



Salesforce DX Developer Guide

Version 58.0, Summer '23



© Copyright 2000–2023 Salesforce, Inc. All rights reserved. Salesforce is a registered trademark of Salesforce, Inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

Chapter 1: Release Notes	1
Chapter 2: How Salesforce Developer Experience Changes the Way You Work	2
Use a Sample Repo to Get Started	4
Create an Application	5
Migrate or Import Existing Source	6
Chapter 3: Enable Dev Hub Features in Your Org	7
Free Limited Access License	9
Enable Unlocked and Second-Generation Managed Packaging	9
Enable Source Tracking in Sandboxes	10
Enable Einstein Features	10
Enable Language Extension Packages (Beta)	10
Add Salesforce DX Users	11
Add a Developer User to Your Dev Hub Org	11
Add a System Administrator or Standard User to Your Dev Hub Org	12
Permission Set for Salesforce DX Users	12
Chapter 4: Project Setup	14
Sample Repository on GitHub	15
Create a Salesforce DX Project	15
Salesforce DX Project Structure and Source Format	16
How to Exclude Source When Syncing or Converting	22
Create a Salesforce DX Project from Existing Source	25
Retrieve Source from an Existing Managed Package	25
Retrieve Unpackaged Source Defined in a package.xml File	26
Convert the Metadata Source to Source Format	27
Salesforce DX Usernames and Orgs	28
Override or Add Definition File Options at the Command Line	30
Link a Namespace to a Dev Hub Org	31
Salesforce DX Project Configuration	32
Multiple Package Directories	35
Replace Strings in Code Before Deploying	38
Chapter 5: Authorization	43
Authorize an Org Using the Web Server Flow	44
Authorize an Org Using the JWT Bearer Flow	45
Authorize a Scratch Org	46
Create a Private Key and Self-Signed Digital Certificate	47
Create a Connected App in Your Org	48

Contents

Use the Default Connected App Securely	49
Use an Existing Access Token instead of Authorizing	50
Authorization Information for an Org	50
Log Out of an Org	51
Chapter 6: Metadata Coverage	53
Chapter 7: Scratch Orgs	54
Supported Scratch Org Editions and Allocations	56
Build Your Own Scratch Org Definition File	57
Scratch Org Features	61
Scratch Org Settings	113
Create a Scratch Org Based on an Org Shape	114
Enable Org Shape for Scratch Orgs	115
Org Shape Permissions	116
Create and Manage Org Shapes	117
Scratch Org Definition for Org Shape	118
Troubleshooting for Org Shape	119
Create Scratch Orgs	121
Select the Salesforce Release for a Scratch Org	123
Push Source to the Scratch Org	125
Pull Source from the Scratch Org to Your Project	127
Scratch Org Users	128
Create a Scratch Org User	129
User Definition File for Customizing a Scratch Org User	130
Generate or Change a Password for a Scratch Org User	131
Manage Scratch Orgs from Dev Hub	132
Scratch Org Error Codes	133
Chapter 8: Sandboxes	134
Authorize in to Your Production Org	135
Create a Sandbox Definition File	135
Create, Clone, or Delete a Sandbox	137
Chapter 9: Track Changes Between Your Project and Org	140
See Changes Identified by Source Tracking	142
Pull and Push Changes Identified by Source Tracking	143
Resolve Conflicts Between Your Local Project and Org	144
Get Change Information by Querying the SourceMember Object	144
Best Practices	145
Differences and Considerations	146
Performance Considerations of Source Tracking	146
Retrieve and Pull Changes to Profiles with Source Tracking	146
Chapter 10: Development	148

Contents

Develop Against Any Org	150
Assign a Permission Set	153
Ways to Add Data to Your Org	154
Example: Export and Import Data Between Orgs	155
Create Lightning Apps and Aura Components	156
Create Lightning Web Components	157
Create an Apex Class	157
Create an Apex Trigger	158
Run Apex Tests	159
Debug Apex	160
Generate and View Apex Debug Logs	161
Chapter 11: Build and Release Your App	162
Build and Release Your App with Metadata API	164
Develop and Test Changes Locally	166
Build and Test the Release Artifact	167
Test the Release Artifact in a Staging Environment	167
Release Your App to Production	168
Cancel a Metadata Deployment	168
Chapter 12: Create a First-Generation Managed Package using Salesforce DX	169
Build and Release Your App with Managed Packages	170
Packaging Checklist	170
Deploy the Package Metadata to the Packaging Org	171
Create a Beta Version of Your App	172
Install the Package in a Target Org	172
Create a Managed Package Version of Your App	173
View Information About a Package	174
View All Package Versions in the Org	174
Package IDs	175
Chapter 13: Second-Generation Managed Packages	176
What's a Second-Generation Managed Package?	178
Why Switch to Second-Generation Managed Packaging?	178
Comparison of First- and Second-Generation Managed Packages	180
Before You Create Second-Generation Managed Packages	181
Know Your Orgs for Second-Generation Managed Packages	181
Namespaces for Second-Generation Managed Packages	182
Create and Register Your Namespace for Second-Generation Managed Packages	183
Key Concepts in Second-Generation Managed Packaging	183
How Manageability Rules and Ancestry Impact Upgrades for Second-Generation Managed Packages	184
Which Package Dependencies Work with Second-Generation Managed Packages?	185
Workflow for Second-Generation Managed Packages	186

