, or disrrupt any subscriber data associated with your package.

About Package Conversion and Package Migration
There are two main stages involved in package migrations. It starts with package conversion and ends with package migration.

Three Phases of Package Migration Development
As you move from 1GP development to 2GP development, you'll move through three phases.

Plan Your Package Migration
Are you ready to move your subscribers to 2GP? The speed and ease of your package migration depend on a few key factors.

Before You Begin Package Migrations
If you've never created or worked with managed 2GP packages, scratch orgs, or Salesforce CLI, take some time to learn more about Salesforce DX and second-generation managed packages.

Convert Your Managed 1GP Package to 2GP
Before you convert your managed 1GP package version, ensure your development environment is set up.

Migrate Your Subscribers from 1GP to 2GP

Are you ready to migrate your converted package? To migrate a package you install the converted 2GP package into a subscriber org that already has the managed 1GP package version installed. Package migration requires that the major and minor version of the subscriber's installed package, match the major and minor version of 2GP you're installing.

Move to 2GP Package Development

To fully transition to 2GP package development, you retrieve the source files for your package's latest converted version. Then, you confirm in Setup for your packaging org that you're ready to develop and distribute your package solely using 2GP packaging.

Considerations for Package Migrations

Review these limits and considerations for converting and migrating packages.

Troubleshoot Package Conversion Failures

Here are some possible error scenarios that can occur when you convert a package.

## About Package Conversion and Package Migration

There are two main stages involved in package migrations. It starts with package conversion and ends with package migration.

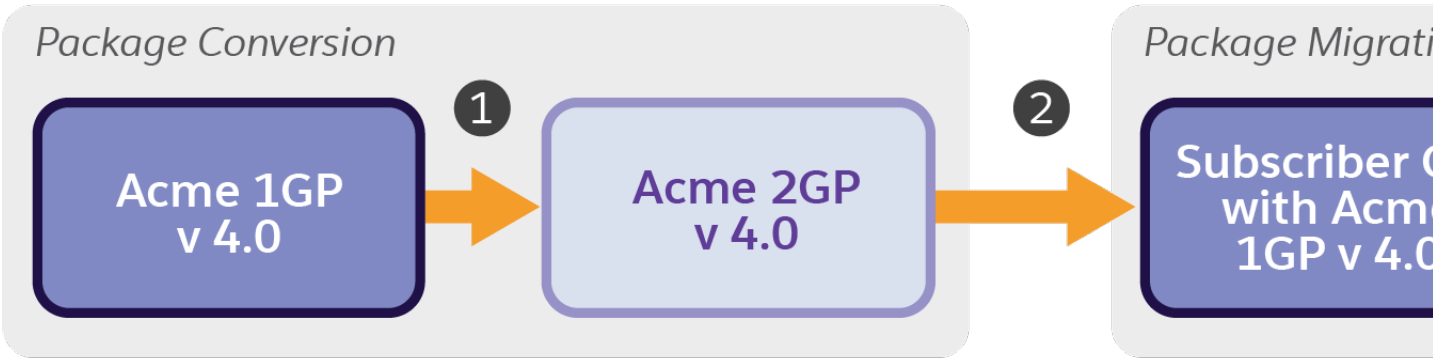
Package conversion takes a managed-released 1GP package version, and creates a 2GP package version with identical metadata. As you explore the package conversion process, you can convert a package to 2GP and still continue your 1GP package development.

After a package conversion completes:

- The original managed 1GP package is unaltered.
- You have both your original managed 1GP package version and a new managed 2GP package version.

Package migration is the process of upgrading your existing 1GP subscribers to the new 2GP package. During package migration there's no change in package metadata or customer data associated with the package.

Let's look at an example scenario.




1. The package conversion process takes the latest released major.minor of Acme's 1GP managed package (4.0) and converts it into a 2GP managed package version with the same major.minor version number (4.0).

After package conversion, Acme has a new 2GP managed package version 4.0, and the metadata in both their 1GP and 2GP package versions are identical.

2. Package migration occurs when a 1GP managed package installed in a subscriber org is migrated to the corresponding converted 2GP package.

To migrate a subscriber to the managed 2GP package version, the major.minor version of the managed 2GP and the installed managed 1GP must match. In the example in the diagram, Subscriber A has version 4.0 of Acme's 1GP package installed, and after migration

Subscriber A has version 4.0 of Acme's 2GP package installed. Moving forward, if Acme decides to release version 4.2, they must use a 2GP package version to upgrade migrated subscribers to version 4.2.

## Three Phases of Package Migration Development

As you move from 1GP development to 2GP development, you'll move through three phases.

- Phase One: Plan for and test package migration
- Phase Two: Transition from 1GP to 2GP
- Phase Three: Fully transitioned to 2GP package development

Here's a quick comparison of the three phases.

| Phase | New package versions are built using 1GP? | Convert Packages? | Migrate Packages? | New package versions built using 2GP? |
|---|---|---|---|---|
| Phase 1<br>Plan and Test | Yes. Use 1GP to create and distribute packages. | Only for testing. When converting packages for testing only, you don't need to promote the converted version. | Migrate in scratch orgs or test environments only. | No. |
| Phase 2<br>Transition from 1GP to 2GP | Yes. In phase two, even if you have migrated a subscriber. All new innovation to your package is built using 1GP. | Yes. You can begin the conversion process with the latest version of your package. For previously converted packages, you can also convert specific patch versions as needed.<br>Remember, a crucial step before updating your subscribers is to promote your converted package version to the released state. | Yes. Use the push upgrade CLI to migrate subscribers to 2GP.<br>Migration is a one-time operation per subscriber and package.<br>In phase two, after package migration, you upgrade subscribers using push upgrades to a converted version of your package. | No. Even if you have migrated some or all subscribers to a 2GP package version. During phase two any major, minor, or patch updates to this package must be built using 1GP.<br>If you make updates to the 1GP package, upload it to the managed-released state, then convert that package version, and use push upgrades to upgrade subscriber orgs. |
| Phase 3<br>Fully Transitioned from 1GP to 2GP | No. In this phase all innovation to your package is built using 2GP. | Before moving to phase three you must convert your latest package version. After a package has moved to phase three, you can no longer convert any new major, minor package versions. | At this stage, most of your subscribers are migrated to 2GP, but you can continue migrating lagging subscribers. | Yes |

## Phase One: Plan for and Test Package Migration

In phase one, you're developing, distributing, and releasing package upgrades using 1GP packaging. While you continue package development using 1GP, it's a good time to learn how package migrations work, and plan out how to migrate. See Plan Your Package Migration for more details.

During phase one you can convert your 1GP package, test the package in a scratch org, and try migrating packages installed in scratch orgs or sandboxes. During package conversion, your original managed 1GP package isn't altered. After a package conversion completes, you have both your original managed 1GP package version, and a new managed 2GP package version. Because your original 1GP isn't altered, you can continue developing that 1GP package using its associated packaging org.

If your subscribers aren't all using the same version of your package, phase one is a great time to get everyone upgraded to the same 1GP version of your package. Let's look at why this is recommended.

The package conversion command automatically selects and converts the latest major.minor package version. (We'll cover patch versions in a later section). To migrate subscribers using a lower major.minor package version, you must upgrade them to either your latest 1GP package version, or to a previously converted 1GP version.

When you migrate a package, the 1GP package version installed on a subscriber org must match the major.minor.patch version of the converted 2GP package you migrate to that org.

If you're able to get your subscribers all updated to the same package version, you only need to convert that one package version (for example version 6.0), and then you could choose to bulk migrate all subscribers to version 6.0 of the converted 2GP package.

Before moving to phase two, ensure you understand how migration works, including some best practices.

- Review: Plan Your Migration
- Review: Before you Begin Package Migration
- Review: Considerations for Package Migration
- Try package conversion and package migration in a test environment

## Phase Two: Transition from 1GP to 2GP

Phase two is a transition time.

In phase two you will convert your package, promote it to the released state, and start migrating your subscribers. The package conversion CLI command converts your latest major.minor released version of your package. To ultimately migrate all subscribers and move to phase three, any subscribers not using your latest package version must be upgraded to either your latest package version, or upgraded to a previously converted version, before they can be migrated.

If you've already tested package conversion and migration, you understand how to run those CLI commands. The thing to note about phase two is that after you migrate subscribers to your 2GP package, you'll continue to use 1GP for new package innovation. After you create a new 1GP package version, you convert that package version, promote it, and push upgrade the new converted 2GP to your subscribers. Package migration is a one-time per package and subscriber operation, but during phase two you'll likely convert additional versions of your 1GP package and use those new converted versions to upgrade previously migrated subscribers.

## Phase Three: Fully Transitioned to 2GP Package Development

In phase three, you confirm in Setup for your packaging org that you're ready to develop and distribute your package solely using 2GP packaging. When you confirm that you're ready to move to 2GP, the packaging org for your package is blocked from creating any new major or minor 1GP versions of that package. See Move to 2GP Package Development on page 15 for instructions.

You can continue to create 1GP patch versions for versions prior to moving to 2GP development. For example, if you moved package version 6.0.0 to 2GP development, you can create patch version 5.0.1 or 4.1.1 using 1GP, but you can't create patch version 6.0.1 or later using 1GP.

# Plan Your Package Migration

Are you ready to move your subscribers to 2GP? The speed and ease of your package migration depend on a few key factors.

Before you start, think about your current situation:

- How many 1GP packages do you own?
- How many subscribers are using them?
- Are your subscribers all using the same major and minor version of your package, or are you supporting multiple package versions?
- Do your packages depend on other packages?

Be sure to review the Considerations for Package Migrations and the prerequisites listed in Before you Begin Package Migrations.

## Supporting Multiple Versions of a Package?

If you currently have subscribers using multiple versions of your package, think through how to upgrade your subscribers using lower versions of your 1GP package.

When you migrate a package, the 1GP package version installed on a subscriber org must match the major.minor.patch version of the converted 2GP package you migrate to that org. When you run the Salesforce CLI package conversion command, the latest major.minor managed-released package version is converted.

If you're able to get your subscribers all updated to the same package version, you only need to convert that one package version (for example version 6.0), and then you could bulk migrate all subscribers to version 6.0 of the converted 2GP package. Package migration is significantly easier if you aren't having to convert and migrate several versions of the same package.

But if upgrading all subscribers to the same package version isn't possible for you, make sure to create an upgrade path for subscribers using lower package versions. One way to do that is to convert each 1GP package version that you promote to the managed-released state.

| Package Version | Package Built After 6/15/25? | Can package version be converted to 2GP? |
|---|---|---|
| 1.0 | No | No |
| 1.1 | No | No |
| 1.2 | No | No |
| 1.3 | Yes | Yes. If it's converted before creating 1.4, otherwise, no. |
| 1.4 | Yes | Yes. If it's converted before creating 1.5, otherwise, no. |
| 1.5 | Yes | Yes. If it's converted before creating 1.6, otherwise, no. |
| 1.6 | Yes | Yes. If it's converted before creating 1.7, otherwise, no. |

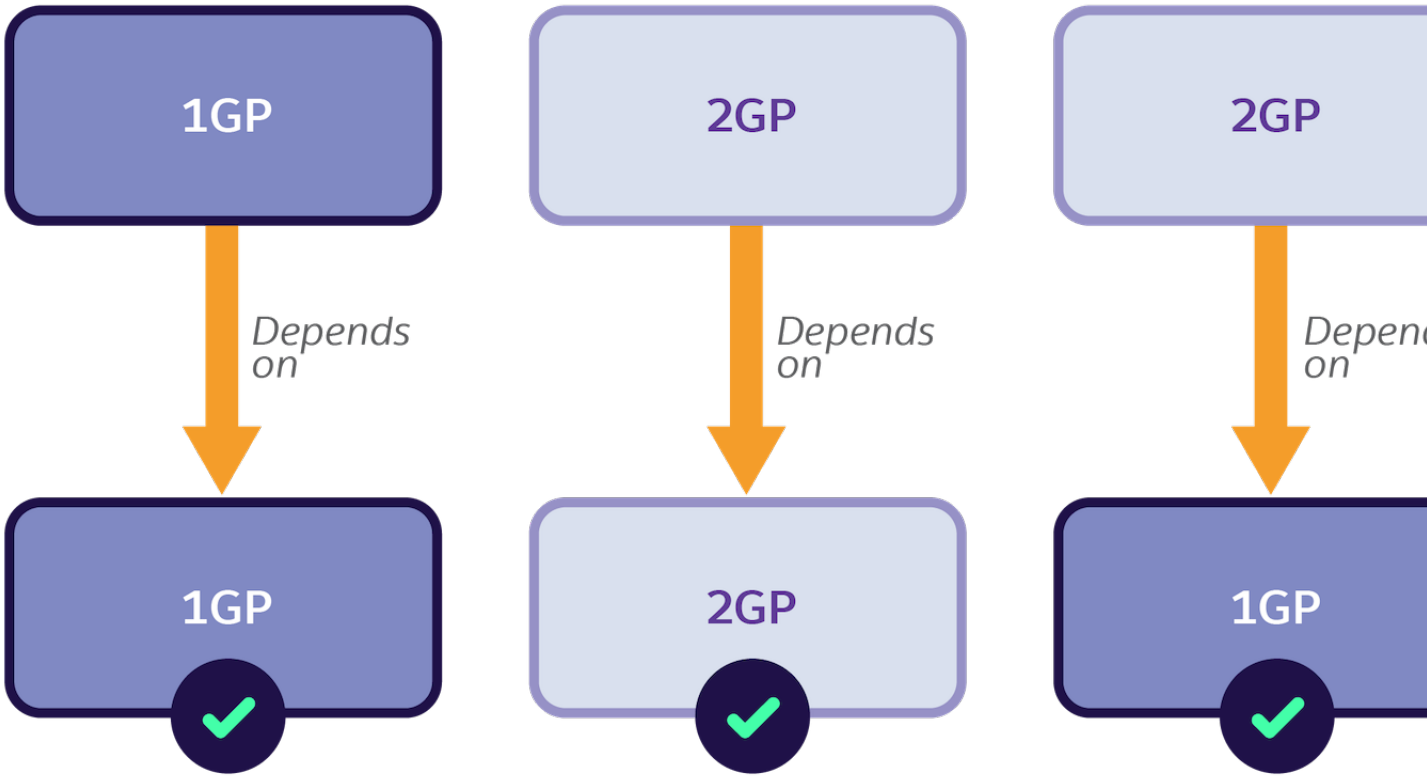
If you convert package version 1.3 before you create version 1.4, then subscribers using versions 1.0-1.2 can be upgraded to 1.3 and migrated to 2GP. Even if you aren't ready to migrate subscribers, simply having the converted 1.3 package version gives subscribers

using version 1.3 and lower an upgrade path in the future. Without the converted version 1.3, subscribers must to upgrade to version 1.4 or whatever converted versions are available.

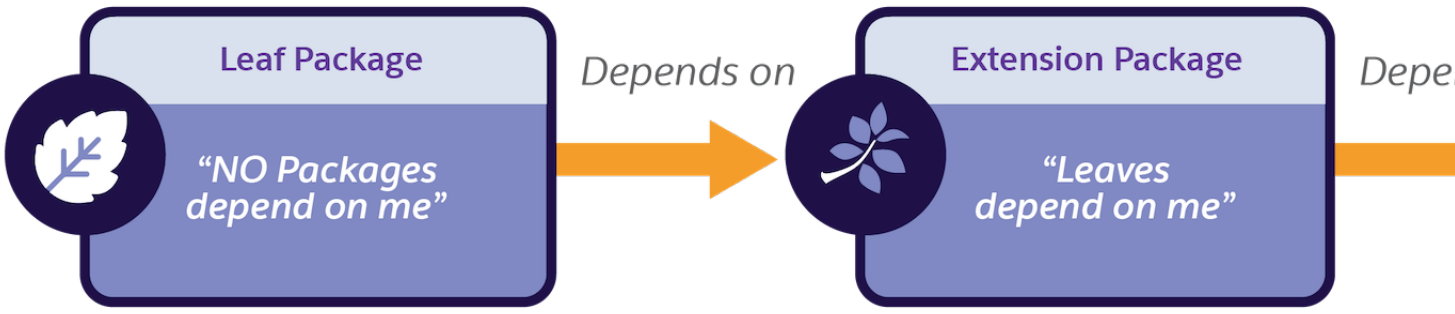
If your subscribers are using a 1GP major.minor package version that hasn't been converted, you must first upgrade them to a 1GP package version that has been converted.

## Migrating Packages with Dependencies

If the package you plan to convert depends on other packages, the order in which you migrate your package might be significant. As you may already know, a 1GP package can't depend on a 2GP package.




This limitation impacts the order in which you migrate a package with dependencies. Because 1GP packages can't depend on 2GP packages, when you migrate a 1GP package with dependencies, we recommend migrating the packages in a particular order starting with the leaf package.




In this example of a three-level package dependency, you migrate the leaf package, followed by the extension package, and end with migrating the base package.

It's possible to migrate the base package first, but this creates added complexity as 1GP extension packages can't depend on a base 2GP package. Therefore, our recommendation is to start migrating with leaf packages and then make your way down to the base package.

# Before You Begin Package Migrations

If you've never created or worked with managed 2GP packages, scratch orgs, or Salesforce CLI, take some time to learn more about Salesforce DX and second-generation managed packages.

Review Know Your Orgs for Second-Generation Managed Packages, How Scratch Orgs Fit in the Package Development Workflow, and Before You Create Second-Generation Managed Packages and ensure you have enabled:

- Dev Hub in your Partner Business Org
- Second-Generation Managed Packaging in your Dev Hub

The Second-Generation Managed Packages Trailhead module is a great resource to learn more about 2GP.

## Prerequisites for Package Conversion

Before starting package conversion, review the requirements regarding metadata support, API version, and managed-released versions. Ensure your package meets these criteria.

**Metadata Support**

If your 1GP package contains metadata that isn't currently supported in 2GP packages, the package conversion fails. You can compare the metadata in your package against the Metadata Coverage Report and ensure the metadata in your 1GP managed package is also supported in 2GP.

Alternatively, you can identify metadata issues by running the CLI `package convert` command. The `package convert` CLI command detects and reports back to you any 1GP metadata types that prevent the package from being converted to 2GP. Using the `package convert` CLI command in this way lets you quickly assess whether you have any packaged metadata that isn't supported in 2GP. During package conversion, your original managed 1GP package isn't altered. It's safe to test out the conversion command, before you are ready to migrate your packages to 2GP.

If your package contains metadata that isn't supported in managed 2GP, either remove the metadata from your package, or wait to convert the package until managed 2GP supports that metadata type.

**API Version**

You can convert first-generation managed packages created using API version 57 (Spring '23) or higher.

**Managed-Released Package Versions**

You can convert managed-released package versions only.

**User Permissions**

Package conversion and migration require the second-generation managed packaging user permissions. See Assign Second-Generation Managed Packaging User Permissions for details.

# Convert Your Managed 1GP Package to 2GP

Before you convert your managed 1GP package version, ensure your development environment is set up.

You need the latest version of Salesforce CLI installed, and the Dev Hub and second-generation managed packaging enabled in your PBO (Partner Business Org).

Package Conversion Steps

1. Run `sf update` to ensure you have the latest Salesforce CLI.
2. From your command line, navigate to the directory in which you plan to create your Salesforce DX project directory.
3. Create a Salesforce DX project. For example: `sf project generate --name myconvertedpkg`.
4. Authorize to your Dev Hub org.