Contents

Components Available in Managed Packages	188
Account Relationship Share Rule	194
Action	195
Action Link Group Template	196
Action Plan Template	197
Activation Platform	197
AI Application	199
AI Application Config	200
AIUsecaseDefinition	201
Analytics	202
Apex Class	203
Apex Sharing Reason	204
Apex Sharing Recalculation	205
Apex Trigger	206
Application	206
Application Subtype Definition	207
Article Type	208
AssessmentQuestion	209
AssessmentQuestionSet	210
Batch Calc Job Definition	210
Batch Process Job Definition	211
Benefit Action	212
Branding Set	213
Briefcase Definition	214
Building Energy Intensity Record Type Configuration	215
Business Process Group	216
Business Process Type Definition	217
Call Center	218
Care Benefit Verify Settings	218
Care Limit Type	219
Care Request Configuration	220
Care System Field Mapping	221
Chatter Extension	222
Community Template Definition	223
Community Theme Definition	224
Compact Layout	225
Connected App	226
Contract Type	228
Conversation Vendor Info	229
CORS Allowlist	229
CSP Trusted Site	230
Custom Application	232
Custom Button or Link	232
Custom Console Components	233

Contents

Custom Field on Standard or Custom Object	234
Custom Field on Custom Metadata Type	236
Custom Help Menu	236
Custom Index	236
Custom Label	237
Custom Metadata Records	238
Custom Metadata Types	238
Custom Notification Type	239
Custom Object Translation	240
Custom Object	241
Custom Permission	242
Custom Setting	243
Custom Tab	244
Dashboard	245
Data Classification on Custom Fields	246
Data Connector Ingest API	247
Data Connector S3	248
Data Stream Definition	249
Data Source	250
Data Source Object	251
Decision Matrix Definition	252
Decision Matrix Definition Version	253
Decision Table	254
Decision Table Dataset Link	255
Discovery AI Model	256
Discovery Goal	256
Discovery Story	257
Document	258
Document Generation Setting	259
Eclair GeoData	259
Email Template (Classic)	260
Email Template (Lightning)	261
Embedded Service Config	262
Embedded Service Menu Settings	263
Entitlement Process	263
Entitlement Template	264
ESignature Config	265
ESignature Envelope Config	266
Experience Builder Template	267
Experience Builder Theme	267
Explainability Action Definition	267
Explainability Action Version	268
Explainability Message Template	269
Expression Set Definition	270

Contents

Expression Set Definition Version	271
Expression Set Object Alias	272
Embedded Service Deployment	273
External Actions	273
External Credential	275
External Data Connector	276
External Data Source	277
External Services	278
Field Set	279
Field Source Target Relationship	280
Flow	281
Flow Category	283
Flow Definition	284
Flow Test	285
Fuel Type	286
Fuel Type Sustainability Unit of Measure	287
Folder	288
Gateway Provider Payment Method Type	289
Global Picklist	289
Home Page Component	290
Home Page Layout	291
Identity Verification Proc Def	292
Inbound Network Connection	293
Letterhead	294
Lightning Application	295
Lightning Bolt	295
Lightning Component	296
Lightning Event	297
Lightning Interface	298
Lightning Message Channel	298
Lightning Page	299
List View	300
Live Chat Sensitive Data Rule	301
Loyalty Program Setup	301
Marketing App Extension	302
Market Segment Definition	304
Milestone Type	304
MktCalculatedInsightsObjectDef	305
MktDataTranObject	306
ML Prediction Definition	307
ML Recommendation Definition	308
Named Credential	309
Recommendation Strategy	311
Sustainability UOM	311

Contents

Sustainability UOM Conversion	312
Object Source Target Map	313
OcrSampleDocument	314
OcrTemplate	315
Outbound Network Connection	316
Page Layout	317
Path Assistant	318
Payment Gateway Provider	319
Permission Set	320
Permission Set Groups	321
Platform Cache	322
Platform Event Channel	322
Platform Event Channel Member	323
Platform Event Subscriber Configuration	324
Process	324
Prompts (In-App Guidance)	325
Quick Action	325
Record Action Deployment	326
Record Type	327
RedirectWhitelistUrl	328
Referenced Dashboard	329
Remote Site Setting	330
Report	331
Report Type	332
Reporting Snapshot	333
Salesforce IoT	334
Slack App	334
Service Catalog Category	335
Service Catalog Item Definition	336
Service Catalog Fulfillment Flow	337
Stationary Asset Environmental Source Record Type Configuration	338
Static Resource	339
Streaming App Data Connector	340
Timeline Object Definition	341
Timesheet Template	342
Translation	343
UI Object Relation Config	345
Validation Rule	346
Vehicle Asset Emissions Source Record Type Configuration	347
View Definition	348
Virtual Visit Config	349
Visualforce Component	350
Visualforce Page	351
Wave Application	352

Contents

Wave Component	353
Wave Dataflow	354
Wave Dashboard	355
Wave Dataset	356
Wave Lens	357
Wave Recipe	358
Wave Template Bundle	359
Wave Xmd	360
Web Store Template	361
Workflow Email Alert	361
Workflow Field Update	362
Workflow Outbound Message	363
Workflow Rule	364
Workflow Task	365
Develop Second-Generation Managed Packages	366
Create and Update a Second-Generation Managed Package	366
View Package Details for a Second-Generation Managed Package	367
Create and Update Versions of a Second-Generation Managed Package	367
View Details about a Second-Generation Managed Package Version	368
Project Configuration File for a Second-Generation Managed Package	370
Get Ready to Promote and Release a Second-Generation Managed Package Version	374
Specify a Package Ancestor in the Project File for a Second-Generation Managed Package	374
Install and Uninstall Second-Generation Managed Packages	376
Use the CLI to Install a Second-Generation Managed Package	377
Use a URL to Install a Second-Generation Managed Package	378
Upgrade a Second-Generation Managed Package Version	378
Uninstall a Second-Generation Managed Package	378
Prepare to Distribute Your Second-Generation Managed Package	379
Code Coverage for Second-Generation Managed Packages	380
Package Installation Key for Second-Generation Managed Packages	380
Release a Second-Generation Managed Package	381
Share Release Notes and Post-Install Instructions for Second-Generation Managed Packages	382
Publishing Your App on AppExchange	383
Push a Package Upgrade for Second-Generation Managed Packages	383
Schedule a Push Upgrade Using SOAP API for First- and Second-Generation Managed Packages	384
Assign Access to New and Changed Features in First- and Second-Generation Managed Packages	384
Sample Post Install Script for a Push Upgrade for First- and Second-Generation Managed Packages	385
Advanced Features for Second-Generation Managed Packages	386
Package Ancestors for Second-Generation Managed Packages	388

Contents

Patch Versions for Second-Generation Managed Packages	392
Create Dependencies Between Second-Generation Managed Packages	392
Advanced Project Configuration Parameters for Second-Generation Managed Packages	395
Sample Script for Installing Second-Generation Managed Packages with Dependencies	398
Skip Validation to Quickly Iterate Second-Generation Managed Package Development	400
Second-Generation Managed Packaging Keywords	400
Target a Specific Release for Your Second-Generation Managed Packages During Salesforce Release Transitions	401
Use Branches in Second-Generation Managed Packaging	402
Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages	403
Package IDs and Aliases for Second-Generation Managed Packages	404
Avoid Namespace Collisions in Second-Generation Managed Packages	405
Customize Second-Generation Managed Package Installs and Uninstalls Using Scripts	407
Behavior of Specific Metadata in Second-Generation Managed Packages	407
Remove Metadata Components from Second-Generation Managed Packages	415
Delete a Second-Generation Managed Package or Package Version	418
Frequently Used Packaging Operations for Second-Generation Managed Packages	419
Transfer a Second-Generation Managed Package to a Different Dev Hub	420
Best Practices for Second-Generation Managed Packages	424
Manage Licenses for Managed Packages	425
Get Started with the License Management App	426
Lead and License Records in the License Management App	429
Modify a License Record	430
Refresh Licenses for a Managed Package	430
Extending the License Management App	430
Move the License Management App to Another Salesforce Org	434
Troubleshoot the License Management App	434
Best Practices for the License Management App	435
Troubleshoot Subscriber Issues	436
Manage Features in Second-Generation Managed Packages	438
Feature Parameter Metadata Types and Custom Objects	439
Set Up Feature Parameters	439
Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features	442
Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters ..	443
Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs	443
Best Practices for Feature Management	443
Considerations for Feature Management	444
Gaps Between First-Generation and Second-Generation Managed Packaging	444

Contents

Chapter 14: Partner Licensing Platform (Developer Preview)	445
Enable the Partner Licensing Platform (Developer Preview)	447
Quick Start: Get Started with the Partner Licensing Platform (Developer Preview)	448
Partner Licensing Platform Components and Concepts (Developer Preview)	450
Licensing Components (Developer Preview)	453
Entitlements and Grants (Developer Preview)	455
Design Your Package Licensing Structure (Developer Preview)	457
Outline Pricing Feature Strategy (Developer Preview)	457
Identify Licenses (Developer Preview)	459
Identify License Settings Use Cases (Developer Preview)	460
Map Settings to Licenses (Developer Preview)	468
Design Permission Sets (Developer Preview)	469
Review Your Structure (Developer Preview)	473
Implement Your Package Licensing (Developer Preview)	474
Create Licensed Custom Permissions (Developer Preview)	475
Define Custom Permission Set Licenses (Developer Preview)	476
Create Custom Permission Sets (Developer Preview)	481
Create Custom Permission Set Groups (Developer Preview)	481
Create Access Checks in Apex (Developer Preview)	482
Deploy Your Licensing Structure and Create Your Package (Developer Preview)	483
Test Your Licensing Structure (Developer Preview)	483
Best Practices for Testing Your Package Licensing (Developer Preview)	484
Example: Test Plan for Validating License Design (Developer Preview)	484
Manage Migrations to the Partner Licensing Platform (Developer Preview)	486
Manage a Hybrid Licensing Model During Migrations (Developer Preview)	487
Licensing for Existing Custom Permissions (Developer Preview)	487
Package Access Validation in a Hybrid Licensing Model (Developer Preview)	488
Chapter 15: Unlocked Packages	489
What's a Package	490
Package-Based Development Model	490
Before You Create Unlocked Packages	490
Know Your Orgs	491
Create Org-Dependent Unlocked Packages	492
Workflow for Unlocked Packages	493
Configure Packages	494
Project Configuration File for Unlocked Packages	495
Unlocked Packaging Keywords	500
Package Installation Key	501
Extract Dependency Information from Unlocked Packages	501
Understanding Namespaces	503
Share Release Notes and Post-Install Instructions	507
Specify Unpackaged Metadata or Apex Access for Apex Tests (Unlocked Packages)	507
Best Practices for Unlocked Packages	508

Contents

Package IDs and Aliases for Unlocked Packages	509
Frequently Used Packaging Operations	510
How We Handle Profile Settings in Unlocked Packages	510
Create a Package	511
Generate the Package	512
Generate a Package Version	513
Code Coverage for Unlocked Packages	517
Release an Unlocked Package	518
Update a Package Version	518
Hard-Deleted Components in Unlocked Packages	519
Delete an Unlocked Package or Package Version	523
View Package Details	524
Push a Package Upgrade	525
Install a Package	525
Install Packages with the CLI	525
Install Packages from a URL	527
Upgrade a Package Version	527
Sample Script for Installing Packages with Dependencies	528
Migrate Deprecated Metadata from Unlocked Packages	530
Uninstall a Package	530
Transfer an Unlocked Package to a Different Dev Hub	531
Take Ownership of an Unlocked Package Transferred from a Different Dev Hub	533
Chapter 16: Continuous Integration	536
Continuous Integration Using CircleCI	537
Configure Your Environment for CircleCI	537
Connect CircleCI to Your DevHub	538
Continuous Integration Using Jenkins	539
Configure Your Environment for Jenkins	540
Jenkinsfile Walkthrough	541
Sample Jenkinsfile	547
Continuous Integration with Travis CI	552
Sample CI Repos for Org Development Model	552
Sample CI Repos for Package Development Model	552
Chapter 17: Troubleshoot Salesforce DX	554
Error: API Version Mismatch	555
CLI Version Information	555
Run CLI Commands on macOS Sierra (Version 10.12)	555
Error: No defaultdevhubusername org found	556
Unable to Work After Failed Org Authorization	556
Error: Lightning Experience-Enabled Custom Domain Is Unavailable	557
Chapter 18: Limitations for Salesforce DX	558

CHAPTER 1 Salesforce DX Release Notes

Use the Salesforce Release Notes to learn about the most recent updates and changes to development environments, packaging, platform development tools, and Salesforce APIs.