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

5. Link the namespace of your managed 1GP to the Dev Hub in your Partner Business Org (PBO).
   a. Log in to your Dev Hub org.
   b. Follow the steps listed in Link a Namespace to a Dev Hub Org.

   After you convert a package, it's associated with your Dev Hub org. The association between the Dev Hub org and your package can't be changed. And you can't associate the package with a second, separate Dev Hub org. This means any subsequent package conversions you initiate for this package, must be associated with the Dev Hub org you specify the first time you convert a particular package.
6. If your package depends on standard value sets, create a seed metadata file. Then in step 7, when you run `sf package convert`, include the `--seed-metadata` flag.

   For details on setting up a seed metadata file, see Picklist Value Errors.
7. If your package has tests that depend on unpackaged metadata, add an unpackaged metadata directory to your `sfdx-project.json` file. To learn more about `sfdx-project.json` file, see Project Configuration File for a Second-Generation Managed Package.

   Example:

   ```
   "packageDirectories": [
       {
           "path": "force-app",
           "package": "TV_unl",
           "versionName": "ver 2.1",
           "default": true,
           "unpackagedMetadata": {
               "path": "my-unpackaged-directory"
            }
       },
   ]
   ```
8. From the command line, navigate to the directory on your computer that contains your Salesforce DX project, then convert your package.

Example:

```
sf package convert --installation-key mdpTest --package 033xxx --wait 20
```

To locate your 033 package ID, log in to your 1GP packaging org. From Setup, enter *Package Manager* in the Quick Find box, and then select **Package Manager**. Select the name of your managed package. After you navigate to the Package Detail page, inspect the URL in your browser's address bar. The 15-character string in the URL that begins with 033 is your package ID.

If the package version you're converting is a patch version, you must include the `--patch-version` flag when you run the `package convert` command. 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 is a patch version, but 1.1.0 isn't.

Before converting a patch version you must first convert the major.minor version of that package. Using the above example, you must convert package version 1.1.0 before converting patch version 1.1.2.

What's an Installation Key?

In the CLI command shown in step 7, an installation key with the value of `mpdTest` is specified. An installation key is a security key. By including this flag with the package conversion command, you're setting a password and requiring that password whenever anyone installs the converted 2GP package. If you prefer not to require an installation key, specify `--installation-key-bypass` when you convert your package.

Congrats, Your Package is Converted

After you've completed steps 1-7, your 1GP package is converted to 2GP. You can test your package in a scratch org.

The converted package is a beta version. Before you migrate the package you must promote it to a managed-released version. See Get Ready to Promote and Release a Second-Generation Managed Package Version for details. To promote a package version to released, you must use the `--code-coverage` flag when converting the package version. The package must also meet the code coverage requirements.

**Note:** In a scratch org, you don't need to promote the package version to migrate it.

Keep in mind that after you migrate subscribers to 2GP, all future upgrades to that package version for that subscriber must use a 2GP package version.

To fully transition your package development from your 1GP packaging org to a 2GP, CLI-based development model, see Move to 2GP Package Development on page 15.

View Details about Your Converted 2GP Package

So you've converted your 1GP package to a 2GP package. You can use one of two Salesforce CLI commands to retrieve details about your 2GP packages.

Specify Dependencies on Unpackageable Metadata

It's possible that the managed 1GP package you're converting includes tests or references to unpackageable metadata.

Test Your Converted Managed 2GP Package

Before you migrate your first subscribers, we strongly recommend you install your newly converted managed 2GP package into a new scratch org and test its functionality. If you prefer, you can also install the converted package in a sandbox, or developer edition org.

## View Details about Your Converted 2GP Package

So you've converted your 1GP package to a 2GP package. You can use one of two Salesforce CLI commands to retrieve details about your 2GP packages.

To see details about the new managed 2GP package, run:

```
sf package list --verbose
```

In the verbose command output, there's a column labeled `Converted From Package Id`. When that field is populated it indicates that the package was originally created using 1GP, and has been converted to 2GP. The 033 ID that displays in that column maps to the original 1GP package.

To see details about the new managed 2GP package version, run:

```
sf package version list
```

If the value in the Released column of the command output is `false`, this indicates that the package is a beta package version, and you must promote the package to the managed-released state before you migrate the package. See Get Ready to Promote and Release a Second-Generation Managed Package Version.

The package convert command creates a new managed 2GP package, and a new managed 2GP package version, and both are now associated with your Dev Hub org. The association between the Dev Hub org and your package can't be changed. And you can't associate the package with a second, separate Dev Hub org. This means any subsequent package conversions you initiate for this package, must be associated with the Dev Hub org you specified the first time you converted a particular package.

## Specify Dependencies on Unpackageable Metadata

It's possible that the managed 1GP package you're converting includes tests or references to unpackageable metadata.

To ensure your package conversion is successful, references to unpackageable metadata must be specified in your project configuration file. If you followed the steps in Convert Your Managed 1GP Package to 2GP, you created a Salesforce DX project in step three. When you create a Salesforce DX project, the project configuration file (`sfdx-project.json`) is automatically created.

If you think you may have these kinds of references, review Specify Unpackaged Metadata for Package Version Creation Tests and Reference Standard Value Sets by Specifying a Seed Metadata Directory. In Migrating Your Subscribers from 1GP to 2GP, we share when in the conversion workflow to specify unpackageable metadata.

## Test Your Converted Managed 2GP Package

Before you migrate your first subscribers, we strongly recommend you install your newly converted managed 2GP package into a new scratch org and test its functionality. If you prefer, you can also install the converted package in a sandbox, or developer edition org.

1. Create a scratch org. When creating your scratch org, be sure to include a scratch org definition file that includes any Salesforce features that your package depends on.

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

2. Migrate the converted package.

   To verify the 04t ID of the converted package, run `sf package version list` and review the command output. Then run the package install command and specify the 04t ID. In this scenario, the package version is being migrated, not installed.

   Example:

   ```
   sf package install --package 04t...
   ```

   Next, explore your package and test its functionality. Make sure everything is operating as expected. Note that the converted package is a beta version.

During phase one you can convert your 1GP package, install the converted package in a scratch org, and try out package migration by using packages that are installed in scratch orgs. By testing these steps in a scratch org first, you can understand how package migration works, before migrating packages used by active customers.

# Migrate Your Subscribers from 1GP to 2GP

Are you ready to migrate your converted package? To migrate a package you install the converted 2GP package into a subscriber org that already has the managed 1GP package version installed. Package migration requires that the major and minor version of the subscriber's installed package, match the major and minor version of 2GP you're installing.

After a package is migrated there's no change to the metadata or customer data associated with the package.

If your package has dependencies on other packages, the order in which you migrate the base and dependent packages is important. For more details, see Plan Your Package Migration.

## Bulk Migrate Multiple Subscribers (Beta)

To migrate multiple subscribers that all have the same package version installed, use the Salesforce CLI command to schedule a push upgrade and specify the `--migrate-to-2gp` flag. Remember the major.minor.patch version of the package you're migrating, must be identical to the major.minor.patch version installed in a subscriber org.

The subscriber package version ID (starts with 04t) that you specify when migrating a package version, must be the ID for the converted 2GP package version, not the ID for the original 1GP package version.

Example:

```
sf package push-upgrade schedule --migrate-to-2gp --package-version 04txyz
--scheduled-start-time "2024-12-06T21:00:00" --org-file upgrade-orgs.csv
```

Push upgrades for package migrations have a daily limit. Currently, you can use push upgrades to migrate 5 subscriber orgs per day.

Note: Push migrations is a beta service that is subject to the Beta Services Terms at Agreements - Salesforce.com or a written Unified Pilot Agreement if executed by Customer, and applicable terms in the Product Terms Directory. Use of this beta service is at the Customer's sole discretion.

Before migrating customers using push upgrades, please review these best practices. In particular, we strongly recommend that you work with your subscribers and follow a staggered approach, starting by migrating sandboxes first, before carrying out migrations in production.

Note: Before migrating customers using push upgrades, please review these best practices. In particular, we strongly recommend that you work with your subscribers and follow a staggered approach, starting by migrating sandboxes first, before carrying out migrations in production.

## Manually Migrate One Subscriber

If you have subscribers who don't allow push upgrades to their org, they can manually upgrade to the 2GP package. Provide them with the package installation URL and they can migrate the package.

## Push Upgrade History and Org Migrations

From time to time, Salesforce migrates production orgs from one instance to another. If your Dev Hub org is scheduled for an org migration, it's crucial to back up your historical data related to push upgrades and 2GP migrations, as this data won't be retained post-migration.

# Move to 2GP Package Development

To fully transition to 2GP package development, you retrieve the source files for your package's latest converted version. Then, you confirm in Setup for your packaging org that you're ready to develop and distribute your package solely using 2GP packaging.

Before you move to 2GP package development, review Plan Your Package Migration and confirm that you're ready to move your subscribers to 2GP.

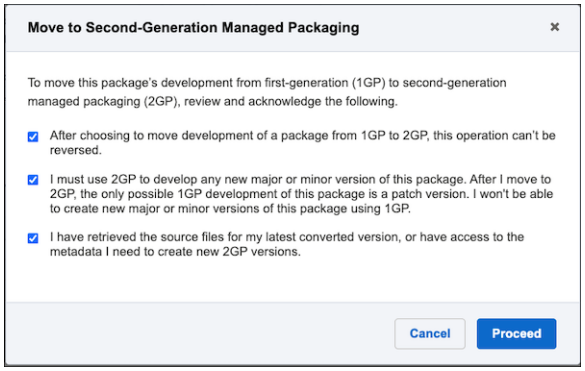
1. To retrieve the source files for your package's latest converted version, run the `sf package version retrieve` CLI command. This example retrieves the package metadata for a converted subscriber package version ID into `my-folder/` within a Salesforce DX project directory:

   ```
   sf package version retrieve  --package 04tXXX --output-dir my-folder --target-dev-hub devhub@example.com
   ```

   Note: By default, `sf package version retrieve` is available to 2GP managed packages that were converted from 1GP. To use this command with an unlocked package or a managed package created using 2GP (not converted from 1GP), set `IsDevUsePkgZipRequested` to *`true`* in the Package2VersionCreateRequest Tooling API object. If you run this command and the zip folder with the package version's source files is missing, confirm that `IsDevUsePkgZipRequested` is set to *`true`*.

2. Open the new `sfdx-project.json` file and update the `versionName`, `versionNumber`, and `ancestorVersion`.
3. From Setup in the packaging org, enter *`Package Manager`* in the Quick Find box, and then select **Package Manager**.
4. Click the package that you want to move to 2GP development.
5. On the package's detail page, click **Move to 2GP**.


6. Carefully review and acknowledge the statements in the Move to Second-Generation Managed Packaging window, then click **Proceed**.




After you click **Proceed**, you can no longer create new major or minor versions in the packaging org. To innovate on the package using 2GP, you create new package versions using the `sf package version create` CLI command.

# Considerations for Package Migrations

Review these limits and considerations for converting and migrating packages.

1. Packages converted before June 15, 2025 can't be promoted or migrated. Instead, run the convert command on your latest package version, and move forward with that package version.
2. The daily package conversion limit is the same as your Dev Hub org's scratch org allocation. See Scratch Org Allocations for Salesforce Partners for details.
3. The convert command may fail on your first attempt to convert a package. If this happens, retry again in a few minutes. This issue is most likely to occur if you create a new package for testing purposes, and immediately try to convert that package version.
4. The package conversion process performs limited validation when creating the managed 2GP package version.
   a. In some instances the limited validation could mask issues that would otherwise result in package conversion failure.
   b. You can expect to see cryptic error messages in certain scenarios. Please share those error messages with us so we can improve them.
5. During migration, the managed 1GP package version installed in the target org has the same version number as the managed 2GP migration package version.
6. Only package versions created using API version 57 (Spring '23) or above, can be converted. To promote a package version, it must have been created using API version 64 (Summer '25) or later.
7. Large packages containing 20,000+ metadata files aren't supported at this time.
8. Packages containing Data Cloud metadata can't be converted.

# Troubleshoot Package Conversion Failures

Here are some possible error scenarios that can occur when you convert a package.

## API Version

Salesforce API version 57 is the minimum API version required for converting a package from first-generation packaging (1GP) to second-generation packaging (2GP). If you encounter errors during conversion, first make sure that the package version you're converting was created using Salesforce API version 57 or later. If your package is using a lower API version, create a 1GP package version using the latest API version, then try the conversion process again.

## RecordType Errors

If the package you're converting contains a RecordType, you must specify it under object settings in a scratch org definition file. For example,

```
"objectSettings": {
    "Contact": {
    "defaultRecordType": "default"
    }
    }
```

Then specify the scratch org definition file when you run the package conversion CLI command.

```
sf package convert -p 033xx0000004Jx3 --definition-file config/project-scratch-def.json
```

## Org Shape Errors

An org shape is a stored representation of a source org's baseline setup. In the conversion process, the org shape is used to create the 2GP version of the 1GP package.

When you convert a 1GP package to 2GP, and you have not previously set up an org shape for the source 1GP packaging org, we create an org shape for you so that we're using a replica of your 1GP packaging org's baseline setup. To learn more about org shapes, see Create and Manage Org Shapes and Troubleshoot Org Shape.

If you already have an org shape set up for your 1GP packaging org, we use your existing org shape in the conversion process. When you have an existing org shape, you can sometimes experience org shape issues in the conversion process.

If you encounter an org shape error when you run the `sf package convert` command, delete the packaging org's existing org shape and then recreate the org shape. Then, try the conversion process again.

If you continue to experience org shape errors, contact Salesforce Partner Support for help.

## Picklist Value Errors

If your package references a custom picklist value that isn't included in Salesforce's StandardValueSet, a picklist error occurs during package conversion.

For more details on Standard Value Sets, StandardValueSet in *Metadata API Developer Guide*.

To resolve this error, follow these steps.

1. Create a package.xml file that follows this example. Replace the example AccountType value with the object identified in the error you received. Update the version field to the current Salesforce API version

   ```
   <?xml version="1.0" encoding="UTF-8"?>
        <Package xmlns="http://soap.sforce.com/2006/04/metadata">
          <types>
            <members>AccountType</members>
            <name>StandardValueSet</name>
          </types>
          <version>64.0</version>
        </Package>
   ```

2. Authenticate to your packaging org. The required picklist values should be present in your packaging org
3. Run the project retrieve Salesforce CLI command. Replace the variables for org alias and path, with the actual values you need.

   ```
   sf project retrieve start --manifest package.xml --target-org <org-alias> --output-dir
   <path>
   ```

   After the retrieve command completes, a new file is created in your force-app/main/default/standardValueSets directory. The name of the file will be AccountType.standardValueSet-meta.xml, where AccountType is the object you specified in your package.xml file.

4. Create a parent and child folder in your project root

   New parent folder: seed-metadata/

   New child folder: standardValueSets/

   Note: Only standardValueSet metadata type is allowed in the seed-metadata folder

5. Move the <object>.standardValueSet-meta.xml file into the seedmetadata/standardValueSets/ directory you created.
6. Run the package conversion CLI command and specify the `--seed-metadata` flag and the path to your seed metadata file.

```
sf package convert --package <033xxxx> --seed-metadata=<path-to-seedmetadata>
```

If you package encounter new errors for other picklist values, repeat these steps.

# Register a Namespace for a First-Generation Managed Package

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

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

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

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

Your namespace must:

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

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

To register a namespace:

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

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

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

# Create a First-Generation Managed Package Using a UI

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

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

USER PERMISSIONS

To create packages:
- Create AppExchange Packages

1. From Setup, in the Quick Find box, enter `Package Manager`, and then select **Package Manager**.
2. Click **New**.
3. Enter a name for your package. You can use a different name than what appears on AppExchange.
4. From the dropdown menu, select the default language of all component labels in the package.
5. (Optional) Choose a custom link from the `Configure Custom Link` field to display configuration information to installers of your app. You can select a predefined custom link to a URL that you've created for your home page layouts; see the Configure Option on page 339. The custom link appears as a **Configure** link within Salesforce on AppExchange Downloads page and app detail page of the installer's organization.
6. (Optional) In the `Notify on Apex Error` field, enter the username of the person to notify if an uncaught exception occurs in the Apex code. If you don't specify a username, all uncaught exceptions generate an email notification that is sent to Salesforce. This option is only available for managed packages.
7. (Optional) In the `Notify on Packaging Error` field, enter the email address of the person who receives an email notification if an error occurs when a subscriber's attempt to install, upgrade, or uninstall a packaged app fails. This field appears only if packaging error notifications are enabled. To enable notifications, contact Salesforce Partner Support.
8. (Optional) Enable language extension packages. (Beta)
   a. Under `Language Settings`, click **Edit**.
   b. Select `Enable Language Extension Package` and save your changes.
9. (Optional) Enter a description that describes the package. You can change this description before you upload it to AppExchange.
10. (Optional) Specify a post install script. You can run an Apex script in the subscriber organization after the package is installed or upgraded. For more information, see Running Apex on Package Install/Upgrade.
11. (Optional) Specify an uninstall script. You can run an Apex script in the subscriber organization after the package is uninstalled. For more information, see Running Apex on Package Uninstall.
12. Save your work.

### What Are Beta Versions of Managed Packages?

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

### Create a Beta Package for First-Generation Managed Packages

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

Create and Upload a First-Generation Managed Package

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

Publish Extensions to Managed Packages

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

View Package Details in First-Generation Managed Packages

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

Notifications for Package Errors

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

# What Are Beta Versions of Managed Packages?

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

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

- Beta packages can be installed in scratch, sandbox, or Developer Edition orgs, or test orgs furnished through the Environment Hub only.
- The components of a beta package are editable in the packaging org until a Managed - Released package is uploaded.
- Beta versions aren't considered major releases, so the package version number doesn't change.
- Beta packages aren't upgradeable. Because developers can still edit the components of a beta package, the Managed - Released version might not be compatible with the beta package installed. To install a new beta package or released version, first, uninstall the beta package. For more information, see Uninstall a Managed Package on page 357 and Install a Managed Package on page 347.

# Create a Beta Package for First-Generation Managed Packages

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

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

USER PERMISSIONS

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages

1. Create a package:
   a. From Setup, enter `Package Manager` in the `Quick Find` box, then select **Package Manager**.
   b. Click **New**.
   c. Enter a name for your package. You can use a different name than what appears on AppExchange.
   d. From the dropdown menu, select the default language of all component labels in the package.

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

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

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

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

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

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

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

k. Click **Save**.

2. Optionally, change the API access privileges. By default, API access is set to `Unrestricted`, but you can change this setting to further restrict API access of the components in the package.
3. Add the necessary components for your app.
   a. Click **Add Components**.
   b. From the dropdown list, choose the type of component.
   c. Select the components you want to add.
   d. Click **Add To Package**.
   e. Repeat these steps until you added all the components you want in your package.