For the latest changes, visit:

- [Salesforce Extensions for Visual Studio Code Release Notes](#)
- [Salesforce CLI Release Notes](#)
- [Development Environments Release Notes](#) (Includes Developer Edition orgs, sandboxes, and scratch orgs)
- [Packaging Release Notes](#)
- [New and Changed Items for Developers](#) (Includes Apex, standard objects, Metadata API, and more)

CHAPTER 2 How Salesforce Developer Experience Changes the Way You Work

In this chapter ...

- [Use a Sample Repo to Get Started](#)
- [Create an Application](#)
- [Migrate or Import Existing Source](#)

Salesforce Developer Experience (DX) is a new way to manage and develop apps on the Lightning Platform across their entire lifecycle. It brings together the best of the Lightning Platform to enable source-driven development, team collaboration with governance, and new levels of agility for custom app development on Salesforce.

Highlights of Salesforce DX include:

- Your tools, your way. With Salesforce DX, you use the developer tools you already know.
- The ability to apply best practices to software development. Source code and metadata exist outside of the org and provide more agility to develop Salesforce apps in a team environment. Instead of the org, your version control system is the source of truth.
- A powerful command-line interface (CLI) removes the complexity of working with your Salesforce org for development, continuous integration, and delivery.
- Flexible and configurable scratch orgs that you build for development and automated environments. This new type of org makes it easier to build your apps and packages.
- You can use any IDE or text editor you want with the CLI and externalized source.
- [Salesforce Extensions for VS Code](#) to accelerate app development. These tools provide features for working with scratch orgs, Apex, Lightning components, and Visualforce.

Are You Ready to Begin?

Here's the basic order for doing your work using Salesforce DX. These workflows include the most common CLI commands. For all commands, see the *Salesforce CLI Command Reference*.

- [Install Salesforce CLI](#)

Salesforce CLI is a command-line interface that simplifies development and build automation when working with Salesforce. It's a bundle of two executables: `sfc` and `sfdx`. We first launched `sfdx` to provide you the ability to develop and test your apps more easily on Salesforce Platform. But if you want to work across *all* Salesforce clouds, `sfdx` doesn't provide all the commands you need. With `sfc`, we're bringing together a cross-cloud set of commands that streamline how you build and deploy across Salesforce.

This developer guide uses the `sfdx` executable in all its examples. Some of the `sfdx` commands have `sfc` equivalents. We encourage you to try them out! See [Get Started with Salesforce CLI Unification](#) for more information about the `sfc` executable, such as how to install it, how you can use the two executables together, and a mapping between equivalent commands.

- [Enable Dev Hub](#)
- [Use a Sample Repo to Get Started](#)

How Salesforce Developer Experience Changes the Way You Work

- [Create an Application](#)
- [Migrate or Import Existing Source](#)

SEE ALSO:

[Salesforce Developer Tooling Learning Map](#)

[*Salesforce CLI Command Reference*](#)

The quickest way to get going with Salesforce DX tooling is to clone the `dreamhouse-lwc` GitHub repo. Use its configuration files and Salesforce application to try some commonly used Salesforce CLI commands. In addition to source code for the application, the repo includes sample data and Apex tests.

1. Open a terminal or command prompt window, and clone the `dreamhouse-lwc` GitHub sample repo using HTTPS or SSH.
- HTTPS:

```
git clone https://github.com/trailheadapps/dreamhouse-lwc.git
```

SSH:

```
git clone git@github.com:trailheadapps/dreamhouse-lwc.git
```

2. Change to the `dreamhouse-lwc` project directory.

```
cd dreamhouse-lwc
```

3. Authorize your Dev Hub org by logging into it, set it as your default, and assign it an alias.

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

Enter your Dev Hub org credentials in the browser that opens. After you log in successfully, you can close the browser.

4. Create a scratch org using the `config/project-scratch-def.json` file, set the org as your default, and assign it an alias.

```
sf org create scratch --definition-file config/project-scratch-def.json --set-default --alias my-scratch-org
```

The command uses the default Dev Hub you set with the `sf org login web` command in a previous step.

5. View the orgs that you've either created or logged into.

```
sf org list
```

The first table displays the Dev Hub you logged into and the second table displays the scratch org you created. The right-most column in both tables indicates the default scratch org and Dev Hub org with (U) and (D), respectively. The ALIAS column displays the aliases you assigned each org. Here's some sample output.

Non-scratch orgs

	ALIAS	USERNAME	ORG ID	CONNECTED STATUS
	—	—	—	—
(D)	DevHub	jules@sf.com	00DB0000000c7j	Connected

Scratch orgs

	ALIAS	USERNAME	ORG ID	EXPIRATION DATE
(U)	my-scratch-org	test-ibnpzayw@example.com	00D9A000000EFO	2023-05-12

6. Deploy the Dreamforce app, whose source is in the `force-app` directory, to the scratch org.

```
sf project deploy start --source-dir force-app
```

7. Assign the `dreamhouse` permission set to the default scratch org user (`test-ibnpzayw@example.com`).

```
sf org assign permset --name dreamhouse
```

8. Import sample data from three objects (Contact, Property, and Broker) into the scratch org using the specified plan definition file.

```
sf data import tree --plan data/sample-data-plan.json
```

9. Run Apex tests.

```
sf apex run test --result-format human --wait 1
```

Apex tests run asynchronously by default. If the tests finish before the `--wait` value, the results are displayed. Otherwise, use the displayed command to get the results using a job ID.