   Note: Some related components are automatically included in the package even though they don't display in the Package Components list. For example, when you add a custom object to a package, its custom fields, page layouts, and relationships with standard objects are automatically included. For a complete list of components, see Components Automatically Added to First-Generation Managed Packages on page 327.
4. Optionally, click **View Dependencies** and review a list of components that rely on other components, permissions, or preferences within the package. To return to the Package detail page, click **Done**.
5. Click **Upload**.
6. On the Upload Package page, do the following:
   a. Enter a `Version Name`, such as *`Spring '22`*. The version name is the marketing name for a specific release of a package and allows you to create a more descriptive title for the version than just a number.
   b. Enter a `Version Number`, such as *`1.0`*. For more information on versions, see Update Your First-Generation Managed Package on page 358.
   c. Select a `Release Type` of Managed - Beta.
   d. (Optional) Enter and confirm a password to share the package privately with anyone who has the password. Don't enter a password if you want to make the package available to anyone on AppExchange and share your package publicly.

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

f. Click **Upload**.

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

# Create and Upload a First-Generation Managed Package

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

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

USER PERMISSIONS

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages

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

1. Create a package:
   a. From Setup, enter *`Package Manager`* in the `Quick Find` box, then select **Package Manager**.
   b. Click **New**.
   c. Enter a name for your package. You can use a different name than what appears on AppExchange.
   d. From the dropdown menu, select the default language of all component labels in the package.
   e. Optionally, choose a custom link from the `Configure Custom Link` field to display configuration information to installers of your app. You can select a predefined custom link to a URL or s-control that you've created for your home page layouts; see the Configure Option on page 339. The custom link displays as a **Configure** link within Salesforce on AppExchange Downloads page and app detail page of the installer's organization.
   f. Optionally, in the `Notify on Apex Error` field, enter the username of the person to notify if an uncaught exception occurs in the Apex code. If you don't specify a username, all uncaught exceptions generate an email notification that is sent to Salesforce. This option is only available for managed packages.

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

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

3. Add the necessary components for your app.
   a. Click **Add Components**.
   b. From the dropdown list, choose the type of component you want to add to your package.
      - At the top of the list, click a letter to display the contents of the sorted column that begin with that character.
      - If available, click the **Next Page** (or **Previous Page**) link to go to the next or previous set of components.
      - If available, click **fewer** or **more** at the bottom of the list to view a shorter or longer display list.
   c. Select the components you want to add.
   d. Click **Add To Package**.
   e. Repeat these steps until you added all the components you want in your package.

   Note: Some related components are automatically included in the package even if they don't display in the Package Components list. For example, when you add a custom object to a package, its custom fields, page layouts, and relationships with standard objects are automatically included.

   When you package a joined report, each block is included in the package. Although the blocks appear in the package as reports, when you click a block, an error message indicates that you have "insufficient privileges" to view the report. This error message is expected behavior. Instead, click the name of the joined report to run it.

4. Optionally, click **View Dependencies** and review a list of components that rely on other components, permissions, or preferences within the package. An entity can include such things as an s-control, a standard or custom field, or an organization-wide setting like multicurrency. Your package can't be installed unless the installer has the listed components enabled or installed. For more information on dependencies, see Understanding Dependencies on page 332. To return to the Package detail page, click **Done**.

   Note: You can't upload packages that contain any of the following:
   - Workflow rules or workflow actions (such as field updates or outbound messages) that reference record types.
   - Reports that reference record types on standard objects.

5. Click **Upload**.

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

6. On the Upload Package page, do the following:
   a. Enter a `Version Name`. As a best practice, it's useful to have a short description and the date.
   b. Enter a `Version Number` for the upload, such as *`1.0`*. The format is *`majorNumber.minorNumber`*.

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

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

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

| Upload Sequence | Type | Version Number | Notes |
|---|---|---|---|
| First upload | Managed - Beta | 1.0 | The first Managed - Beta upload. |

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

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

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

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

**d.** Change the `Description`, if necessary.

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

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

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

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

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

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

**i.** Click **Upload**.

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

Note:

- When using the install URL, the old installer is displayed by default. You can customize the installation behavior by modifying the installation URL you provide your customers.
  - To access the new installer, append the text `&newui=1` to the installation URL.
  - To access the new installer with the "All Users" option selected by default, append the additional text `&p1=full` to the installation URL.

- If you uploaded from your Salesforce production org, notify installers who want to install it in a sandbox org to replace the *login.salesforce.com* portion of the installation URL with *test.salesforce.com*.t"

After your upload is complete you can do any of the following.

- To revert the Managed - Released package version back to Managed - Beta, click **Revert to Beta**.

  Important: The Revert to Beta option is available only for the latest released package version that is not a patch version. For example, if both package versions 1.0 and 2.0 are released, the Revert to Beta option is available only on the detail page of package version 2.0. To revert the released package version to beta, the package version can't be installed in any orgs and it can't be associated with a patch development org. If the package version is installed in any subscriber orgs, uninstall the package version before reverting the version to the beta state. After you revert a package version to beta, when you create a new package version, the new package version number is the highest package version number that's in the beta state.

- To change the password option, click **Change Password** link.
- To prevent new installations of this package while allowing existing installations to continue operating, click **Deprecate**.

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

  When you deprecate a package, remember to remove it from AppExchange as well.

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

# Publish Extensions to Managed Packages

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

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

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

When working with both first-generation (1GP) and second-generation (2GP) managed packages, only certain combinations of packages are supported. See: Which Package Types Can Your Package Depend On?

EDITIONS

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

USER PERMISSIONS

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages

To publish extensions to a managed package:

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

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

3. Create a package and add your extension components. Salesforce automatically includes some related components.
4. Upload the new package that contains the extension components.

5. Proceed with the publishing process as usual. For information on creating a test drive or registering and publishing your app, go to Salesforce Partner Community.

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

# View Package Details in First-Generation Managed Packages

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

From the package detail page:

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

## View Package Details

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

| Attribute | Description |
|---|---|
| API Access | The type of access that the API and dynamic Apex that package components have. The default is **Unrestricted**, which means that all package components that access the API have the same access as the user who is logged in. Click **Enable Restrictions** or **Disable Restrictions** to change the API and dynamic Apex access permissions for a package. |
| Created By | The name of the developer that created this package, including the date and time. |
| Description | A description of the package. |
| Language | The language used for the labels on components. The default value is your user language. |
| Last Modified By | The name of the last user to modify this package, including the date and time. |
| Notify on Apex Error | The username of the person who receives email notifications when an exception occurs in Apex that isn't caught by the code. If you don't specify a username, all uncaught exceptions send an email notification to Salesforce. Available only for managed packages.<br>Apex can be packaged only from Developer, Enterprise, Unlimited, and Performance Edition orgs. |
| Notify on Packaging Error | The email address of the person who receives email notifications if an error occurs when a subscriber's attempt to install, upgrade, |

| Attribute | Description |
| --- | --- |
| | or uninstall a packaged app fails. This field appears only if packaging error notifications are enabled. To enable notifications, contact your Salesforce representative. |
| `Package Name` | The name of the package, provided by the publisher. |
| `Push Upgrade Exclusion List` | A comma-separated list of org IDs to exclude when you push a package upgrade to subscribers. |
| `Post Install Script` | The Apex code that runs after this package is installed or upgraded. For more information, see Run Apex on Package Install/Upgrade on page 352. |
| `Type` | Indicates whether it's a managed or unmanaged package. |
| `Uninstall Script` | The Apex code that runs after this package is uninstalled. For more information, see Run Apex on Package Uninstall on page 356. |

## View Package Components

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

Note: Some related components are automatically included in the package even though they aren't displayed in the Package Components list. For example, when you add a custom object to a package, its custom fields, page layouts, and relationships with standard objects are included. For a list of components that Salesforce automatically includes, see Components Automatically Added on page 327.

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

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

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

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

## View Version History

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

To manage an uploaded package version, click the version number of a listed upload. For more information, see Manage Versions of First-Generation Managed Packages on page 369.

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

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

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

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

## View Patch Development Orgs

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

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

| Attribute | Description |
|---|---|
| `Action` | Lists the actions that you can perform on a patch development org. The possible actions are:<br>• **Login**—Log in to your package version.<br>• **Reset**—Emails a temporary password for your patch development org. |

| Attribute | Description |
| --- | --- |
| `Administrator Username` | The login associated with the patch org. |
| `Patching Major Release` | The package version number that you're patching. |

# Notifications for Package Errors

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

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

Errors can happen with these package operations:

- Installation
- Upgrade
- Push upgrade
- Uninstallation

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

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

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

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

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

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

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

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

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

## Set the Notification Email Address

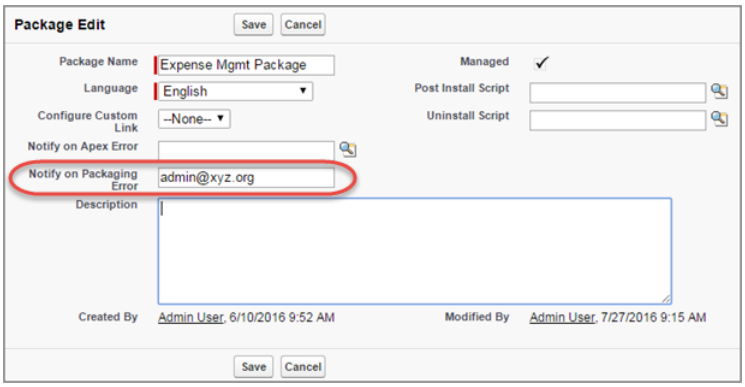
Specify which address to email when a package installation, upgrade, or uninstallation fails.

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

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

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

**Notifications for Package Errors Configured in a Partner Org**




# Create a First-Generation Managed Package using Salesforce DX

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

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

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

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

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.

### 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.

### 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.

### Build and Release Your App with Managed Packages

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

### Packaging Checklist

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

### Deploy the Package Metadata to the Packaging Org

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

### Create a Beta Version of Your App

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

### Install the Package in a Target Org

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

### Create a Managed Package Version of Your App

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

### View Information About a Package

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

# Enable Dev Hub and Second-Generation Managed Packaging

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

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.
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.

- When a trial or Developer Edition org expires, you lose access to all packages associated with that Dev Hub org.
- 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.

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 Records permissions can create scratch orgs using an existing org shape.

- Users with the Salesforce Limited Access - Free license and View All Records 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.

## 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.

# 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.

### 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.

SEE ALSO:

*Salesforce CLI Setup Guide*

*Salesforce DX Developer Guide*

## 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.
- Authorize the Dev Hub org.
- Create a Salesforce DX Project.

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 can be a Partner Developer, Partner Enterprise, Developer, or Enterprise 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 MyNamespacedScratchOrg --set-default --target-dev-hub MyDevHubOrg
```

## Scratch Orgs with both Agentforce and Data Cloud

For some use cases such as prompt templates that use RAG, Retrievers, or 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.

- Create Agents manually in the scratch org.
- To use prompt templates with your Agent Actions, assign prompt template permissions.

SEE ALSO:

- Get Access to Agentforce in Your 1GP Packaging Org
- *Trailhead*: Quick Start: Build Your First Agent with Agentforce
- *Salesforce Help*: Agentforce Agents
- *Salesforce Help*: The Building Blocks of Agents
- *Salesforce Help*: Customize Your Agents with Topics and Actions
- *Salesforce Help*: Considerations for Agents
- *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.

# Build and Release Your App with Managed Packages

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

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

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

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

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

SEE ALSO:

Link a Namespace to a Dev Hub Org

Retrieve Source from an Existing Managed Package

# Packaging Checklist

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

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

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

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

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

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

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

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

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

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

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

# Deploy the Package Metadata to the Packaging Org

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

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

1. Convert from source format to the metadata format.
   ```
   sf project convert source --output-dir mdapi_output_dir --package-name
   managed_pkg_name
   ```
   Create the output directory in the root of your project, not in the package directory. If the output directory doesn't exist, it's created. Be sure to include the `--package-name` so that the converted metadata is added to the managed package in your packaging org.
2. Review the contents of the output directory.
   ```
   ls -lR mdapi_output_dir
   ```
3. Authenticate the packaging org, if needed. This example specifies the org with an alias called MyPackagingOrgAlias, which helps you refer to the org more easily in subsequent commands.
   ```
   sf org login web --alias MyPackagingOrgAlias
   ```
   You can also authenticate with an OAuth client ID: `sf org login web --client-id oauth_client_id`
4. Deploy the package metadata back to the packaging org.
   ```
   sf project deploy start --metadata-dir mdapi_output_dir --target-org me@example.com
   ```
   The `--target-org` is the username. Instead of the username, you can use `-u MyPackagingOrgAlias` to refer to your previously defined org alias. You can use other options, like `--wait` to specify the number of minutes to wait. Use the `--metadata-dir` parameter to provide the path to a zip file that contains your metadata. Don't run tests at the same time as you deploy the metadata. You can run tests during the package upload process.

   A message displays the job ID for the deployment.
5. Check the status of the deployment.

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

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

SEE ALSO:

*Salesforce CLI Command Reference*

How to Exclude Source When Syncing or Converting

# Create a Beta Version of Your App

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

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

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

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

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

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

2. Create the beta version of the package.

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

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

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

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

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

SEE ALSO:

*Salesforce CLI Command Reference*

Link a Namespace to a Dev Hub Org

# Install the Package in a Target Org

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

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

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

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

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

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

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

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

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

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

# Create a Managed Package Version of Your App

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# View Information About a Package

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

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

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

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

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

SEE ALSO:

*Salesforce CLI Command Reference*

## View All Package Versions in the Org

View the details about all package versions in the org.

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

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

SEE ALSO:

*Salesforce CLI Command Reference*

## Package IDs

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

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

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

# Components Available in First-Generation Managed Packages

Each metadata component that you include in a first-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 | **If yes:** 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.<br>**If no:** 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. |
| Subscriber Can Delete Component | **If yes:** 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.<br>**If no:** The subscriber or installer of the managed package can't delete the packaged component from their org. |
| Package Developer Can Remove Component | **If yes:** 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.<br>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.<br>To request access to this feature, log a support case in the Salesforce Partner Community.<br>**If no:** After the package that contains the component is promoted and released, the package developer can't remove the component in a future package version. |
| Component Has IP Protection | **If yes:** 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.<br>**If no:** The component is visible in the subscriber's org. |

# 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 First-Generation Managed Packages

Account Plan Objective Measure Calculation Definition

Represents the definition of a target object, rollup field, and logic for calculating the current value of a sales account plan objective measure.

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.

Agent Action

Represents an action, for use in Agentforce.

Agent Topic

Represents a topic, for use in Agentforce.

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.

Analytics Dashboard

Represents a Tableau Next dashboard.

Analytics Visualization

Represents a Tableau Next visualization.

Analytics Workspace

Represents a Tableau Next workspace.

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.

App Framework Template Bundle

Represents the app framework template bundle. Use these templates for Data Cloud and Tableau Next assets.

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.

### CommunicationChannelType

Represents the type of communication channel, such as WhatsApp and SMS, to use for referral promotions.

### 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.

### Context Definition

A context definition defines the relationship between the nodes and the attributes within each node. For efficient data access, users can use nodes and attributes to easily access the relevant data from the mapped data source. Various Salesforce products offer predefined context definitions based on their use case.

### 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 Field Display

Represents the CustomFieldDisplay view type assigned to product attribute custom fields.

### 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.

DataObjectBuildOrgTemplate

Represents the output objects of the components the user includes in a 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.

### Event Relay

Represents an event relay that you can use to send platform events and change data capture events from Salesforce to Amazon EventBridge.

### 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 Auth Identity Provider

Represents the external auth identity provider that obtains OAuth tokens for callouts that use named credentials.

External Client App Header

Represents the header file for an external client application configuration.

External Client App Notification Settings

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

External Client App OAuth Settings

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

External Client App Push Settings

Represents the settings configuration for the external client app's push notification 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 Field

Use ExternalDataTranField to add a field to the ExternalDataTranObject in your managed packages. ExternalDataTranObject is a Data Cloud schema object.

External Data Transport Object Template

Represents the definition of a Data Cloud schema object.

External Data Transport Object

To include a Data Cloud schema object in your managed packages, add ExternalDataTranObject.

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 Planner Bundle

Represents a planner for an agent or agent template. It's a container for all the topics and actions used to interact with a large language model (LLM).

### Generative AI Prompt Template

Represents a generative AI prompt template, for use in Agentforce.

### 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.

### 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.

### Life Science Config Category

Represents the category that a Life Sciences configuration record is organized into.

### Life Science Config Record

Represents a configuration record for Life Sciences. This object is a child of Life Science Config Category.

### 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 Type

Represents a custom Lightning type. Use this type to override the default user interface to create a customized appearance of responses on the custom agent's action input and output. Deploy this bundle to your organization to implement the overrides.

### Lightning Web Component

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

### List View

ListView 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.

### Managed Content Type

Represents the definition of custom content types for use with Salesforce CMS. Custom content types are displayed as forms with defined fields.

### Marketing App Extension

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

### 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, the trigger's running user, and parallel subscription settings.

### 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.

### Procedure Output Resolution

Represents the pricing resolution for an pricing element determined using strategy name and formula.

### 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 and 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 Plan Objective Measure Calculation Definition

Represents the definition of a target object, rollup field, and logic for calculating the current value of a sales account plan objective measure.

## 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

- Description, DeveloperName, MasterLabel, RollupType, Status, TargetField, TargetObject

Neither Package Developer or Subscriber Can Edit

- None

## More Information

**Feature Name**

Metadata Name: AccountPlanObjMeasCalcDef

Component Type in 1GP Package Manager UI: Account Plan Objective Measure Calculation Definition

**Documentation**

Sales Account Plan Objectives, Measures, and Calculation Definitions

# 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 Editv

- 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

# Agent Action

Represents an action, for use in Agentforce.

## 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 (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

- Description
- IsConfirmationRequired
- MasterLabel

Action Input Fields:

- CopilotAction.IsUserInput
- Description
- IsPII
- Properties (Inherited from invocationTarget like flows or Apex code.)
- Title (Inherited from invocationTarget like flows or Apex code.)
- Required
- Lightning.Type

Action Output Fields:

- Description
- CopilotAction.IsDisplayable
- IsPII
- CopilotAction.IsUsedByPlanner
- Properties (Inherited from invocationTarget like flows or Apex code.)
- Title (Inherited from invocationTarget like flows or Apex code.)

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- DeveloperName
- InvocationTarget
- InvocationTargetType

## More Information

**Feature Name**

Metadata Name: GenAiFunction

Component Type in 1GP Package Manager UI: Generative AI Function Definition

**Use Case**

Provide actions that customers can add to their own topics and agents.

**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:* Agentforce Agents

*Salesforce Help:* Agentforce Actions

*Metadata API Developer Guide:* GenAiFunction

# Agent Topic

Represents a topic, for use in Agentforce.

## 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

- Description
- MasterLabel
- Scope
- AiPluginUtterances
- GenAiFunctions
- GenAiPluginInstructions

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- DeveloperName
- PluginType

## More Information

**Feature Name**

Metadata Name: GenAiPlugin

Component Type in 1GP Package Manager UI: Generative AI Plugin Definition

**Use Case**

Provide topics that customers can add to their own agents. Actions can be added to topics.

**Considerations When Packaging**

Subscribers can't edit which actions are associated with a managed-installed topic. Instead, subscribers must manually create a copy of the topic and then assign actions to their copy of the topic. We're working to improve this experience.

**Documentation**

*Salesforce Help:* Agentforce Agents

*Salesforce Help:* Agentforce Topics

# 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
- MlExternalId

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).<br>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.<br>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.

# Analytics Dashboard

Represents a Tableau Next dashboard.

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 |

## 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

- Template Asset Source Name
- Template Source
- Version

## More Information

**Feature Name**

Metadata Name: AnalyticsDashboard

Component Type in 1GP Package Manager UI: Analytics Dashboard

**License Requirements**

Tableau Next Admin or Tableau Next Analyst permission sets

**Documentation**

For more information on Tableau Next dashboards, see Create Effective Dashboards With Tableau Next in *Salesforce Help*.

# Analytics Visualization

Represents a Tableau Next visualization.

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 |

## 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

- Full Name
- Is Original
- Version

## More Information

**Feature Name**

Metadata Name: AnalyticsVisualization

Component Type in 1GP Package Manager UI: Analytics Visualization

**License Requirements**

Tableau Next Admin or Tableau Next Analyst permission sets

**Documentation**

For more information on Tableau Next visualizations, see Build Insightful Visualizations in Tableau Next in *Salesforce Help*.

# Analytics Workspace

Represents a Tableau Next workspace.

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 |

## 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

- None

## More Information

**Feature Name**

Metadata Name: AnalyticsWorkspace

Component Type in 1GP Package Manager UI: Analytics Workspace

**License Requirements**

Tableau Next Admin or Tableau Next Analyst permission sets

**Documentation**

For more information on Tableau Next workspaces, see Tableau Next Workspaces in *Salesforce Help*.

# 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).<br>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

# App Framework Template Bundle

Represents the app framework template bundle. Use these templates for Data Cloud and Tableau Next 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 |

## Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- Label
- MaxAppCount

Both Package Developer and Subscriber Can Edit

- Description
- TemplateBadgeIcon

Neither Package Developer or Subscriber Can Edit

- AssetVerion
- TemplateType

## More Information

**Feature Name**

Metadata Name: AppFrameworkTemplateBundle

Component Type in 1GP Package Manager UI: App Framework Template Bundle

**Considerations When Packaging**

Data Cloud and Tableau Next assets are installed in subscriber orgs via templates using the AppFrameworkTemplateBundle. The template framework supports the data sync and orchestration needed for visualization assets, along with customizations for each org.

**License Requirements**

Tableau Included App Manager permission set

# 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
- MlDomain
- 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 | 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 EditNone
- Both Package Developer and Subscriber Can EditAll attributes
- Neither Package Developer or Subscriber Can EditNone

## 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

# CommunicationChannelType

Represents the type of communication channel, such as WhatsApp and SMS, to use for referral promotions.

## 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 |

## 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

- API Name

## More Information

**Feature Name**
Metadata Name: CommunicationChannelType

**Use Case**
Use WhatsApp as a communication channel for referral promotions.

**License Requirements**
Referral Marketing permission set license

**Documentation**
*Salesforce Help:* Communication Assets

# 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.

# Context Definition

A context definition defines the relationship between the nodes and the attributes within each node. For efficient data access, users can use nodes and attributes to easily access the relevant data from the mapped data source. Various Salesforce products offer predefined context definitions based on their use case.

## 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 doesn't contain any active context definitions. |
| 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

- Standard Context Definitions

## More Information

**Feature Name**

Metadata Name: ContextDefinition

Component Type in 1GP Package Manager UI: Context Definition

**Documentation**

*Industries Common Resources Developer Guide:* Context Definition

*Salesforce Help:* Context Service

# 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.

**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
- Resizeable
- 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 Field Display

Represents the CustomFieldDisplay view type assigned to product attribute custom 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) |
|---|---|

| | |
|---|---|
| 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: CustomFieldDisplay

**License Requirements**
A B2B Commerce or D2C Commerce license and access to Commerce objects is required.

**Documentation**
*Salesforce Help:* Create Attributes for Product Variations in Commerce Cloud

# 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

# DataObjectBuildOrgTemplate

Represents the output objects of the components the user includes in 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 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 |

## 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: DataObjectBuildOrgTemplate

Component Type in 1GP Package Manager UI: DataObjectBuildOrgTemplate

**Use Case**

Supports extension packages that reference the output of any object.

**Considerations When Packaging**

A Data Cloud feature is always packaged via a data kit. You add the data object build org 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 DataStream using ui-api or the Data Cloud App.

**Relationship to Other Components**
This isn't a top-level entity. AddDataStreamDefinition 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 `global` 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.

Note: 2GP support for Business Rules Engine Components is a pilot or beta service that is subject to the Beta Services Terms at Agreements - Salesforce.com or a written Unified Pilot Agreement if executed by Customer, and applicable terms in the Product Terms Directory. Use of this pilot or beta service is at the Customer's sole discretion.

## 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 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.

Note: 2GP support for Business Rules Engine Components is a pilot or beta service that is subject to the Beta Services Terms at Agreements - Salesforce.com or a written Unified Pilot Agreement if executed by Customer, and applicable terms in the Product Terms Directory. Use of this pilot or beta service is at the Customer's sole discretion.

## 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 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 an enhanced workspace in Salesforce CMS, including images, documents, and news articles. In Marketing Cloud, you can package the content of general and marketing workspaces, including landing pages, forms, and emails (and their associated images and branding).

**Considerations When Packaging**

Enhanced LWR sites are unsupported.

In marketing workspaces, the default data graph, personalization recommenders, personalization points, and decisions aren't included in the bundle. If the workspace includes emails with personalized content that's based on these objects, then:

- Any merge field or repeater that uses the default data graph or a personalization recommender from the source org is broken in the target org.
- Any dynamic content variations of email components are removed and only the default variations appear in the email.

**Post Install Steps**

After the package is installed, publish the workspace content to make it available to customers.

**Documentation**

*Salesforce Help:* Salesforce CMS

*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

# Event Relay

Represents an event relay that you can use to send platform events and change data capture events from Salesforce to Amazon EventBridge.

## 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

- None

Both Package Developer and Subscriber Can Edit

- `Label`
- `RelayOption`
- `State`

Neither Package Developer or Subscriber Can Edit

- `DestinationResourceName`
- `EventChannel`
- `UsageType`

## More Information

**Feature Name**
Metadata Name: EventRelayConfig

Component Type in 1GP Package Manager UI: Event Relay

**Documentation**

*Metadata API Developer Guide:* EventRelayConfig

# Explainability Action Definition

Define where the metadata for your Decision Explainer 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.

Note: 2GP support for Business Rules Engine Components is a pilot or beta service that is subject to the Beta Services Terms at Agreements - Salesforce.com or a written Unified Pilot Agreement if executed by Customer, and applicable terms in the Product Terms Directory. Use of this pilot or beta service is at the Customer's sole discretion.

## 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.

Note: 2GP support for Business Rules Engine Components is a pilot or beta service that is subject to the Beta Services Terms at Agreements - Salesforce.com or a written Unified Pilot Agreement if executed by Customer, and applicable terms in the Product Terms Directory. Use of this pilot or beta service is at the Customer's sole discretion.

## 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 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 Auth Identity Provider

Represents the external auth identity provider that obtains OAuth tokens for callouts that use named credentials.

## 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, ParameterName, ParameterType, ParameterValue, and SequenceNumber properties have the same editability as the ExternalAuthIdentityProviderParameters they're included in.

Only Package Developer Can Edit

- AuthenticationFlow
- AuthenticationProtocol
- Description
- Label

Both Package Developer and Subscriber Can Edit

- ExternalAuthIdentityProviderParameter
  - AuthorizeUrl
  - ClientAuthentication
  - Description
  - IdentityProviderOptions

- ParameterName
- ParameterType
- ParameterValue
- RefreshRequestBodyParameter
- RefreshRequestHttpHeader
- RefreshRequestQueryParameter
- SequenceNumber
- StandardExternalIdentityProvider
- TokenRequestBodyParameter
- TokenRequestHttpHeader
- TokenRequestQueryParameter
- TokenUrl
- UserInfoUrl

Neither Package Developer or Subscriber Can Edit

- FullName

## More Information

**Feature Name**

Metadata Name: ExternalAuthIdentityProvider

Component Type in 1GP Package Manager UI: External Auth Identity Provider

**Considerations When Packaging**

Though external auth identity providers are represented by metadata, the standard Metadata API can't fully expose and render sensitive information like tokens in plain text. This means that sensitive values such as client secrets aren't included in packages.

Package upgrades delete any additional custom request parameters that subscribers add after installing the package. Alert subscribers that they must recreate custom parameters.

Package developers can only create parameters and delete existing parameters. After package installation, subscribers don't receive updated parameter values from package upgrades.

**Relationship to Other Components**

A callout to an external system references a named credential, which in turn links to an external credential. For external credentials that use OAuth 2.0 authentication, external auth identity providers obtain the OAuth tokens necessary for outbound callouts.

**Documentation**

*Salesforce Help:* Named Credentials

*Named Credentials Developer Guide:* Named Credentials Packaging Guide

*Metadata API Developer Guide:* ExternalAuthIdentityProvider

# 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 Notification Settings

Represents the settings configuration for the external client app's notifications 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: ExtlClntAppNotificationSettings

**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**

ExtlClntAppNotificationSettings contains all of the packageable configurations for the External Client Apps notifications plugin.

**Documentation**

*Salesforce Help:* External Client Apps

ExtlClntAppNotificationSettings

# 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 Client App Push Settings

Represents the settings configuration for the external client app's push notification 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: ExtlClntAppPushSettings

**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.

To deploy ExtlClntAppPushSettings retrieved from the Dev Hub org, delete androidPushConfig or applePushConfig from the metadata file.

**Relationship to Other Components**

External Client App plugins like the push notification 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.

ExtlClntAppPushSettings contains all of the packageable configurations for the External Client Apps push notifcation plugin. Sensitive information, like APNS or Firebase consumer credentials that can't be packaged or added to source control, are stored in the ExtlClntAppApplePushConfig and ExtlClntAppAndroidPushConfig, respectively. Policies are saved in ExtlClntAppSamlConfigurablePolicies, which is not packaged but is generated with default values at runtime.

**Documentation**

*Salesforce Help:* External Client Apps

ExtlClntAppPushSettings

# 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

- FullName

## 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 Field

Use ExternalDataTranField to add a field to the ExternalDataTranObject in your managed packages. ExternalDataTranObject is 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) |
|---|---|
| Component Is Updated During Package Upgrade | No |
| Subscriber Can Delete Component From Org | Yes |
| 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

- Length

- Precision
- Scale
- IsDataRequired
- ExternalName
- PrimaryIndexOrder
- DateFormat
- CreationType
- MktDataTranField
- Sequence
- IsImplicitFilteringRequired
- ExtDataTranFieldTemplate
- IsCurrencyIsoCode

Both Package Developer and Subscriber Can Edit

- CustomFieldDatatypes

Neither Package Developer or Subscriber Can Edit

- None

## More Information

**Feature Name**
Metadata Name: ExternalDataTranField

**Use Case**
This component holds reference to ExternalDataTranObject metadata and represents the fields in the ExternalDataTranObject.

**License Requirements**
Data Cloud must be provisioned.

**Post Install Steps**
You must to create a data stream via ui-api or by using the Data Cloud App.

**Relationship to Other Components**
This isn't a top-level entity. Add DataStreamDefinition to pick up this entity. This entity's parent is ExternalDataTranObject.

**Documentation**
*Metadata API Developer Guide:* ExternalDataTranField

# 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 Data Transport Object

To include a Data Cloud schema object in your managed packages, add ExternalDataTranObject.

## 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 | No |
| Component Has IP Protection | Yes |

## Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- AvailabilityStatus
- CreationType
- MktDataTranObject
- ObjectCategory
- ExtDataTranObjectTemplate

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

## More Information

**Feature Name**

Metadata Name: ExternalDataTranObject

**Use Case**

ExternalDataTranObject contains specific schema event information that is used to describe events for ingestion via Data Cloud Ingestion API, Web, and Mobile connectors. This object is related to many child schema fields, ExternalDataTranField.

**License Requirements**

Data Cloud must be provisioned.

**Post Install Steps**

You must create a data stream via ui-api or by using the Data Cloud App.

**Relationship to Other Components**

This isn't a top-level entity. Add DataStreamDefinition to pick up this entity. This entity's parent is ExternalDataConnector.

**Documentation**

*Data Cloud Integration Guide:* Mobile and Web SDK Schema Quick Guide for Data Cloud

*Data Cloud Integration Guide:* Requirements for Ingestion API Schema File

*Metadata API Developer Guide:* ExternalDataTranObject

# 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 | Yes (If there are no dependencies on the External Services registration and its actions from flows or other features) |
| 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

- Description
- Label
- Schema
- Schema URL

Both Package Developer and Subscriber Can Edit

- Named Credential

Neither Package Developer or Subscriber Can Edit

- None

## 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.

**Usage Limits**

*Salesforce Help:* External Services System Limits

**Documentation**

*Metadata API Developer Guide:* ExternalServiceRegistration

*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 **Unavailable** to **Available for the Field Set** or **In the Field Set** | 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 **Available for the Field Set** or **In the Field Set** to **Unavailable** | The field is removed from the upgraded field set. |
| Changes a field from **In the Field Set** to **Available for the Field Set** (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 in a managed package or between a package developer org and a subscriber org where either org has a namespace and the other doesn't have a namespace.
- 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

*Salesforce DX Developer Guide:* Hard-Deleted Components in Unlocked 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 Planner Bundle

Represents a planner for an agent or agent template. It's a container for all the topics and actions used to interact with a large language model (LLM).

## 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

- Capabilities
- Description
- MasterLabel

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

## More Information

**Feature Name**

Metadata Name: GenAiPlannerBundle

Component Type in 2GP Package Manager UI: Generative AI Planner Bundle

**Use Case**

Represents a planner for an agent or agent template. It's a container for all the topics and actions used to interact with a large language model (LLM).

**Documentation**

*Salesforce Help:* Agentforce Agents

*Salesforce Help:* The Building Blocks of Agents

# Generative AI Prompt Template

Represents a generative AI prompt template, for use in Agentforce.

## 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

- Template Active Version

Both Package Developer and Subscriber Can Edit

- Template Description

Neither Package Developer or Subscriber Can Edit

- Prompt Template Name
- Prompt Template Version

## More Information

**Feature Name**

Metadata Name: GenAIPromptTemplate

Component Type in 1GP Package Manager UI: Generative AI Prompt Template

**Use Case**

To package prompt templates created from Prompt Builder for Generative AI use cases.

**Considerations When Packaging**

See Considerations for Packaging Prompt Templates.

**License Requirements**

Generative AI SKUs are needed to provision Prompt Builder in the org.

**Documentation**

*Metadata API Developer Guide:* GenAiPromptTemplate

# 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

# 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 Editv

- 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

# Life Science Config Category

Represents the category that a Life Sciences configuration record is organized into.

## 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 |

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

- CategoryLabel
- DeveloperName
- MasterLabel

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- CategoryType

## More Information

**Feature Name**

Metadata Name: LifeSciConfigCategory

Component Type in 1GP Package Manager UI: Life Science Config Category

**Considerations When Packaging**

When packaging the LifeSciConfigCategory component, the DeveloperName must match the Category.

**License Requirements**

Industries Life Sciences Cloud with the Life Sciences Cloud for Customer Engagement Add-on license and the Life Sciences Customer Engagement managed package.

**Relationship to Other Components**

This component defines the category of the configuration defined in a child LifeSciConfigRecord component.

**Documentation**

*Life Sciences Cloud Developer Guide*: LifeSciConfigCategory

# Life Science Config Record

Represents a configuration record for Life Sciences. This object is a child of Life Science Config 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), 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 |

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

- IsActive
- IsOrgLevel
- MasterLabel
- ParentConfigRecordId
- Type

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- DeveloperName
- LifeSciConfigCategoryId

## More Information

**Feature Name**

Metadata Name: LifeSciConfigRecord

Component Type in 1GP Package Manager UI: Life Science Config Record

**Use Case**

This component holds the configuration records for Life Sciences Cloud for Customer Engagement application.

**Considerations When Packaging**

- You must package the LifeSciConfigRecord component with its parent LifeSciConfigCategory component.
- The component must be in the inactive state.

**License Requirements**

Industries Life Sciences Cloud with the Life Sciences Cloud for Customer Engagement Add-on license and the Life Sciences Customer Engagement managed package.

**Post Install Steps**

For the configuration to work, make the component active by setting IsActive to true.

**Relationship to Other Components**

A LifeSciConfigRecord is a child of LifeSciConfigCategory, and can't exist without the parent LifeSciConfigCategory.

**Documentation**

*Life Sciences Cloud Developer Guide*: LifeSciConfigRecord

# 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

Note: You must have the Manage Prompt Templates permission to successfully package Lightning pages that reference prompt templates. Without this permission, package creation succeeds, but the prompt template isn't included in the package.

## More Information

**Feature Name**

Metadata Name: FlexiPage

**Documentation**

*Metadata API Developer Guide:* Flexipage

# Lightning Type

Represents a custom Lightning type. Use this type to override the default user interface to create a customized appearance of responses on the custom agent's action input and output. Deploy this bundle to your organization to implement the overrides.

## 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

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

## More Information

**Feature Name**

Metadata Name: LightningTypeBundle

Component Type in 1GP Package Manager UI: Lightning Type

**Documentation**

*Metadata API Developer Guide*: LightningTypeBundle

# 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
- targetConfig
- property

You can't make certain changes to `<property>` tags on a custom component that's used in a managed package or an Experience Builder site. For more information, see Considerations for configuring a component for Experience Builder in the *Lightning Web Components Developer Guide*.