10. Open the scratch org and view the deployed metadata under Most Recently Used.

```
sf org open
```

11. In App Launcher, find and open the Dreamhouse application.

Congrats! You just deployed an application to a new scratch org.

SEE ALSO:

[Sample Repository on GitHub](#)

[Authorization](#)

[Create Scratch Orgs](#)

[Push Source to the Scratch Org](#)

[Run Apex Tests](#)

Create an Application

Follow the basic workflow when you are starting from scratch to create and develop an app that runs on the Lightning Platform.

1. [Set up your project.](#)
2. [Authorize the Developer Hub org for the project.](#)
3. [Configure your local project.](#)
4. [Create a scratch org.](#)
5. [Push the source from your project to the scratch org.](#)
6. [Develop the app.](#)
7. [Pull the source to keep your project and scratch org in sync.](#)
8. [Run tests.](#)
9. Add, commit, and push changes. Create a pull request.

Deploy your app using one of the following methods:

- Build and release your app with managed packages
- Build and release your app using the Metadata API

Migrate or Import Existing Source

Use the Metadata API to retrieve the code, and then convert your source for use in a Salesforce DX project.

 **Tip:** If your current repo follows the directory structure that is created from a Metadata API retrieve, you can skip the retrieve step and go directly to converting the source.

1. Set up your project.
2. Retrieve your metadata.
3. Convert the metadata formatted source you just retrieved to source format.
4. Authorize the Developer Hub org for the project.
5. Configure your local project.
6. Create a scratch org.
7. Push the source from your project to the scratch org.
8. Develop the app.
9. Pull the source to sync your project and scratch org.
10. Run tests.

11. Add, commit, and push changes. Create a pull request.

Deploy your app using one of the following methods:

- Build and release your app with managed packages.
- Build and release your app using the Metadata API.

CHAPTER 3 Enable Dev Hub Features in Your Org

In this chapter ...

- [Free Limited Access License](#)
- [Enable Unlocked and Second-Generation Managed Packaging](#)
- [Enable Source Tracking in Sandboxes](#)
- [Enable Einstein Features](#)
- [Enable Language Extension Packages \(Beta\)](#)
- [Add Salesforce DX Users](#)

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

It's not necessary to enable Dev Hub if you plan to use Salesforce CLI with only sandboxes unless you plan to create second-generation (2GP) packages. The 2GP packages use a scratch org during the package generation process.

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

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

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

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

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

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

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

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

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

Enable Dev Hub Features in Your Org

To enable Dev Hub in an org:

1. Log in as System Administrator to your Developer Edition, trial, or production org (for customers), or your business org (for ISVs).

2. From Setup, enter *Dev Hub* in the Quick Find box and select **Dev Hub**.

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

3. To enable Dev Hub, click **Enable**.

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

Free Limited Access License

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

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

Scratch Org Creation and Second-Generation Packages

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

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

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

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

- System Permissions > Create and Update Second-Generation Packages

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

DevOps Center

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

See Salesforce Help: [Assign the DevOps Center Permission Sets](#)

Features Not Currently Supported

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

SEE ALSO:

[Salesforce Help: Add Team Members as Users in the DevOps Center Org](#)

Enable Unlocked and Second-Generation Managed Packaging

Enable packaging in your org so you can develop unlocked packages or second-generation managed packages. You can work with the packages in scratch orgs, sandbox orgs, and target subscriber orgs.

Enable Dev Hub in your org.

1. Log in to the org where you've enabled Dev Hub.
2. From Setup, enter *Dev Hub* in the Quick Find box and select **Dev Hub**.
3. Select **Enable Unlocked Packages and Second-Generation Managed Packages**.
After you enable Second-Generation Packaging, you can't disable it.
4. (Optional) Allow non-admin users access to the Dev Hub to create packages.
Assign non-admin users the Create and Update Second Generation Packages user permission. See [Add Salesforce DX Users](#) for details.

Enable Source Tracking in Sandboxes

Turn on source tracking in the source (production) org so Developer and Developer Pro sandboxes automatically track changes between sandboxes created from it and Salesforce DX projects.

To enable Source Tracking in Sandboxes for Developer and Developer Pro sandboxes:

1. Log in to the source (production) org.
2. From Setup, in the Quick Find Box, enter *Dev Hub* and select **Dev Hub**.
If you don't see Dev Hub in the Setup menu, make sure that the source org is one of the supported editions.
3. Select **Enable Source Tracking in Developer and Developer Pro Sandboxes**.

After you enable this setting, Developer and Developer Pro Sandboxes that are created or refreshed have source tracking enabled. Existing sandboxes don't have source tracking enabled until you refresh them.

Enable Einstein Features

Turn on Einstein Features in your Dev Hub to eliminate the manual steps for enabling the Chatbot feature in scratch orgs. When you accept the Terms of Service for Einstein, a separate acceptance is not required in each scratch org created from this Dev Hub org. If you previously accepted the Terms of Service for Einstein to turn on an Einstein-related feature, this setting is already enabled.

Complete this task before attempting to create a scratch org with the Chatbot feature.

1. Log in to your Dev Hub org.
2. From Setup, enter *Dev Hub* in the Quick Find box and select **Dev Hub**.
3. On the Dev Hub Setup page, turn on **Enable Einstein Features**.

Enable Language Extension Packages (Beta)

Enable Language Extension Packages in Dev Hub to create language extension packages that contain translations of components in other packages. This feature is available in unlocked and first- and second-generation managed 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](#).

Editions

Available in: Enterprise, Performance, and Unlimited Editions

User Permissions

To view a sandbox:

- View Setup and Configuration

To create, refresh, activate, and delete a sandbox:

- Manage Sandbox

To enable Source Tracking:

- Customize Application

Language extension packages can only contain 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.

1. In Dev Hub, from Setup, in the Quick Find box, enter *Dev Hub*, and then select **Dev Hub**.
2. On the Dev Hub Setup page, turn on **Enable Language Extension Packages**.