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

# Managed Content Type

Represents the definition of custom content types for use with Salesforce CMS. Custom content types are displayed as forms with defined 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) |
|---|---|
| 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
- Description
- Labels

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

## More Information

**Feature Name**

Metadata Name: ManagedContentType

**Use Case**

Share or distribute custom content types for use in enhanced workspaces in Salesforce CMS.

**Considerations When Packaging**

Installed content types are available only to enhanced CMS workspaces.

To refer to an installed content type when using Connect REST API, you must use the content type's fully qualified name. Installed content types are available only to enhanced CMS workspace resources.

**Documentation**

*Metadata API Developer Guide*: ManagedContentType

*Connect REST API Developer Guide*: Enhanced CMS Workspaces Resources

*CMS Developer Guide*: Create Custom Content Type Sample

# 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

# 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 242 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 242 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)
  - HttpHeader
  - 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 OathToken (all deprecated)
  - OutboundNetworkConnectionId (deprecated)
  - Password (deprecated)
  - PrincipalType (deprecated)
  - Protocol (deprecated)
  - Username (deprecated)

Neither Package Developer or Subscriber Can Edit

- FullName

## 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

- SobjectType, SobjectProcessField, 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

# 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

# Platform Event Subscriber Configuration

Represents configuration settings for a platform event Apex trigger, including the batch size, the trigger's running user, and parallel subscription 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: | 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<br>Note: 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
- numPartitions
- partitionKey
- platformEventConsumer

Both Package Developer and Subscriber Can Edit

- user

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

*Platform Events Developer Guide*: Platform Event Processing at Scale with Parallel Subscriptions for Apex Triggers

# 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

# Procedure Output Resolution

Represents the pricing resolution for an pricing element determined using strategy name and formula.

## 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

- Active Checkbox

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

## More Information

**Feature Name**

Metadata Name: ProcedureOutputResolution

Component Type in 1GP Package Manager UI: ProcedureOutputResolution

**Use Case**

To determine the best price for a product if a pricing rule produces multiple outcomes.

**License Requirements**

Salesforce Pricing permissions

**Documentation**

*Salesforce Help:* Procedure Output Resolution

# 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 and 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: | 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: Prompt

Component Type in 1GP Package Manager UI: Prompt

**Considerations When Packaging**

For 2GP packages, ensure that the scratch org definition file includes the `GuidanceHubAllowed` and `Enablement` features. See Build Your Own Scratch Org Definition File in the *Salesforce DX Developer Guide*.

**License Requirements**

Enablement Admin permission set and Enablement permission set license are required.

**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 Editv

- 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 First-Generation Managed Packages

Learn how profiles and other metadata are handled for first-generation managed packages.

### Get Access to Agentforce in Your 1GP Packaging Org

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.

### Components Automatically Added to First-Generation Managed Packages

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

### Protected Components in Managed Packages

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

### Set Up a Platform Cache Partition with Provider Free Capacity

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

### Package Dependencies in First-Generation Managed Packages

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

### Metadata Access in Apex Code

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

### Permission Sets and Profile Settings in Packages

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

### Permission Set Groups

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

### Custom Profile Settings

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

### Protecting Your Intellectual Property

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

### Call Salesforce URLs Within a Package

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

### Develop App Documentation

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

### API and Dynamic Apex Access in Packages

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

### Connected Apps

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

## Get Access to Agentforce in Your 1GP Packaging Org

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.

To enable Agentforce Extensibility for ISVs on a 1GP packaging org, you must log a case with Salesforce Partner Support. In the case details, list your packaging org ID and request Agentforce Extensibility for ISVs be provisioned. Only 1GP development orgs created from Environment Hub can have Agentforce provisioned.

**Table 2: Packageable Agentforce Components**

| Feature Name | Component Type in Package Manager | Available in … | More Information |
|---|---|---|---|
| Agent Actions | Generative AI Function Definition | Managed 2GP and Managed 1GP | Agent Action |
| Agent Topics | Generative AI Plugin Definition | Managed 2GP and Managed 1GP | Agent Topic |

| Feature Name | Component Type in Package Manager | Available in … | More Information |
|---|---|---|---|
| Prompt Templates | Generative AI Prompt Template | Managed 2GP and Managed 1GP | Generative AI Prompt Template |

SEE ALSO:

- Get Access to Scratch Orgs That Have Agentforce
- *Salesforce Help*: Considerations for Packaging Prompt Templates
- *Trailhead*: Quick Start: Build Your First Agent with Agentforce
- *Salesforce Help*: Agentforce: Agents
- *Salesforce Help*: The Building Blocks of Agents
- *Salesforce Help*: Customize Your Agents with Topics and Actions
- *Salesforce Help*: Considerations for Agents
- *Salesforce Help*: AI Project Success

# Components Automatically Added to First-Generation Managed Packages

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

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

| When you add this component | These components are automatically added |
|---|---|
| Action | Action target object (if it's a custom object), action target field, action record type, predefined field values, action layout; and any custom fields that the action layout or predefined values refer to on the target object |
| Apex class | Custom fields, custom objects, and other explicitly referenced Apex classes, and anything else that the Apex class references directly<br>**Note:** If an Apex class references a custom label, and that label has translations, you must explicitly package the individual languages desired for those translations to be included. |
| Apex trigger | Custom fields, custom objects, and any explicitly referenced Apex classes, and anything else that the Apex trigger references directly |
| Article type | Custom fields, the default page layout |
| Compact layout | Custom fields |
| Custom app | Custom tabs (including web tabs), documents (stored as images on the tab), documents folder, asset files |
| Custom button or link | Custom fields and custom objects |
| Custom field | Custom objects |
| Custom home page layouts | Custom home page components on the layout |

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

| When you add this component | These components are automatically added |
| --- | --- |
| Email template (Lightning) created in Email Template Builder | • Custom object<br>• Custom field references (in Handlebars Merge Language syntax)<br>• Enhanced folder (except the default public and private folders)<br>• Inline images referencing Salesforce Files<br>• Attachments referencing Salesforce Files<br>• The associated FlexiPage<br>These items aren't included and can't be added to a package:<br>• Enhanced letterhead<br>• CMS files (Account Engagement only) |
| External Credential | Permission set and authentication provider<br>Note: External credentials that use the Oauth 2.0 authentication protocol must reference an authentication provider to capture the details of the authorization endpoint. If you add an external credential that references an authentication provider, the authentication provider is added to the package. See Authentication Providers for information on which elements of an authentication provider are and aren't packageable. |
| Field set | Any referenced fields |
| Lightning page | All Lightning resources referenced by the page, such as record types, actions, custom components, events, and interfaces. Custom fields, custom objects, list views, page layouts, Visualforce pages, and Apex classes referenced by the components on the page. |
| Lightning page tab | Lightning page |
| Flow | Custom objects, custom fields, Apex classes, and Visualforce pages |
| Folder | Everything in the folder |
| Lightning application | All Lightning resources referenced by the application, such as components, events, and interfaces. Custom fields, custom objects, list views, page layouts, and Apex classes referenced by the application. |
| Lightning component | All Lightning resources referenced by the component, such as nested components, events, and interfaces. Custom fields, custom objects, list views, page layouts, and Apex classes referenced by the component. |
| Lightning event | Custom fields, custom objects, list views, and page layouts |
| Lightning interface | Custom fields, custom objects, list views, and page layouts |
| Lightning web component | All Lightning web component resources referenced by the component, such as nested components and modules. Custom fields, custom objects, list views, page layouts, and Apex classes referenced by the component |
| Named Credential | External credential; for legacy named credentials, an authentication provider |
| Page layout | Actions, custom buttons, custom links, s-controls, and Visualforce pages |

| When you add this component | These components are automatically added |
|---|---|
| Permission set | Any custom permissions, external data sources, Visualforce pages, record types, and Apex classes that are assigned in the permission set |
| Record type | Record type mappings, compact layout |
| Report | Folder, custom fields, custom objects, custom report types, and custom s-controls |
| Reporting Snapshot | Reports |
| S-control | Custom fields and custom objects |
| Translation | Translated terms for the selected language on any component in the package |
| Validation rule | Custom fields (referenced in the formula) |
| Visualforce home page component | Associated Visualforce page |
| Visualforce pages | Apex classes that are used as custom controllers, Visualforce custom components, and referenced field sets |
| Workflow rule | All associated workflow alerts, field updates, outbound messages, and tasks; also, if the workflow rule is designed for a custom object, the custom object is automatically included |

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

## Protected Components in Managed Packages

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

Developers can mark these components as protected in managed packages.

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

## Considerations for Protected Custom Objects in Subscriber Sandboxes

When a subscriber creates 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.

# Package Dependencies in First-Generation Managed Packages

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

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

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

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

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

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

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

**EDITIONS**

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

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

**USER PERMISSIONS**

To upload packages:
- Upload AppExchange Packages

To view Visualforce dependencies:
- Developer Mode

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

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

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

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

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

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

## Metadata Access in Apex Code

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

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

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

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

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

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

Salesforce verifies the notice during the security review.

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

## Permission Sets and Profile Settings in Packages

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

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

| Behavior | Permission Sets | Profile Settings |
|---|---|---|
| What permissions and settings are included? | • Assigned custom apps<br>• Custom object permissions<br>• External object permissions<br>• Custom field permissions<br>• Custom metadata types permissions<br>• Custom permissions<br>• Custom settings permissions | • Assigned custom apps<br>• Assigned connected apps<br>• Tab settings<br>• Page layout assignments<br>• Record type assignments<br>• Custom field permissions<br>• Custom metadata type permissions |

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 |
| --- | --- | --- |
| | • Custom tab visibility settings<br>• Apex class access<br>• Visualforce page access<br>• External data source access<br>• Record types<br>Note: Although permission sets include standard tab visibility settings, these settings can't be packaged as permission set components.<br>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. | • Custom object permissions<br>• Custom permissions<br>• Custom settings permissions<br>• External object permissions<br>• Apex class access<br>• Visualforce page access<br>• External data source access |
| 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: |

| Behavior | Permission Sets | Profile Settings |
| --- | --- | --- |
| | | • Fields (CustomField)<br>• Objects (CustomObject),<br>• Tabs (CustomTab)<br>• Apps (CustomApplication)<br>• Apex classes (ApexClass)<br>• Apex pages (ApexPage)<br>• Layouts (Layout)<br>• Record types (RecordType)<br>• Custom permissions (CustomPermission)<br>• Custom settings (CustomSetting)<br>• Custom metadata types (CustomMetadata) |
| What are the user license requirements? | A permission set is only installed if the subscriber org has at least one user license that matches the permission set. For example, permission sets with the Salesforce 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.<br>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. | None. In a subscriber org, the installation overrides the profile settings, not their user licenses. |
| How are they assigned to users? | Subscribers must assign packaged permission sets after installing the package. | Profile settings are applied to existing profiles. |
| Can permission sets in an extension package grant access to objects installed in a base package? | A permission set in the extension package can't modify access permissions for either the parent objects in the base package or the associated child objects in the extension package. | Same behavior as for permission sets. |

## Best Practices

- If users need access to apps, standard tabs, page layouts, and record types, don't use permission sets as the sole permission-granting model for your app.

- Create packaged permission sets that grant access to the custom components in a package, but not standard Salesforce components.

## Permission Set Groups

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

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

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

Also:

- You can't add permission sets constrained by a permission set license to managed or unmanaged packages.
- You can only package permissions for metadata that's included in your package.
- You can add or remove permission sets in permission set groups as part of a package upgrade. Subscribers can also modify the permission set groups by muting permissions or adding or removing local permissions sets. Subscribers can't remove included permission sets from the permission set groups in the managed package.

SEE ALSO:

*Salesforce Help*: Create a Permission Set Group

*Salesforce Help*: Permission Set Groups Considerations

## Custom Profile Settings

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

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

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

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

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

# Protecting Your Intellectual Property

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

To protect your intellectual property, consider the following:

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

# Call Salesforce URLs Within a Package

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

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

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

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

## Use the My Domain Login URL for Logins

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

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

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

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

## Use Relative URLs

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

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

## Generate Hostnames for Full URLs

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

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

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

//Get the Visualforce hostname
String vfHostname = DomainCreator.getVisualforceHostname(packageName);

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

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

## Get Part of a Domain

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

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

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

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

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

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

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

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

# Develop App Documentation

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

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

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

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

# API and Dynamic Apex Access in Packages

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

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

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

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

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

## Considerations for API and Dynamic Apex Access in Packages

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

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

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

- API access in a package overrides the following user permissions:
  - Author Apex
  - Customize Application
  - Edit HTML Templates
  - Edit Read Only Fields
  - Manage Billing
  - Manage Call Centers
  - Manage Categories
  - Manage Custom Report Types
  - Manage Dashboards
  - Manage Letterheads
  - Manage Package Licenses
  - Manage Public Documents
  - Manage Public List Views
  - Manage Public Reports
  - Manage Public Templates
  - Manage Users
  - Transfer Record
  - Use Team Reassignment Wizards
  - View Setup and Configuration
  - Weekly Export Data
- If Read, Create, Edit, and Delete access aren't selected in the API access setting for objects, users don't have access to those objects from the package components, even if the user has the Modify All Data and View All Data permissions.
- A package with Restricted API access can't create users.
- Salesforce denies access to Web service and executeanonymous requests from an AppExchange package that has Restricted access.

The following considerations also apply to API access in packages:

- Workflow rules and Apex triggers fire regardless of API access in a package.
- If a component is in more than one package in an organization, API access is unrestricted for that component in all packages in the organization regardless of the access setting.
- If Salesforce introduces a new standard object after you select restricted access for a package, access to the new standard object isn't granted by default. You must modify the restricted access setting to include the new standard object.
- When you upgrade a package, changes to the API access are ignored even if the developer specified them, which ensures that the administrator installing the upgrade has full control. Installers must carefully examine the changes in package access in each upgrade

during installation and note all acceptable changes. Then, because those changes are ignored, the administrator must manually apply any acceptable changes after installing an upgrade.

- S-controls are served by Salesforce and rendered inline in Salesforce. Because of this tight integration, there are several means by which an s-control in an installed package could escalate its privileges to the user's full privileges. In order to protect the security of organizations that install packages, s-controls have the following limitations:
  - For packages you're developing (that is, not installed from AppExchange), you can only add s-controls to packages with the default `Unrestricted` API access. After a package has an s-control, you can't enable `Restricted` API access.
  - For packages you've installed, you can enable access restrictions even if the package contains s-controls. However, access restrictions provide only limited protection for s-controls. Salesforce recommends that you understand the JavaScript in an s-control before relying on access restriction for s-control security.
  - If an installed package has `Restricted` API access, upgrades are successful only if the upgraded version doesn't contain any s-controls. If s-controls are present in the upgraded version, you must change the currently installed package to `Unrestricted` API access.

Manage API and Dynamic Apex Access in Packages

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

Configure Default Package Versions for API Calls

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

About the Partner WSDL

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

Generate an Enterprise WSDL with Managed Packages

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

Work with Services Outside of Salesforce

## Manage API and Dynamic Apex Access in Packages

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

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

USER PERMISSIONS

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

- Create AppExchange packages

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

- Download AppExchange packages

## Setting API and Dynamic Apex Access in Packages

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

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

   Use the API Access field to:

   - **Enable Restrictions**— This option is available only if the current setting is Unrestricted. Select this option if you want to specify the dynamic Apex and API access that package components have to standard objects in the installer's organization. When you select this option, the Extended Object Permissions list is displayed. To enable access for each object in the list, select the Read, Create, Edit, or Delete checkboxes. This selection is disabled in some situations. Click **Save** when finished. For more information about choosing the Restricted option, including information about when it's disabled, see Considerations for API and Dynamic Apex Access in Packages on page 339.
   - **Disable Restrictions**—This option is available only if the current setting is Restricted. Select this option if you don't want to restrict the Apex and API access privileges that the components in the package have to standard and custom objects. This option gives all the components in the package the same API access as the user who is logged in. For example, if a user can access accounts, an Apex class in the package that accesses accounts would succeed when triggered by that user.
   - **Restricted**—Click this link if you already restricted API access and wish to edit the restrictions.

## Accepting or Rejecting API and Dynamic Apex Access Privileges during Installation

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

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

## Changing API and Dynamic Apex Access Privileges After Installation

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

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

   Use the API Access field to:

   - **Enable Restrictions**— This option is available only if the current setting is Unrestricted. Select this option if you want to specify the dynamic Apex and API access that package components have to standard objects in the installer's organization. When you select this option, the Extended Object Permissions list is displayed. To enable access for each object in the list, select the Read, Create, Edit, or Delete checkboxes. This selection is disabled in some situations. Click **Save** when finished. For more information about choosing the Restricted option, including information about when it's disabled, see Considerations for API and Dynamic Apex Access in Packages on page 339.

- **Disable Restrictions**—This option is available only if the current setting is `Restricted`. Select this option if you don't want to restrict the Apex and API access privileges that the components in the package have to standard and custom objects. This option gives all the components in the package the same API access as the user who is logged in. For example, if a user can access accounts, an Apex class in the package that accesses accounts would succeed when triggered by that user.
- **Restricted**—Click this link if you have already restricted API access and wish to edit the restrictions.

## Configure Default Package Versions for API Calls

EDITIONS

Available in: Salesforce Classic

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

USER PERMISSIONS

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

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

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

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

You can specify the default package versions for enterprise API and partner API calls. The enterprise WSDL is for customers who want to build an integration with their Salesforce organization only. It's strongly typed, which means that calls operate on objects and fields with specific data types, such as `int` and `string`. The partner WSDL is for customers, partners, and ISVs who want to build an integration that can work across multiple Salesforce organizations, regardless of their custom objects or fields. It is loosely typed, which means that calls operate on name-value pairs of field names and values instead of specific data types.

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

1. The PackageVersionHeader SOAP header.
2. The SOAP endpoint contains a URL with a format of *serverName*`/services/Soap/c/`*api_version*`/`*ID* where *api_version* is the version of the API, such as 65.0, and *ID* encodes your package version selections when the enterprise WSDL was generated.
3. The default enterprise package version settings.

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

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

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

To configure default package versions for API calls:

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

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

## About the Partner WSDL

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

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

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

## Generate an Enterprise WSDL with Managed Packages

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

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

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

EDITIONS

Available in: Salesforce Classic

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

USER PERMISSIONS

To download a WSDL:
- Customize Application

Note: This is only available to first-generation managed packages.

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

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

6. On your computer, import the local copy of the WSDL document into your development environment.

Note the following in your generated enterprise WSDL:

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

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

## Work with Services Outside of Salesforce

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

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

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

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

### Provisioning a Service External to Salesforce

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

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

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

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

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

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

- If your application requires asynchronous updates after a user session has expired, dedicate a distinct administrator user license for this.

## Connected Apps

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

Note: Connected apps creation is restricted as of Spring '26. You can continue to use existing connected apps during and after Spring '26. However, we recommend using external client apps instead. If you must continue creating connected apps, contact Salesforce Support. See New connected apps can no longer be created in Spring '26 for more details.

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

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

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

### More Resources

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

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

# Package and Test Your First-Generation Managed Package

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

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

Install a Managed Package

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

Install First-Generation Managed Packages Using Metadata API

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

Component Availability After Deployment

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

Install Notifications for Unauthorized Managed Packages

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

Resolve Apex Test Failures

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.

Run Apex on Package Uninstall

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

Uninstall a First-Generation Managed Package

You can uninstall a first-generation managed package from an org using the Setup UI. When you uninstall a first-generation managed package, all components in the package, including any deprecated components that were previously associated with the package, are deleted from the org.

# Install a Managed Package

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

## Pre-Installation

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

## Default Installation

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

## Custom Installation

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

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

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

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

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

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

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

     **Install for Admins Only**

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

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

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

     **Install for All Users**

     Specifies the following settings on all internal custom profiles.

     - Object permissions— `Read`, `Create`, `Edit`, and `Delete` enabled
     - Field-level security—set to visible and editable for all fields
     - Apex classes—enabled
     - Visualforce pages—enabled
     - App settings—enabled

- Tab settings—determined by the package developer
- Page layout settings—determined by the package developer
- Record Type settings—copied from admin profile

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

**Install for Specific Profiles...**

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

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

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

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

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

## Post-Installation Steps

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

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

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

- Configure components in the package as required.

# Install First-Generation Managed Packages Using Metadata API

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

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

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

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

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

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

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

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

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

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

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

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

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

The default value is `false`.

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

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

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

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

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

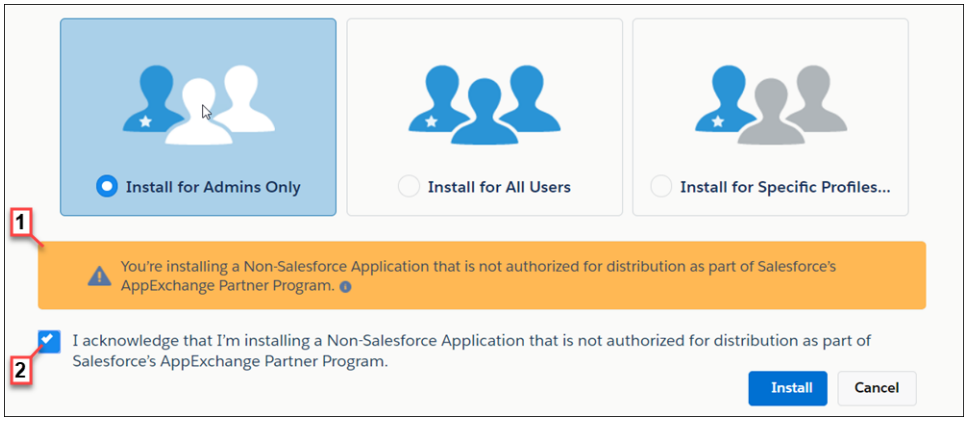
# Component Availability After Deployment

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

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

# Install Notifications for Unauthorized Managed Packages

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




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

The notification displays when a managed package:

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

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

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

# Resolve Apex Test Failures

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

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

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

# Run Apex on Package Install/Upgrade

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

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

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

The post install script has the following additional properties.

- It can initiate batch, scheduled, and future jobs.
- It can't access Session IDs.
- It can only perform callouts using an async operation. The callout occurs after the script is run and the install is complete and committed.
- It can't call another Apex class in the package if that Apex class uses the `with sharing` 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 |
|---|---|---|---|
| `compareTo` | System.Version *`version`* | Integer | Compares the current version with the specified version and returns one of the following values:<br>• Zero if the current package version is equal to the specified package version<br>• An Integer value greater than zero if the current package version is greater than the specified package version<br>• An Integer value less than zero if the current package version is less than the specified package version<br>If a two-part version is being compared to a three-part version, the patch number is ignored |

| Method | Arguments | Return Type | Description |
|---|---|---|---|
| | | | and the comparison is based only on the major and minor numbers. |
| `major` | | Integer | Returns the major package version of the calling code. |
| `minor` | | Integer | Returns the minor package version of the calling code. |
| `patch` | | Integer | Returns the patch package version of the calling code or `null` if there's no patch version. |

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

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

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

## Example of a Post Install Script

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

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

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

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

      User u = [Select Id, Email from User where Id =:context.installerID()];
      String toAddress= u.Email;
```

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

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

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

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

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

## Specifying a Post Install Script

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

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

# Run Apex on Package Uninstall

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

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

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

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

How Does an Uninstall Script Work?

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

Example of an Uninstall Script

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

Specifying an Uninstall Script

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

## How Does an Uninstall Script Work?

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

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

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

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

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

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

## Example of an Uninstall Script

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

- Inserts an entry in the feed describing which user did the uninstall and in which organization

- Creates and sends an email message confirming the uninstall to that user

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

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

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

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

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

## Specifying an Uninstall Script

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

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

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

# Uninstall a First-Generation Managed Package

You can uninstall a first-generation managed package from an org using the Setup UI. When you uninstall a first-generation managed package, all components in the package, including any deprecated components that were previously associated with the package, are deleted from the org.

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

When you uninstall packages, consider the following:

- If you're uninstalling a package that includes a custom object, all components on that custom object are also deleted. Deleted items include custom fields, validation rules, custom buttons, and links, workflow rules, and approval processes.
- You can't uninstall a package whenever a component not included in the uninstall references any component in the package. For example:
  - When an installed package includes any component on a standard object that another component references, Salesforce prevents you from uninstalling the package. An example is a package that includes a custom user field with a workflow rule that gets triggered when the value of that field is a specific value. Uninstalling the package would prevent your workflow from working.
  - When you've installed two unrelated packages that each include a custom object and one custom object component references a component in the other, you can't uninstall the package. An example is if you install an expense report app that includes a custom user field and create a validation rule on another installed custom object that references that custom user field. However, uninstalling the expense report app prevents the validation rule from working.
  - When an installed folder contains components you added after installation, Salesforce prevents you from uninstalling the package.
  - When an installed letterhead is used for an email template you added after installation, Salesforce prevents you from uninstalling the package.
  - When an installed package includes a custom field that's referenced by Einstein Prediction Builder or Case Classification, Salesforce prevents you from uninstalling the package. Before uninstalling the package, edit the prediction in Prediction Builder or Case Classification so that it no longer references the custom field.
- You can't uninstall a package that removes all active business and person account record types. Activate at least one other business or person account record type, and try again.
- You can't uninstall a package if a background job is updating a field added by the package, such as an update to a roll-up summary field. Wait until the background job finishes, and try again.

# Update Your First-Generation Managed Package

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

### Package Versions in First-Generation Managed Packages

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

### Create and Upload Patches in First-Generation Managed Packages

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

### Work with Patch Versions

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

Versioning Apex Code

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

Apex Deprecation Effects for Subscribers

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

# Package Versions in First-Generation Managed Packages

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

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

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

**Major Release**

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

**Patch Release**

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

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

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

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

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

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

# Create and Upload Patches in First-Generation Managed Packages

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

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

To enable patch versioning, log a case in the Salesforce Partner Community and request patch versioning be enabled in the org where you created the namespace for the package. Patch versioning is available only to packages that have passed AppExchange security review.

EDITIONS

Available in: **Developer** Edition

USER PERMISSIONS

To push an upgrade or create a patch development org
- Upload AppExchange Packages

To create a patch version:

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

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

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

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

Development in a patch development org is restricted.

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

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

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

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

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

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

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

# Work with Patch Versions

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

Note: To enable patch versioning, contact Salesforce Partner Support and request patch versioning be enabled in the org where you created the namespace for the package. Patch versioning is available only to packages that have passed AppExchange security review.

Subscribers can install patch upgrades just like they would any other package version. However, you can also distribute a patch by using push upgrades.

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

## Patch Development Organizations

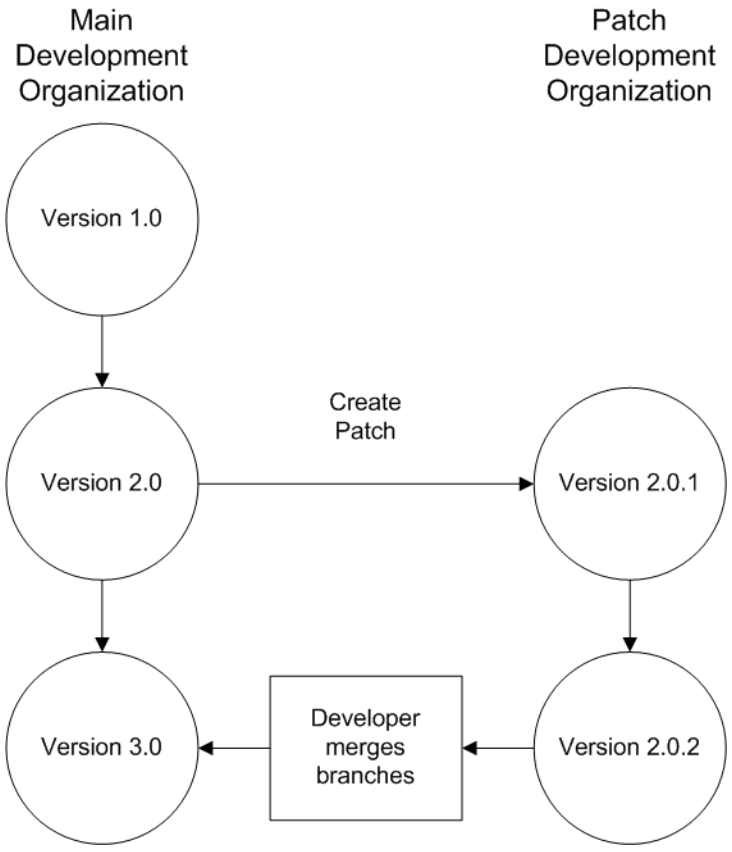
Every patch is developed in a patch development org, which is the org where patch versions are developed, maintained, and uploaded. To start developing a patch, create a package version. See Create and Upload Patches in First-Generation Managed Packages. Patch development orgs are necessary to permit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

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

## Integrating Patch Development

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

**Patch Development Workflow**




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

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

# Versioning Apex Code

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

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

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

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

    for (Opportunity o : Trigger.new){

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

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

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

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

# Apex Deprecation Effects for Subscribers

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

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

| Package Developer Action | Subscriber Action | Notes |
|---|---|---|
| Create a global Apex class, `PackageDevClass`, containing a global method `m1`. | | |
| Upload as Managed - Released version 1.0 of a package that contains `PackageDevClass`. | | |
| | Install version 1.0 of the package. | The `Version Number` for the package is 1.0. The `First Installed Version Number` is 1.0. |

| Package Developer Action | Subscriber Action | Notes |
|---|---|---|
| | Create an Apex class, `SubscriberClass`, that references `m1` in `PackageDevClass`. | |
| Deprecate `m1` and create a new method, `m2`. | | |
| Upload as Managed - Released version 2.0 of the package. | | |
| | Install version 2.0 of the package. | The `Version Number` for the package is 2.0. The `First Installed Version Number` is still 1.0. `SubscriberClass` still references version 1.0 of the package and continues to function, as before. |
| | Edit the version settings for `SubscriberClass` to reference version 2.0 of the package. Save the class. Note an error message indicating that `m1` can't be referenced in version 2.0 of the package. | |
| | Change `SubscriberClass` to reference `m2` instead of `m1`. Successfully save the class. | |

# Publish Upgrades to First-Generation Managed Packages

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

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

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

USER PERMISSIONS

To configure namespace settings:
- Customize Application

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages

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

4. Click **Upload** and upload it as usual.

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

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

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

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

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

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

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

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

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

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

# Plan the Release of First-Generation Managed Packages

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

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

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

| State | Description |
| --- | --- |
| Managed - Beta | The package or component was created in the current Salesforce org and is managed, but it isn't released because of one of these reasons:<br>• It hasn't been uploaded. |

| State | Description |
| --- | --- |
| | • It has been uploaded with `Managed - Beta` option selected. This option prevents it from being published, publicly available on AppExchange. The developer can still edit any component but the installer isn't able to depending on which components were packaged.<br>Don't install a Managed - Beta package over a Managed - Released package. If you do, the package is no longer upgradeable and your only option is to uninstall and reinstall it. |
| Managed - Released | The package or component was created in the current Salesforce org and is managed. It's also uploaded with the `Managed - Released` option selected, indicating that it can be published on AppExchange and is publicly available. After you've moved a package to this state, some properties of the components can't be editable.<br>This type of release is considered a major release. |
| Patch | If you must provide a minor upgrade to a managed package, consider creating a patch instead of a new major release. A patch enables a developer to change the functionality of existing components in a managed package. Subscribers experience no visible changes to the package.<br>This type of release is considered a patch release. |
| Managed - Installed | The package or component was installed from another Salesforce org but is managed. |
| Unmanaged (Legacy) | The package hasn't been converted into a managed package. |

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

# Remove Components from First-Generation Managed Packages

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

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

**Impact of Component Removal in Subscriber Orgs**

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

To enable component deletion in your packaging org, log a case with Salesforce Partner Support.

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

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

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

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

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

These restrictions apply when deleting managed components.

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

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

## Removing Public Apex Classes and Public Visualforce Components

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

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

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

1. Stage one: Remove references.

i. Edit the global component that you want to delete.
   - For Visualforce, edit your Visualforce page or global Visualforce component to remove all references to public Apex classes or public Visualforce components.
   - For Lightning components, edit the global Lightning component to remove all references to other Lightning components.

ii. Upload your new package version.

iii. Push the stage-one upgrade to your subscribers.

2. Stage two: Delete your obsolete pages or components.
   i. Delete your Visualforce page, global Visualforce component, or global Lightning component.
   ii. Optionally, delete other related components and classes.
   iii. Upload your new package version.
   iv. Push the stage-two upgrade to your subscribers.

# Delete Components from First-Generation Managed Packages

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

One of the following situations may occur:

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

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

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

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

You can retrieve these types of components.

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

  - Workflow alerts
  - Workflow field updates
  - Workflow outbound messages
  - Workflow tasks
  - Workflow flow triggers

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

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

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

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

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

# Modifying Custom Fields after a Package Is Released

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

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

EDITIONS

Available in: **Developer** Edition

# Manage Versions of First-Generation Managed Packages

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

USER PERMISSIONS

To upload packages:
- Upload AppExchange Packages

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

To manage your versions:

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

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

   - To make a deprecated version available for installation again, click **Undeprecate**.
   - To view the package installation URL, see **Installation URL**. Installation URLs contain the 04t package ID for the package version.

# View Unused Components in a Managed Package

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

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

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

EDITIONS

Available in: Salesforce Classic

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

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

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

# Push Package Upgrades to Subscribers

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

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

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

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

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

Push Upgrades

Push Upgrade Best Practices

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

Assign Access to New and Changed Features in First- and Second-Generation Managed Packages

Determine how to provide existing non-admin users access to new and changed features. By default, any new components included in the push upgrade package version are assigned only to admins.

Sample Post Install Script for a Push Upgrade for First- and Second-Generation Managed Packages

Automate the assignment of new components to existing users of a package.

Scheduling Push Upgrades

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

## Push Upgrades

Overview of Push Upgrade Steps

- Upgrade your managed package installed in a customer organization from version X to version Y
- Select one, many, or all customer organizations to upgrade and select a particular version to upgrade to
- Schedule the upgrade to start at a particular date and time
- View progress of upgrades, abort upgrades in progress, or view the result of a push upgrade
- In conjunction with push, you can use a post-install Apex script to automate post-upgrade configurations that your customers have previously performed manually

Warning: When you push an upgrade, you're making changes to a subscriber's org without explicit consent. Therefore, it's important to plan ahead and exercise caution. You can also exclude specific subscriber orgs from a push upgrade by entering the org IDs, separated by a comma, in the Push Upgrade Exclusion List.

Pushing a major upgrade entails a higher degree of risk as it can impact existing functionality in a subscriber's organization. This is because new components in the upgraded package might not be available to existing users of the package, or could overwrite users' customizations.

As the app developer, it's your responsibility to protect users from any adverse impact due to upgrading. We strongly recommend you consider all potential consequences of the upgrade and take appropriate steps to prevent any problems.

When pushing a major upgrade, we recommend that you divide changes in your package into two categories:

1. Enhancements to existing features that users already have access to—Use a post install Apex script to automatically assign the relevant components to existing users. This ensures all current users of the package can continue using it without explicit action by administrators.
2. New features you're introducing—Don't use a post install Apex script to auto-assign components. This ensures your subscribers have the opportunity to decide if and when to use the new features.

Here are some additional guidelines to keep in mind when planning a push upgrade.

- Avoid changes to validation rules, formula fields, and errors thrown from Apex triggers, as they may negatively impact subscribers' integrations.
- Don't make visible changes to a package in a patch. This is because other than a change in the package version number, subscribers aren't notified of push upgrades.
- Test your upgraded package in multiple environments, replicating all relevant features of your customers' organizations, including editions, customizations, other installed packages, and permission sets.
- Schedule push upgrades at your customers' off-peak hours and outside of Salesforce's major release windows, to minimize potential subscriber impact.
- Notify your subscribers in advance about the timing of the upgrade, its potential consequences, and any steps they need to take.

## Push Upgrade Best Practices

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

### Plan, Test, and Communicate

- Share an upgrade timeline plan with your customers so they know when you'll upgrade, and how often.
- Plan when you want to push upgrades to your customers' orgs. Keep in mind that most customers don't want changes around their month-end, quarter-end, and year-end or audit cycles. Do your customers have other critical time periods when they don't want any changes to their org? For example, there might be certain times when they don't have staff available to verify changes or perform any required post-installation steps.
- Schedule push upgrades during your customers' off-peak hours, such as late evening and night. Have you considered time zone issues? Do you have customers outside the United States who have different off-peak hours? You can schedule push upgrades to any number of customer organizations at a time. Consider grouping organizations by time zone, if business hours vary widely across your customer base.
- Don't schedule push upgrades close to Salesforce-planned maintenance windows. In most cases, it might be better to wait 3-4 weeks after a major Salesforce release before you push major upgrades.
- Test, test, and test! Since you're pushing changes to the organization instead of the customer pulling in changes, there's a higher bar to ensure the new version of your app works well in all customer configurations.

### Stagger Your Push Upgrades

- Don't push changes to all customers at once. It's important to ensure that you have sufficient resources to handle support cases if there are issues. Also, it's important that you discover possible issues before your entire customer base is affected.
- Push to your own test organizations first to confirm that the push happens seamlessly. Log in to your test organization after the push upgrade and test to see that everything works as expected.

- When applicable, push to the sandbox organizations of your customers first before pushing to their production organizations. Give them a week or more to test, validate, and fix in the sandbox environment before you push to their production organizations.
- Push upgrades to small batches of customer production organizations initially. For example, if you have 1,000 customers, push upgrades to 50 or 100 customers at a time, at least the first few times. After you have confidence in the results, you can upgrade customers in larger batches.

### Focus on Customer Trust

- You're responsible for ensuring that your customers' organizations aren't adversely affected by your upgrade. Avoid making changes to the package, such as changes to validation rules or formula fields, that might break external integrations made by the customer. If for some reason you do, test and communicate well in advance. Please keep in mind that you can impact customer data, not just metadata, by pushing an upgrade that has bugs.
- Write an Apex test on install to do basic sanity testing to confirm that the upgraded app works as expected.
- If you're enhancing an existing feature, use a post-install script to automatically assign new components to existing users using permission sets.
- If you're adding a new feature, don't auto-assign the feature to existing users. Communicate and work with the admins of the customer org so they can determine who should have access to the new feature, and the timing of the rollout.

## Assign Access to New and Changed Features in First- and Second-Generation Managed Packages

Determine how to provide existing non-admin users access to new and changed features. By default, any new components included in the push upgrade package version are assigned only to admins.

| If the push upgrade includes: | We recommend you: |
|---|---|
| New features | Notify admins about the changes the upgrade introduces, and ask them to assign permissions to all users of the package.<br>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.<br>This approach ensures that current users of the package can continue using features without interruption.<br>Note: Post-install scripts aren't available to unlocked packages. |

## Sample Post Install Script for a Push Upgrade for First- and Second-Generation Managed Packages

Automate the assignment of new components to existing users of a package.

Note: Post-install scripts can be used with first and second-generation managed packages only.

For more information on writing a post-install Apex script, see Run Apex on Package Install/Upgrade on page 352.

In this sample script, the package upgrade contains new Visualforce pages and a new permission set that grants access to those pages. The script performs the following actions.