Add Salesforce DX Users

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

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

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

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

Using a Developer license, add a user with the Developer profile and assign them the Developer permission set.

[Add a System Administrator or Standard User to Your Dev Hub Org](#)

Add system administrator users only if their work requires that level of authority. Otherwise, add standard users and create a permission set with the required Salesforce DX permissions.

[Permission Set for Salesforce DX Users](#)

To give full access to the Dev Hub org, the Developer permission set or the custom permission set you create grants access to specific Salesforce DX objects.

SEE ALSO:

[Org Shape Permissions](#)

Add a Developer User to Your Dev Hub Org

Using a Developer license, add a user with the Developer profile and assign them the Developer permission set.

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 **Developer** for User License, and then **Developer** for Profile.
 - e. After filling out the remaining information, click **Save**.
2. Assign the built-in Developer permission set to the user.
 - a. On the user's detail page, in the Permission Set Assignments related list, click **Edit Assignments**.
 - b. In the Available Permission Sets, add the Developer permission set and click **Save**.

The Developer permission set grants access to Dev Hub features and second-generation packages. For details, see [Permission Set for Salesforce DX Users](#).

Add a System Administrator or Standard User to Your Dev Hub Org

Add system administrator users only if their work requires that level of authority. Otherwise, add standard users and create a permission set with the required Salesforce DX permissions.

1. Create a user in your Dev Hub org, if necessary.
 - a. In Setup, enter *Users* in the Quick Find box, then select **Users**.
 - b. Click **New User**.
 - c. Fill out the form, and assign the System Administrator or Standard User profile.
 - d. Click **Save**.

If you're adding a System Administrator user, you can stop here.
2. If you're adding a Standard User, create a permission set for Salesforce DX users if you don't have one.
 - a. From Setup, enter *Permission Sets* in the Quick Find box, then select **Permission Sets**.
 - b. Click **New**.
 - c. Enter a label, API name, and description. The API name is a unique name used by the API and managed packages.
 - d. Select a user license option. If you plan to assign this permission set to multiple users with different licenses, select **None**.
 - e. Click **Save**. The permission set overview page appears. From here, you can navigate to the permissions you want to add or change for Salesforce DX. For the required permissions, see [Permission Set for Salesforce DX Users](#).
3. Apply the Salesforce DX permission set to the Standard User.
 - a. From Setup, enter *Permission Sets* in the Quick Find box, then select **Permission Sets**.
 - b. Select the Salesforce DX permission set.
 - c. In the permission set toolbar, click **Manage Assignments**.
 - d. Click **Add Assignments**.
 - e. Select the user to assign the permission set to.
 - f. Click **Assign**.
 - g. Click **Done**.

You can limit a user's access by modifying the permissions.

Permission Set for Salesforce DX Users

To give full access to the Dev Hub org, the Developer permission set or the custom permission set you create grants access to specific Salesforce DX objects.

- Object Settings > Scratch Org Infos > Read, Create, Edit, and Delete
- Object Settings > Active Scratch Orgs > Read, Edit, and Delete
- Object Settings > Namespace Registries > Read

To work with second-generation packages in the Dev Hub org, the permission set must also contain:

- System Permissions > Create and Update Second-Generation Packages

This permission provides access to:

Salesforce CLI Command	Tooling API Object (Create and Edit)
force:package:create	Package2
force:package:version:create	Package2VersionCreateRequest
force:package:version:update	Package2Version

CHAPTER 4 Project Setup

In this chapter ...

- [Sample Repository on GitHub](#)
- [Create a Salesforce DX Project](#)
- [Salesforce DX Project Structure and Source Format](#)
- [How to Exclude Source When Syncing or Converting](#)
- [Create a Salesforce DX Project from Existing Source](#)
- [Retrieve Source from an Existing Managed Package](#)
- [Retrieve Unpackaged Source Defined in a package.xml File](#)
- [Convert the Metadata Source to Source Format](#)
- [Salesforce DX Usernames and Orgs](#)
- [Override or Add Definition File Options at the Command Line](#)
- [Link a Namespace to a Dev Hub Org](#)
- [Salesforce DX Project Configuration](#)
- [Multiple Package Directories](#)
- [Replace Strings in Code Before Deploying](#)

Salesforce DX introduces a new project structure for your org's metadata (code and configuration), your org templates, your sample data, and all your team's tests. Store these items in a version control system (VCS) to bring consistency to your team's development processes. Retrieve the contents of your team's repository when you're ready to develop a new feature.

You can use your preferred VCS. Most of our examples use Git.

You have different options to create a Salesforce DX project depending on how you want to begin.

Use the Sample Repository on GitHub	Explore the features of Salesforce DX using one of our sample repos and your own VCS and toolset.
Create a Salesforce DX Project from Existing Source	Start with an existing Salesforce app to create a Salesforce DX project.
Create a Salesforce DX Project	Create an app on the Lightning Platform using a Salesforce DX project.

Sample Repository on GitHub

To get started quickly, see the `dreamhouse-lwc` GitHub repo. This standalone application contains an example DX project with multiple Apex classes, Aura components, custom objects, sample data, and Apex tests.

Cloning this repo creates the directory `dreamhouse-lwc`. See the repo's Readme for more information.

Assuming that you've already set up Git, use the `git clone` command to clone the master branch of the repo from the command line.

To use HTTPS:

```
git clone https://github.com/trailheadapps/dreamhouse-lwc.git
```

To use SSH:

```
git clone git@github.com:trailheadapps/dreamhouse-lwc.git
```

If you don't want to use Git, download a .zip file of the repository's source using Clone, or download on the GitHub website. Unpack the source anywhere on your local file system.

 **Tip:** Check out more complex examples in the [Sample Gallery](#).

It contains sample apps that show what you can build on the Salesforce platform. They're continuously updated to incorporate the latest features and best practices.