- Gets the Id of the Visualforce pages in the old version of the package
- Gets the permission sets that have access to those pages
- Gets the list of profiles associated with these permission sets
- Gets the list of users who have those profiles assigned
- Assigns the permission set in the new package to those users

```
global class PostInstallClass implements InstallHandler {
    global void onInstall(InstallContext context) {

        //Get the Id of the Visualforce pages
        List<ApexPage> pagesList = [SELECT Id FROM ApexPage WHERE NamespacePrefix =
            'TestPackage' AND Name = 'vfpage1'];

        //Get the permission sets that have access to those pages
        List<SetupEntityAccess> setupEntityAccessList = [SELECT Id,
            ParentId, SetupEntityId, SetupEntityType FROM SetupEntityAccess
            WHERE SetupEntityId IN :pagesList];
        Set<ID> PermissionSetList = new Set<ID> ();

        for (SetupEntityAccess sea : setupEntityAccessList) {
            PermissionSetList.add(sea.ParentId);
        }
        List<PermissionSet> PermissionSetWithProfileIdList =
            [SELECT id, Name, IsOwnedByProfile, Profile.Name,
            ProfileId FROM PermissionSet WHERE IsOwnedByProfile = true
            AND Id IN :PermissionSetList];

        //Get the list of profiles associated with those permission sets
        Set<ID> ProfileList = new Set<ID> ();
        for (PermissionSet per : PermissionSetWithProfileIdList) {
            ProfileList.add(per.ProfileId);
        }

        //Get the list of users who have those profiles assigned
        List<User> UserList = [SELECT id FROM User where ProfileId IN :ProfileList];


        //Assign the permission set in the new package to those users
        List<PermissionSet> PermissionSetToAssignList = [SELECT id, Name
            FROM PermissionSet WHERE Name='TestPermSet' AND
            NamespacePrefix = 'TestPackage'];
        PermissionSet PermissionSetToAssign = PermissionSetToAssignList[0];
        List<PermissionSetAssignment> PermissionSetAssignmentList = new
List<PermissionSetAssignment>();
        for (User us : UserList) {
            PermissionSetAssignment psa = new PermissionSetAssignment();
            psa.PermissionSetId = PermissionSetToAssign.id;
            psa.AssigneeId = us.id;
            PermissionSetAssignmentList.add(psa);
        }
        insert PermissionSetAssignmentList;
```

```
    }
}
```

```
// Test for the post install class
@isTest
private class PostInstallClassTest {
    @isTest
    public static void test() {
      PostInstallClass myClass = new PostInstallClass();
      Test.testInstall(myClass, null);
    }
}
```

## Scheduling Push Upgrades

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

USER PERMISSIONS

To push an upgrade:
- Upload AppExchange Packages

1. Push the upgrade to your own orgs so you can run tests and fix any bugs before upgrading subscribers.
2. When you're ready and have coordinated with your customers on their change management process, push to a small number of customer organizations. Try sandbox organizations first, if possible.
3. After you're comfortable with the initial results, push to your wider customer base, based on your agreements with each customer.
4. Deprecate the previous version of your package in your main development organization. Replace the version on AppExchange if necessary, and update your Trialforce setup.
5. If your upgrade was a patch, after you've successfully distributed the upgrade to subscriber organizations, reintegrate those changes into your main development organization. For more information about combining patches in the main development organization, see Working with Patch Versions on page 361.

### Schedule a Push Upgrade Using the UI

Note: Only first-generation managed packages can schedule a push upgrade using the UI. To schedule a push upgrade for unlocked and second-generation managed packages, use the PackagePushRequest in the *Salesforce Object Reference*.

1. Log in to your main development org (not the patch org you used to upload the new version).
2. From Setup, enter *Packages* in the Quick Find box, then select **Packages**.
3. Click the name of the managed package whose upgrade you want to push.
4. On the package detail page, click the **Versions** tab, and then click **Push Upgrades**.
5. Click **Schedule Push Upgrades**.
6. Select a package version to push from the **Patch Version** dropdown list.

   Note: Beta versions aren't eligible for push.

7. For the scheduled start date, enter when you want the push upgrade to begin.

   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.

8. In the Select Target Organizations section, select the orgs to receive your push upgrade. If an org already received a push upgrade for the selected package version, it doesn't appear in this list. You can select orgs by:
   - Entering a term that filters based on an org's name or ID. Names can match by partial string, but IDs must match exactly.
   - Choosing between production and sandbox orgs from the **Organizations** dropdown list.
   - Choosing orgs that have already installed a particular version.
   - Clicking individual orgs or the **Select All** and **Deselect All** checkboxes.

   This section lists the following information about the org (in alphabetical order):

| Field | Description |
|---|---|
| `Current Version` | The current package version an organization has installed. |
| `Organization ID` | The ID of the org. |
| `Organization Name` | The name of the org. To view the upgrade history for the org, click the org name. |
| `Primary Contact` | The name of the user who installed the package. |

9. Click **Schedule**. While a push upgrade is in progress, you can click Abort to stop it.

## Schedule a Push Upgrade Using the Enterprise API

1. Authenticate to your main development org (not the patch org you used to upload the new version).
2. Determine the package version you want to upgrade subscribers to by querying the MetadataPackageVersion object.
3. Gather the list of subscriber orgs that are eligible to be upgraded by querying the PackageSubscriber object.

   Note: If you're retrieving more than 2,000 subscribers, use SOAP API `queryMore()` call.

4. Create a PackagePushRequest object. PackagePushRequest objects take a PackageVersionId and, optionally, a ScheduledStartTime parameter to specify when the push begins. If you omit the ScheduledStartTime, the push begins when you set the PackagePushRequest's status to `Pending`.
5. Create a PackagePushJob for each eligible subscriber and associate it with the PackagePushRequest you created in the previous step.
6. Schedule the push upgrade by changing the status of the PackagePushRequest to `Pending`.
7. Check the status of the PackagePushRequest and PackagePushJob objects by querying the `Status` fields. If the status is either Created or Pending, you can abort the push upgrade by changing the status of the PackagePushRequest to Canceled. You can't abort a push upgrade that has a status of Canceled, Succeeded, Failed, or In Progress.

   Note: If you're pushing the upgrade to more than 2,000 subscribers, use the Bulk_API to process the job in batches.

For sample code and more details, see *SOAP API Developer Guide*.

# Manage Licenses for Managed Packages

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.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

| I need to... | Permissions | For details, see... |
|---|---|---|
| Configure the LMA | System Admin profile | Get Started with the License Management App on page 378 |
| Bill subscribers or monitor license expiration | Object Permissions: Read | Lead and License Records in the LMA |
| Convert trial subscriptions into paying customers | Object Permissions: Edit | Modify a License Record |
| Customize the LMO | Object Permissions: Edit | Extend the LMA |
| Provision licenses to a subscriber | Object Permissions: Edit | Modify a License Record |
| Support subscribers with technical issues | Various permissions (see Assign Permissions to the Subscriber Support Console on page 381 for details) | Troubleshoot Subscriber Issues |

Note: The LMA is available only in English.

The LMA is available to eligible Salesforce partners. For more information on the Partner Program, including eligibility requirements, visit https://partners.salesforce.com.

### Get Started with the License Management App

To start managing leads and licenses with the License Management App (LMA), complete these installation and configuration steps.

### Lead and License Records in the License Management App

Each time a customer installs your managed package, the License Management App (LMA) creates lead and license records.

### Modify a License Record

You can change a customer's access to your offering by modifying a license record using the License Management App (LMA). For example, you can increase or decrease the number of seats included with a license or change the expiration date.

### Refresh Licenses for a Managed Package

To sync all license records for a package across all subscriber installations, you refresh the license. Refreshing the license can also resolve discrepancies between the number of licenses in a subscriber's org and the number displayed in the License Management App (LMA). Refreshing is required when you move the LMA to a different org.

### Extending the License Management App

The License Management App (LMA) is a managed package that you can customize and extend. In addition to using the LMA to manage leads and licenses, many partners also integrate it into their existing business processes.

Move the License Management App to Another Salesforce Org

You can move an LMA to a different org, but your package and license records don't automatically move with it. You must manually relink your packages and refresh the licenses.

Troubleshoot the License Management App

If you're experiencing issues with the License Management App, review these troubleshooting tips.

Best Practices for the License Management App

Follow these best practices when you use the License Management App (LMA).

Troubleshoot Subscriber Issues

Use the Subscriber Support Console to access information about your subscribers. Subscribers can also grant you login access to troubleshoot issues directly within your app. After you're granted access, you can log in to the subscriber's org and view their configuration and data to troubleshoot and resolve issues.

# Get Started with the License Management App

To start managing leads and licenses with the License Management App (LMA), complete these installation and configuration steps.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

Install the License Management App

The License Management App (LMA) is a managed package that is installed in all partner business orgs. The org that the LMA is installed in is called the License Management Org (LMO).

Associate a Package with the License Management App

To receive lead and license records for your package, you connect your License Management Org (LMO), your package, and the Salesforce Partner Console. Your LMO is the Salesforce org where the License Management App (LMA) is installed.

Configure Permissions for the License Management App

Determine who needs access to the License Management App (LMA), and set object permissions. Consider using a permission set to assign user permissions.

## Install the License Management App

The License Management App (LMA) is a managed package that is installed in all partner business orgs. The org that the LMA is installed in is called the License Management Org (LMO).

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.

USER PERMISSIONS

To install packages:
- Download AppExchange Packages

Commercial use of the LMA is prohibited in Developer and Partner Developer Edition orgs. Installing the LMA in a Developer Edition org is allowed only if you're building integrations with the LMA and need an environment only for development and testing purposes. You can install the LMA in Enterprise, Unlimited, or Performance Edition production orgs.

It's not possible to have Slack or the Declarative Lookup Rollup Summary (DLRS) package installed in the same org as the LMA. If the org in which you plan to install the LMA has either Slack or the DLRS package installed, uninstall them before you install the LMA. Alternatively, install the LMA in a different org.

Note: To confirm whether your PBO already has the LMA installed, skip to step 4.

1. To install the LMA in an org other than your PBO, log a case in the Partner Community. After we review the case, you receive an email with an installation URL.
2. Log in to the org where you want to install the LMA, and then go to the installation URL included in the email.
3. Choose which users can access the LMA, and then click **Install**.
4. To confirm that the LMA is installed, open the App Launcher. If the installation was successful, the License Management App appears in the list of available apps.

## Associate a Package with the License Management App

To receive lead and license records for your package, you connect your License Management Org (LMO), your package, and the Salesforce Partner Console. Your LMO is the Salesforce org where the License Management App (LMA) is installed.

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:<br>Assign READ permissions<br>To modify license records:<br>Assign READ and EDIT permissions |
| Package | To view package records:<br>Assign READ permissions<br>To modify package records:<br>Assign READ and EDIT permissions |
| Package Version | To view package version records:<br>Assign READ permissions<br>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 |

| To enable... | Add the Licenses related list to the... |
| --- | --- |
| 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 |

Note: Lead assignment rules aren't triggered for leads created by the 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. You must edit license records on the Modified License page, don't attempt to edit license records directly.
3. Update the field values as needed.

| Field | Description |
|---|---|
| `Expiration` | Enter the last day that the customer can access your package, or select **Does not expire**. |
| `Seats` | Enter the number of licensed seats, or select **Site License** to make your package available to all users in the customer's org. You can allocate up to 99,000,000 seats. |
| `Status` | Select a value from the dropdown.<br>• **Trial**—Lets the customer try your offering for up to 90 days. After the trial license converts to an active license, it can't return to a trial state.<br>• **Active**—Lets the customer use your package according to the license agreement.<br>• **Suspended**—Prohibits the customer from accessing your offering.<br>Note: When your offering is uninstalled, its status is set to Uninstalled, and the license can't be edited. |

4. Click **Save**.

# Refresh Licenses for a Managed Package

To sync all license records for a package across all subscriber installations, you refresh the license. Refreshing the license can also resolve discrepancies between the number of licenses in a subscriber's org and the number displayed in the License Management App (LMA). Refreshing is required when you move the LMA to a different org.

Note: For each package, you can refresh licenses only one time per week.

1. From the LMA, select the **Packages** tab.
2. Open the package record.
3. Click **Refresh Licenses**. In Lightning Experience, Refresh Licenses is located in the dropdown menu.

# Extending the License Management App

The License Management App (LMA) is a managed package that you can customize and extend. In addition to using the LMA to manage leads and licenses, many partners also integrate it into their existing business processes.

The LMA includes these custom objects:

- License
- Package on page 383
- Package Version on page 383

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. |

| Package Custom Object Fields | Description |
|---|---|
| `Last License Refresh` | The date when the License Refresh tool was last run. |
| `Latest Version` | The most recent package version you've released. |
| `Lead Manager` | The owner of the lead records that the LMA creates when a customer installs your package. |
| `Next Available Refresh` | The date when the License Refresh tool can be run again. |
| `Owner` | The LMA owns all package records. |
| `Package ID` | The 18-character ID that identifies the package. This ID starts with 033. |
| `Package Name` | The name you specified when you created the package. |

| Package Version Object Fields | Description |
|---|---|
| `Package` | The package name and links to the package record's detail page. |
| `Package Version Name` | The name you specified when you created the package version. |
| `Release Date` | The date you created this package version. |
| `Version Number` | The version number in major.minor.patch format. For example, 3.1.0. |
| `Version ID` | The 18-character ID of this package version. |

## License Object Fields

Use the License custom object to set limits on how many users in the subscriber's org can use your app and for how long.

The License Management App (LMA) creates a license record every time your package is installed in an org. For example, if a subscriber installs two of your 1GP packages and three of your 2GP packages, you have five license records for that subscriber in your LMA. If you deliver a 2GP app that is composed of multiple packages, a unique license record is created for each package in the app. You can allocate up to 99,000,000 seats per subscriber license.

To view details about a license record, select the **Licenses** tab in the LMA, and then select and open the license record.

License records are automatically created and contain critical information for tracking licenses. Do not directly edit the license record. Instead, use the Modify License tool to change the expiration date, license status, and the number of licensed seats.

| License Custom Object Fields | Description |
|---|---|
| `Account` | A lookup field to the account record for a converted lead. |
| `Contact` | A lookup field to the contact record for a converted lead. |
| `Created By` | License records are always created by the LMA. |
| `Expiration Date` | Displays the expiration date or `Does not expire` (default). |
| `Install Date` | The date the subscriber installed this package version. |
| `Instance` | The Salesforce instance where the subscriber's org resides. |

| License Custom Object Fields | Description |
|---|---|
| `Lead` | The lead record that the LMA created when the package was installed. A lead represents the user who owns the license.<br>If you convert the lead into an opportunity, the lead name is retained but the lead record no longer exists. |
| `License Name` | An auto-generated number that represents an instance of a license. License names are in the format of L-00001, and each new license is incremented by one. |
| `Licensed Seats` | Displays the number of licenses or `Site License` (default). When a package is installed in a sandbox org, `Site License` is the default. If a free trial package is installed in a sandbox org, the `Site License` is applied. |
| `License Status` | The type of license: Active, Suspended, Trial, or Uninstalled. |
| `License Type` | This is a legacy field and can be ignored. |
| `Org Edition` | The edition of the subscriber's org. |
| `Org Expiration Date` | Applies only if the subscriber installs your package in a trial org. Indicates the date when the trial org expires. It isn't related to the package license expiration. |
| `Org Status` | The status of the subscriber's org: Active, Free, or Trial. |
| `Owner` | The LMA owns all license records. Don't edit this field. |
| `Package Version` | A lookup field that links to the package version associated with this license. |
| `Package Version Number` | The version number in major.minor.patch format. For example, 3.1.0. |
| `Sandbox` | Indicates whether the license is for a package installed in a sandbox org. |
| `Subscriber Org ID` | The 15-character ID representing the subscriber's org. |
| `Used Licenses` | Displays the number of users who have a license to the package.<br>This field is blank if:<br>• A customer uninstalled the package.<br>• `Licensed Seats` is set to Site License. |

## Adding Custom Automation to License Management App Objects

Here are some examples of how you can use the License Management App (LMA) to grow your business and retain customers.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

### Alert Sales Reps Before a License Expires

If you're managing licenses for several packages, it can be difficult to track the various expirations. If a license expires accidentally, you could even lose a customer. To help your customers with renewals, set up 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.

1. To remove the association between the LMA and the org where it's currently installed, log a case with Salesforce Partner Support.
2. Install the LMA in the new org on page 378.
3. Associate your packages with the new org on page 379.
4. Refresh licenses for your packages on page 383.

USER PERMISSIONS

To install packages:
- Download AppExchange Packages

To manage licenses in the Partner Community:
- Manage Listings

## 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, it's possible that 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, it's possible that the subscriber org is locked, deleted, or disabled.

If you attempt to contact the subscriber and they aren't responsive, consider deleting the license record.

# Best Practices for the License Management App

Follow these best practices when you use the License Management App (LMA).

- To take advantage of entitlements that are unique to AppExchange partners, use your partner business org as your License Management Org.
- Create a list view filter for leads created by installed packages. The filter helps your team separate subscriber-based leads from leads coming from other sources.
- Use the API to find licensed users. The `isCurrentUserLicensed` method determines if a user has a license to a managed package. For more information, see the *Apex Reference Guide*.
- Treat the LMA custom objects as read-only. Use the Modify License page to edit licenses. Don't attempt to directly or programmatically edit license records.
- The LMA automatically creates package, package version, and license records. Customizations, such as adding required custom fields or creating workflow rules, triggers, or validation rules that require custom fields, can prevent the LMA from working properly.

# Troubleshoot Subscriber Issues

Use the Subscriber Support Console to access information about your subscribers. Subscribers can also grant you login access to troubleshoot issues directly within your app. After you're granted access, you can log in to the subscriber's org and view their configuration and data to troubleshoot and resolve issues.

To access the Subscriber Overview page, click the organization's name from the **Subscribers** tab in the LMA.

Note: This feature is available to eligible Salesforce partners. For more information on the Partner Program, including eligibility requirements, see www.salesforce.com/partners.

Request Login Access from Subscribers
To log in to a subscriber org, first request login access from the subscriber.

Log In to Subscriber Orgs
After your subscriber has granted you login access, you can log in to the subscriber org to troubleshoot the issue.

Debug Subscriber Orgs
After logging in to a subscriber's org, you can view logs, obfuscated code in your package, and initiate ISV Customer Debugger sessions.

## Request Login Access from Subscribers

To log in to a subscriber org, first request login access from the subscriber.

Ask the subscriber to enable either **Grant Account Login Access** or **Grant Login Access**. If they don't see your company listed, one of the following applies.

- A system admin disabled the ability for non-admins to grant access.
- The user doesn't have a license for the package.
- The package is licensed to the entire org. In this scenario, only an admin with the Manage Users permission can grant access.
- The org setting **Administrators Can Log in as Any User** is enabled.

Note: When the org setting **Administrators Can Log in as Any User** is disabled, login access is granted for a limited amount of time, and the subscriber can revoke access at any time.

Any changes you make while logged in as a subscriber are logged in the subscriber org's audit trail.

# Log In to Subscriber Orgs

After your subscriber has granted you login access, you can log in to the subscriber org to troubleshoot the issue.

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

USER PERMISSIONS

To log in to subscriber orgs:
- Log in to Subscriber Org

Note: You can only log in to orgs with a Salesforce Platform or full Salesforce license. You can't log in to subscriber orgs on Government Cloud instances. 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 trial that indicates when and why a subscriber org login has occurred. You can create an audit trail by logging a case in your LMO before each subscriber org login.
- When you access a subscriber org, you're logged out of your LMO. 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 381.

## Debug Subscriber Orgs

After logging in to a subscriber's org, you can view logs, obfuscated code in your package, and initiate ISV Customer Debugger sessions.

### Troubleshoot with Debug Logs

You can debug your code by generating Apex debug logs that contain the output from your managed package. Using this log information, you can troubleshoot issues that are specific to that subscriber.

1. If the user has access, set up a debug log: From Setup, in the Quick Find box, enter `Debug Logs`, and then select **Debug Logs**.
2. Launch the Developer Console.
3. Perform the operation, and view the debug log with your output.

Subscribers are unable to see the logs you set up or generate because they contain your unobfuscated Apex code.