Create a Salesforce DX Project

A Salesforce DX project has a specific structure and a configuration file that identifies the directory as a Salesforce DX project.

Before you create your project, first decide if you're following org-based or package-based project development model.

If you're following org-based development, change to the directory where you want the DX project located. Then run `sfdx force:project:create -n MyProject --manifest` to generate your project with a default manifest (`package.xml`) file.

If you're following package-based development, you can create a project with minimal (empty) or expanded (standard) scaffolding. The default is `standard`, which provides extended scaffolding to facilitate moving source to and from your orgs.

1. Change to the directory where you want the DX project located.
2. Create the DX project.

```
sfdx force:project:create -n MyProject
```

If you don't indicate an output directory, the project directory is created in the current location. You can also specify the default package directory to target when syncing source to and from the scratch org. If you don't indicate a default package directory, this command creates a default package directory, `force-app`.

If you don't choose a template type, the default is `--template standard`. The standard template provides a complete directory structure that takes the guesswork out of where to put your source. It also provides these files that are especially helpful when using Salesforce Extensions for VS Code.

- `.gitignore`: Makes it easier to start using Git for version control.
- `.prettierrc` and `.prettierignore`: Make it easier to start using Prettier to format your Aura components.
- `.vscode/extensions.json`: Causes Visual Studio Code, when launched, to prompt you to install the recommended extensions for your project.

- `.vscode/launch.json`: Configures Replay Debugger, making it more discoverable and easier to use.
- `.vscode/settings.json`: By default, this file has one setting, for push or deploy on save, which is set to false. You can change this value or add other settings.

If you choose `--template empty`, your project contains these sample configuration files to get you started.

- `.forceignore`
- `config/project-scratch-def.json`
- `sfdx-project.json`
- `package.json`

If you choose `--template analytics`, you get all helpful VS Code files but the project scaffolding contains only one directory: `/force-app/main/default/waveTemplates`.



Example:

```
sfdx force:project:create --projectname mywork --template standard
```

```
sfdx force:project:create --projectname mywork --defaultpackagedir myapp
```

Next steps:

- (Optional) Register the namespace with the Dev Hub org.
- Configure the project (`sfdx-project.json`). If you use a namespace, update this file to include it.
- Create a scratch org definition that produces scratch orgs that mimic the shape of another org you use in development, such as sandbox, packaging, or production. The `config` directory of your new project contains a sample scratch org definition file (`project-scratch-def.json`).

SEE ALSO:

[Create a Salesforce DX Project from Existing Source](#)

[Salesforce DX Project Configuration](#)

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

[Build Your Own Scratch Org Definition File](#)

[How to Exclude Source When Syncing or Converting](#)

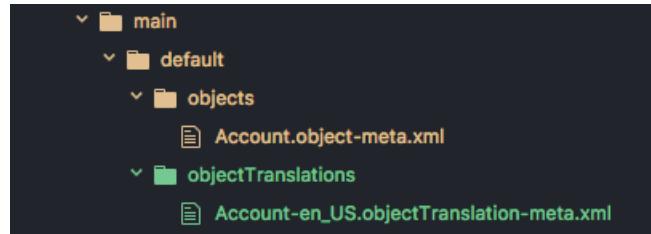
Salesforce DX Project Structure and Source Format

A Salesforce DX project has a specific project structure and source format. Source format uses a different set of files and file extensions from what you're accustomed when using Metadata API.

Source Transformation

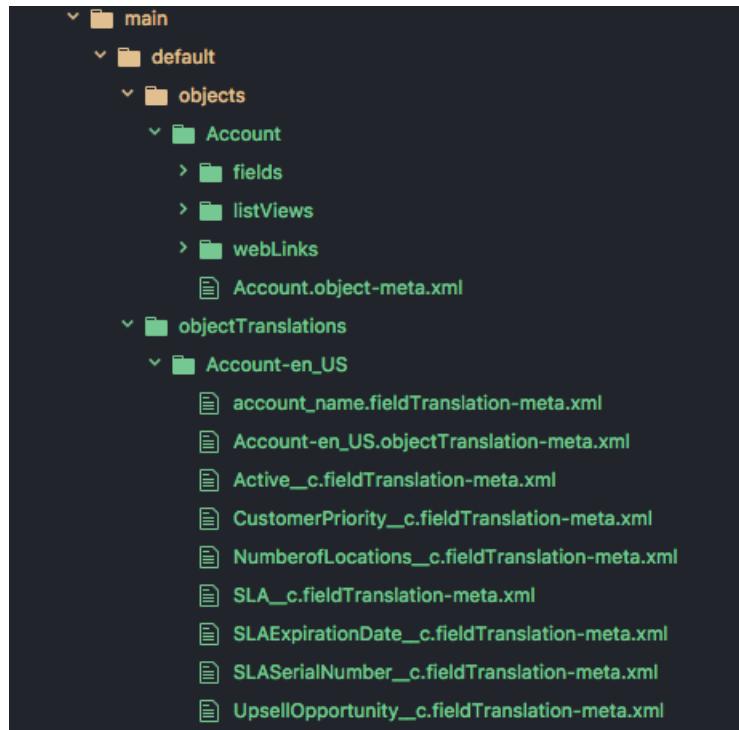
It's not uncommon for metadata formatted source to be very large, making it difficult to find what you want. If you work on a team with other developers who update the same metadata at the same time, you have to deal with merging multiple updates to the file. If you're thinking that there has to be a better way, you're right.

Before, all custom objects and object translations were stored in one large metadata file.



We solve this problem by providing a new source shape that breaks down these large source files to make them more digestible and easier to manage with a version control system. It's called source format.

A Salesforce DX project stores custom objects and custom object translations in intuitive subdirectories. Source format makes it much easier to find what you want to change or update. And you can say goodbye to messy merges.