You can also view and edit data contained in protected custom settings from your managed packages when logged in as a user.

### Troubleshoot with the ISV Debugger

Each License Management Org can use one free ISV Customer Debugger session at a time. The ISV Customer Debugger is part of the Salesforce Extensions for Visual Studio Code. You can use the ISV Customer Debugger only in sandbox orgs, so you can initiate debugging sessions only from a customer's sandbox.

For details, see the ISV Customer Debugger documentation.

# Manage Features in First-Generation Managed Packages

Control how you release features to customers with the Feature Management App (FMA). The FMA extends the functionality of the License Management App (LMA). Use the FMA to manage features as easily as you manage licenses with the LMA.

Here at Salesforce, we sometimes run pilot programs, like the one we ran when we introduced Feature Management. Sometimes we dark-launch features to see how they work in production before sharing them with you. Sometimes we make features available to select orgs for limited-time trials. And sometimes we want to track activation metrics for those features.

With feature parameters, we're extending this functionality to you. Install the FMA in your License Management Org (LMO). The FMA extends the License Management App, and like the LMA, it's a managed package.

Feature Parameter Metadata Types and Custom Objects

Feature parameters are represented as Metadata API types in your packaging org, as records of custom objects in your License Management Org, and as hidden records in your subscriber's org.

Set Up Feature Parameters

Set up the Feature Management App in your License Management Org, define feature parameters, and add them to your package.

Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features

Feature parameters with a data flow direction value of `LMO to Subscriber` are writable at your end and read-only in your subscriber's org. These feature parameters serve as permissions or limits. Use LMO-to-subscriber feature parameters to enable or disable new features or to control how many of a given resource your subscriber can use. Or, enable features for a limited trial period. Assign values to LMO-to-subscriber feature parameters by updating junction object records in your LMO, and then check those values in your code.

Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters

Use subscriber-to-LMO feature parameters to track feature activation in your subscriber's org. Parameter values are assigned on the subscriber's end and then sent to your LMO. To collect the values, update the feature parameters in your subscriber's org using Apex code. Check with your legal team before obtaining activation metrics from your customers. Use activation metrics to collect only aggregated data regarding feature activation.

Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs

Occasionally, you want to include custom permissions or custom objects in a package but not show them to your subscribers. For example, if you're piloting a feature for a few select orgs, and want to hide custom permissions and custom objects related to the pilot feature.

Best Practices for Feature Management

Here are some best practices when working with feature parameters.

Considerations for Feature Management

Keep these considerations in mind when working with feature parameters.

# Feature Parameter Metadata Types and Custom Objects

Feature parameters are represented as Metadata API types in your packaging org, as records of custom objects in your License Management Org, and as hidden records in your subscriber's org.

## Feature Parameter Fields

Feature parameters are represented as Metadata API types and store boolean, integer, or date values.

The first time a subscriber installs your package, a `FeatureParameter__c` record is created in your License Management Org (LMO) for each feature parameter. The feature parameter records include these fields:

- `FullName__c`
- `DataType__c` (`Boolean`, `Integer`, or `Date`)
- `DataFlowDirection__c`
- `Package__c`
- `IntroducedInPackageVersion__c`
- `Namespace_Prefix__c`

## Lifecycle of a Feature Parameter

**Set Up the Feature Parameter**

Start by defining your feature parameter in the packaging org using the Feature Parameters tab on the Package detail page.

Depending on how you're using the feature parameter, you'll also write code that enables you to check access rights or collect usage information after the parameter is set up.

**Subscriber Installs Your Managed Package**

When a subscriber installs or upgrades your package in their org, a `FeatureParameter__c` record for each feature parameter is created in the LMO. If these records were created during a previous installation or upgrade, this step is skipped.

During package installation, junction object records are created in both the subscriber org and your LMO. A junction object is a custom object with two master-detail relationships. In this case, the relationships are between `FeatureParameter__c` and `License__c` in the LMO. These records store the value of their associated feature parameter for that subscriber org.

**Utilize Your Feature Parameters**

Use the junction objects to override the feature parameters' default values or to collect data. Depending on the value of each feature parameter's `DataFlowDirection__c` field, data flows to the subscriber org (from the LMO) or to the LMO (from the subscriber org). That data is stored in the junction object records.

# Set Up Feature Parameters

Set up the Feature Management App in your License Management Org, define feature parameters, and add them to your package.

Install and Set Up the Feature Management App in Your License Management Org

Install the FMA in your LMO. Then add the Feature Parameters tab to your default view, and adjust your page layout for licenses to display related lists for your feature parameters.

Create Feature Parameters in Your Packaging Org

Create a feature parameter in your packaging org, and set its type, default value, and data flow direction.

Add Feature Parameters to Your Managed Package

After you've created some feature parameters, you can add them to a managed package as components and reference them in your code. Feature parameters aren't available in unmanaged packages.

## Install and Set Up the Feature Management App in Your License Management Org

Install the FMA in your LMO. Then add the Feature Parameters tab to your default view, and adjust your page layout for licenses to display related lists for your feature parameters.

1. To request access to the FMA, log a support case in the Salesforce Partner Community. For product, specify **Partner Programs & Benefits**. For topic, specify **ISV Technology Request**. The FMA extends the License Management App, so be sure to install the LMA before requesting access to the FMA.
2. To install the FMA, follow the instructions in your welcome email.
3. Add the Feature Parameters tab to your default view. For details, see Customize My Tabs in Salesforce Help.
4. Update your page layout for licenses.
   a. Navigate to a license record's detail page.
   b. Click **Edit Layout**.
   c. In the Related Lists section of the License Page Layout Editor, add these lists.
      - Feature Parameter Booleans
      - Feature Parameter Dates
      - Feature Parameter Integers
   d. For each related list, add these columns.
      - Data Flow Direction
      - Feature Parameter Name
      - Full Name
      - Master Label
      - Value

## Create Feature Parameters in Your Packaging Org

Create a feature parameter in your packaging org, and set its type, default value, and data flow direction.

1. From Setup, enter `Package Manager` in the Quick Find box, then select **Package Manager**.
2. In the Packages section, in the Package Name column, select your managed package.
3. On the Feature Parameters tab, click **New Boolean**, **New Integer**, or **New Date**.

   If the Feature Parameters tab isn't visible, log a case with Salesforce Partner Support.
4. Give your feature parameter a developer name that meets the standard criteria for developer names. The name must be unique in your org. It can contain only alphanumeric characters and underscores, and must begin with a letter. It can't include spaces, end with an underscore, nor contain two consecutive underscores.
5. Give the feature parameter a label.
6. Set a default value for the feature parameter. If you're creating a Feature Parameter Boolean, you see only a checkbox for Default Value. If you want your default value to be `true`, select this checkbox. Integer values can't exceed nine digits.
7. Set a data flow direction. To use this feature parameter to control behavior in your subscriber's org, select **LMO to Subscriber**. To collect activation metrics from your subscriber, select **Subscriber to LMO**. Note: After the feature parameter is included in a promoted and released package version, the data flow direction can't be changed.
8. Click **Save**.

## Add Feature Parameters to Your Managed Package

After you've created some feature parameters, you can add them to a managed package as components and reference them in your code. Feature parameters aren't available in unmanaged packages.

A package can include up to 200 feature parameters.

Complete these steps in your packaging org.

1. From Setup, enter `Packages` in the Quick Find box, then select **Packages**.
2. In the Packages section, in the Package Name column, select your managed package.
3. On the Components tab, click **Add**.
4. From the Component Type dropdown, select **Feature Parameter Boolean**, **Feature Parameter Date**, or **Feature Parameter Integer**.
5. Select your feature parameter, and then click **Add to Package**.

# Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features

Feature parameters with a data flow direction value of `LMO to Subscriber` are writable at your end and read-only in your subscriber's org. These feature parameters serve as permissions or limits. Use LMO-to-subscriber feature parameters to enable or disable new features or to control how many of a given resource your subscriber can use. Or, enable features for a limited trial period. Assign values to LMO-to-subscriber feature parameters by updating junction object records in your LMO, and then check those values in your code.

### Assign Override Values in Your LMO

To override the default value of a feature parameter in a subscriber's org, update the appropriate junction object record in your LMO.

Check LMO-to-Subscriber Values in Your Code
You can reference feature parameters in your code, just like you'd reference any other custom object.

## Assign Override Values in Your LMO

To override the default value of a feature parameter in a subscriber's org, update the appropriate junction object record in your LMO.

1. Open the license record for a subscriber's installation of your package.
2. In the related list for Feature Parameter Booleans, Feature Parameter Integers, or Feature Parameter Dates, select the feature parameter whose value you want to update.
3. Click **Edit**.
4. Set a value.
5. Click **Save**.

## Check LMO-to-Subscriber Values in Your Code

You can reference feature parameters in your code, just like you'd reference any other custom object.

Use these Apex methods with LMO-to-subscriber feature parameters to check values in your subscriber's org.

- `System.FeatureManagement.checkPackageBooleanValue('YourBooleanFeatureParameter');`
- `System.FeatureManagement.checkPackageDateValue('YourDateFeatureParameter');`
- `System.FeatureManagement.checkPackageIntegerValue('YourIntegerFeatureParameter');`

SEE ALSO:

*Apex Reference Guide*: FeatureManagement Class

# Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters

Use subscriber-to-LMO feature parameters to track feature activation in your subscriber's org. Parameter values are assigned on the subscriber's end and then sent to your LMO. To collect the values, update the feature parameters in your subscriber's org using Apex code. Check with your legal team before obtaining activation metrics from your customers. Use activation metrics to collect only aggregated data regarding feature activation.

- `System.FeatureManagement.setPackageBooleanValue('YourBooleanFeatureParameter', booleanValue);`
- `System.FeatureManagement.setPackageDateValue('YourDateFeatureParameter', datetimeValue);`
- `System.FeatureManagement.setPackageIntegerValue('YourIntegerFeatureParameter', integerValue);`

Warning: The `Value__c` field on subscriber-to-LMO feature parameters is editable in your LMO. But don't change it. The changes don't propagate to your subscriber's org, so your values will be out of sync.

You can view the value of a subscriber-to-LMO feature parameter from the Subscriber Support Console.

SEE ALSO:

*Apex Reference Guide*: FeatureManagement Class

# Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs

Occasionally, you want to include custom permissions or custom objects in a package but not show them to your subscribers. For example, if you're piloting a feature for a few select orgs, and want to hide custom permissions and custom objects related to the pilot feature.

Note: Check with your company's legal team before releasing hidden functionality.

To hide custom objects when creating your package, set the value of their Visibility field to `Protected`. 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.

# AppExchange App Analytics for First-Generation Managed Packages

AppExchange App Analytics provides usage data about how subscribers interact with your first-generation (1GP) managed packages and packaged components. You can use these details to identify attrition risks, inform feature development decisions, and improve user experience.

**Note:** AppExchange App Analytics is subject to certain usage restrictions as described in the AppExchange Program Policies. Usage data from Government Cloud and Government Cloud Plus orgs isn't available in App Analytics.

App Analytics is available for managed 1GP packages that passed security review and are registered to a License Management App. Usage data is provided as package usage logs, monthly package usage summaries, or subscriber snapshots. All usage data is available as downloadable comma-separated value (.csv) files. To view the data in dashboard or visualization format, use CRM Analytics or a third-party analytics tool.

In a 24-hour period, you can download a maximum 20 GB of AppExchange App Analytics data.

### Enable App Analytics on Your First-Generation Managed Package

Activate AppExchange App Analytics on your first-generation (1GP) managed package to access AppExchange App Analytics package usage logs and subscriber snapshots. Package usage summaries are available by default.

SEE ALSO:

Get Started with AppExchange App Analytics

# Enable App Analytics on Your First-Generation Managed Package

Activate AppExchange App Analytics on your first-generation (1GP) managed package to access AppExchange App Analytics package usage logs and subscriber snapshots. Package usage summaries are available by default.

1. Log in to your packaging org.
2. Click the **gear icon** , and select **Setup**.
3. In the Quick Find box, enter `package`, and select **Package Manager**.
4. Find your package, and click **Edit**.
5. Check **Enable AppExchange App Analytics**.
6. Save your work.

For full documentation on available App Analytics data and query best practices, read Get Started with AppExchange App Analytics in the *Second-Generation Managed Packaging Developer Guide*.

EDITIONS

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

USER PERMISSIONS

To access packages and package versions:

- Read on Packages, Package Versions

To request and retrieve AppExchange App Analytics data:

- Create, Read, Edit, Delete, View All, and Modify All on the AppAnalyticsQueryRequest object

# Developing and Distributing Unmanaged Packages

Unmanaged packages can be used for distributing open-source projects to developers, or as a one-time drop of applications that require customization after installation.

After the components are installed from an unmanaged package, they can be edited in the org they're installed in. The developer who creates and uploads an unmanaged package has no control over the installed components, and can't change or upgrade them.

As a best practice, install an unmanaged package only if the org used to upload the package still exists. If that org is deleted, you may not be able to install the unmanaged package.

Don't use unmanaged packages for sandbox to production migration. Instead, use the Salesforce Extensions for Visual Studio Code or the Ant Migration Tool. If you're using Enterprise, Unlimited, or Performance Edition, see Change Sets.

Create and Upload an Unmanaged Package
Use the following procedure to upload an unmanaged package through the UI. You can also upload a package using the Tooling API. For sample code and more details, see the PackageUploadRequest object in the Tooling API Developer Guide.

Components Available in Unmanaged Packages
Not all components can be packaged for distribution.

Convert Unmanaged Packages to Managed

# Create and Upload an Unmanaged Package

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

1. Create the package:
   a. From Setup, enter `Package Manager` in the `Quick Find` box, then select **Package Manager**.
   b. Click **New**.
   c. Fill in the details of the package.
   d. Click **Save**.

2. On the Components tab, click **Add**.
3. From the Component Type dropdown list, choose a component.
4. Select the component you want to add.
5. Click **Add To Package**.
6. Repeat these steps until you've added all the components you want in your package.
7. Click **Upload**.

You will receive an email that includes an installation link when your package has been uploaded successfully. Wait a few moments before clicking the installation link or distributing it to others, as it might take a few minutes for it to become active.

## Considerations for Uninstalling Unmanaged Packages

If your unmanaged package has dependencies on metadata in another package, remove any dependencies before attempting to uninstall either package.

If you're working in a sandbox org, you must first remove the package dependencies in your production org.

1. Locate the unmanaged package in your production org and remove the dependencies to the package you plan to uninstall.
2. Create or refresh your sandbox org.
3. In your sandbox org, you can now uninstall the package that your unmanaged package previously depended on.

# Components Available in Unmanaged Packages

Not all components can be packaged for distribution.

**Packaged Explicitly or Implicitly**
Components can be added either explicitly or implicitly. Explicit components must be included directly in the package, while implicit components are automatically added. For example, if you create a custom field on a standard object, you must explicitly add the custom field to your package. However, if you create a custom object and add a custom field to it, the field is implicitly added to the package when you add the custom object.

- **Explicitly**: The component must be manually added to the package.
- **Implicitly**: The component is automatically added to the package when another dependent component, usually a custom object, is added.

**Automatic Renaming**
Salesforce can resolve naming conflicts automatically on install.

- **No**: If a naming conflict occurs the install is blocked.
- **Yes**: If a naming conflict occurs Salesforce can optionally change the name of the component being installed.

| Component | Packaged Explicitly or Implicitly | Automatic Renaming |
|---|---|---|
| **Apex Class** | Explicitly | No |
| **Apex Sharing Reason** | Implicitly<br>On an extension: Explicitly | No |
| **Apex Sharing Recalculation** | Implicitly | No |

| Component | Packaged Explicitly or Implicitly | Automatic Renaming |
|---|---|---|
| **Apex Trigger** | On a standard or extension object: Explicitly<br>On an object in the package: Implicitly | No |
| **Application** | Explicitly | No |
| **Custom Button or Link** | On a standard object: Explicitly<br>On a custom object: Implicitly | No |
| **Custom Field** | On a standard object: Explicitly<br>On a custom object: Implicitly | No |
| **Custom Label** | Implicitly | No |
| **Custom Object** | Explicitly | No |
| **Custom Permission** | Implicitly<br>With required custom permissions: Explicitly | No |
| **Custom Report Type** | Explicitly | No |
| **Custom Setting** | Explicitly | No |
| **Dashboard** | Explicitly<br>In a folder: Implicitly | Yes |
| **Document*** (10 MB limit) | Explicitly<br>In a folder: Implicitly | Yes |
| **Email Template** (Classic) | Explicitly<br>In a folder: Implicitly | Yes |
| **External Data Source** | Explicitly<br>Referenced by an external object: Implicitly<br>Assigned by a permission set: Implicitly | No |
| **Flow Definition** | Implicitly | No |
| **Folder** | Explicitly | Yes |
| **Home Page Component** | Explicitly | No |
| **Home Page Layout** | Explicitly | No |
| **Inbound Network Connection** | Explicitly | No |
| **Letterhead** | Explicitly | Yes |
| **Lightning Application** | Explicitly | No |
| **Lightning Component** | Explicitly | No |

| Component | Packaged Explicitly or Implicitly | Automatic Renaming |
|---|---|---|
| **Lightning Event** | Explicitly | No |
| **Lightning Interface** | Explicitly | No |
| **Lightning Page** | Explicitly | No |
| **List View** | On a standard object: Explicitly<br>On a custom object: Implicitly | Yes |
| **Named Credential** | Explicitly | No |
| **Outbound Network Connection** | Explicitly | No |
| **Page Layout** | On a standard object: Explicitly<br>On a custom object: Implicitly | No |
| **Record Type** | On a standard object: Explicitly<br>On a custom object: Implicitly | No |
| **Report** | Explicitly<br>In a folder: Implicitly | Yes |
| **Reporting Snapshot** | Explicitly | Yes |
| **S-Control*** (10 MB limit) | Explicitly | No |
| **Static Resource** | Explicitly | No |
| **Tab** | Explicitly | No |
| **Translation** | Explicitly | No |
| **Validation Rule** | On a standard object: Explicitly<br>On a custom object: Implicitly | No |
| **Visualforce Component** | Explicitly | No |
| **Visualforce Page** | Explicitly | No |
| **Workflow Email Alert** | Explicitly | No |
| **Workflow Field Update** | Explicitly | No |
| **Workflow Outbound Message** | Explicitly | No |
| **Workflow Rule** | Explicitly | No |
| **Workflow Task** | Explicitly | No |

*The combined size of S-Controls and documents must be less than 10 MB.

# Convert Unmanaged Packages to Managed

Before you convert an existing package to managed, alert any current installers that they must save their data:

1. Export all the data from the previous, unmanaged version of the package.
2. Uninstall the unmanaged package.
3. Install the new managed version of the package.
4. Import all the exported data into the new managed package.

> Note: Note to installers: if you have made customizations to an installation of an unmanaged package, make a list of these customizations before uninstalling since you may want to implement them again.

To convert an unmanaged package into a managed package:

1. Register a namespace.
2. From Setup, enter *Package Manager* in the Quick Find box, then select **Package Manager**.
3. Edit the package that you want to make managed, then select **Managed**.

EDITIONS

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

Available in: **Developer** Edition

Package uploads and installs are available in **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To configure namespace settings:
- Customize Application

To create packages:
- Create AppExchange Packages

To upload packages:
- Upload AppExchange Packages