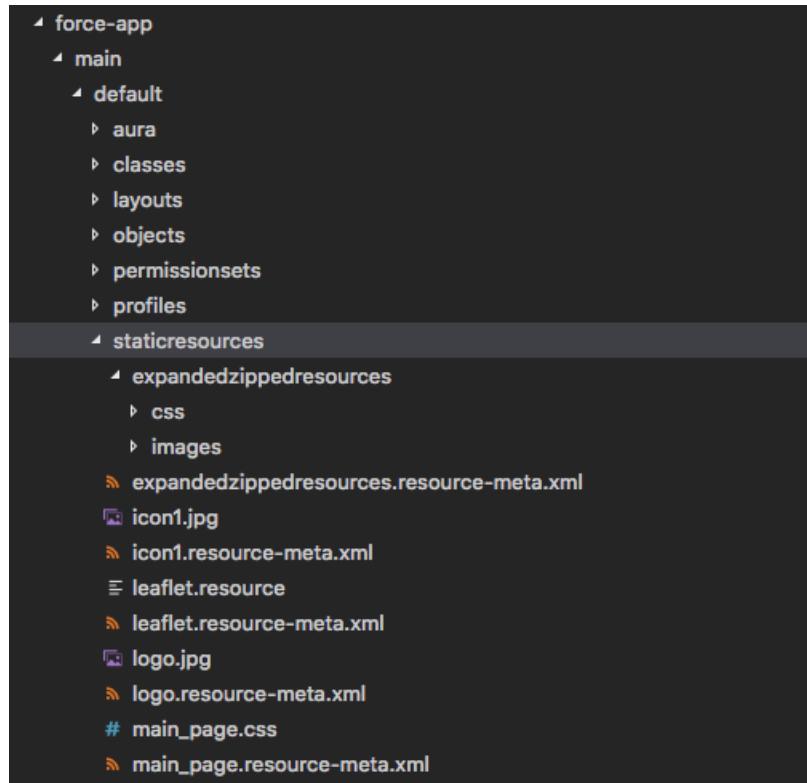


Static Resources

Static resources must reside in the `/main/default/staticresources` directory. The `force:source:push` and `force:source:pull` commands support auto-expanding or compressing archive MIME types within your project. These behaviors support both the `.zip` and `.jar` MIME types. This way, the source files are more easily integrated in your Salesforce DX project and version control system.

If, for example, you upload a static resource archive through the scratch org's Setup UI, `force:source:pull` expands it into its directory structure within the project. To mimic this process from the file system, add the directory structure to compress directly into the static resources directory root, then create the associated `.resource-meta.xml` file. If an archive exists as a single file in your project, it's always treated as a single file and not expanded.

This example illustrates how different types of static resources are stored in your local project. You can see an expanded `.zip` archive called `expandedzippedresource` and its related `.resource-meta.xml` file. You also see a couple `.jpg` files being stored with their MIME type, and a single file being stored with the legacy `.resource` extension



File Extensions

When you convert existing metadata format to source format, we create an XML file for each bit. All files that contain XML markup now have an `.xml` extension. You can then look at your source files using an XML editor. To sync your local projects and scratch orgs, Salesforce DX projects use a particular directory structure for custom objects, custom object translations, Lightning web components, Aura components, and documents.

For example, if you had an object called `Case.object`, source format provides an XML version called `Case.object-meta.xml`. If you have an app call `DreamHouse.app`, we create a file called `DreamHouse.app-meta.xml`. You get the idea.

Traditionally, static resources are stored on the file system as binary objects with a `.resource` extension. Source format handles static resources differently by supporting content MIME types. For example, `.gif` files are stored as a `.gif` instead of `.resource`. By storing files with their MIME extensions, you can manage and edit your files using the associated editor on your system.

You can have a combination of existing static resources with their `.resource` extension, and newly created static resources with their MIME content extensions. Existing static resources with `.resource` extensions keep that extension, but any new static resources show up in your project with their MIME type extensions. We allow `.resource` files to support the transition for existing customers. Although you get this additional flexibility, we recommend storing your files with their MIME extensions.

Custom Objects

When you convert from metadata format to source format, your custom objects are placed in the `<package directory>/main/default/objects` directory. Each object has its own subdirectory that reflects the type of custom object. Some parts of the custom objects are extracted into these subdirectories:

- businessProcesses
- compactLayouts
- fields
- fieldSets
- listViews
- recordTypes
- sharingReasons
- validationRules
- webLinks

The parts of the custom object that aren't extracted are placed in a file.

- For objects, `<object>.object-meta.xml`
- For fields, `<field_name>.field-meta.xml`

Custom Object Translations

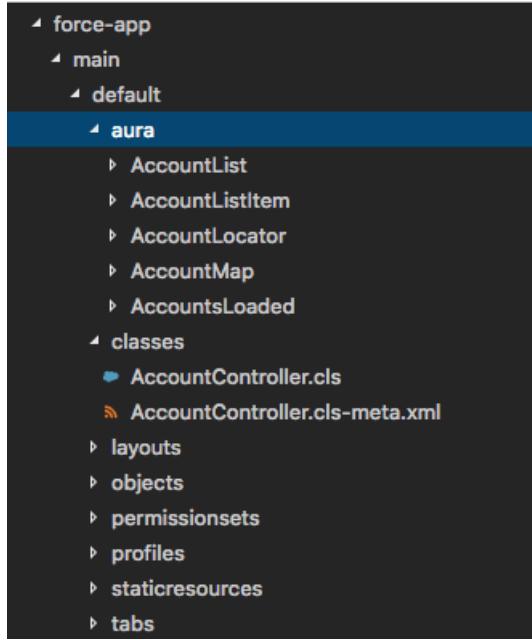
Custom object translations reside in the `<package directory>/main/default/objectTranslations` directory, each in their own subdirectory named after the custom object translation. Custom object translations and field translations are extracted into their own files within the custom object translation's directory.

- For field names, `<field_name>.fieldTranslation-meta.xml`
- For object names, `<object_name>.objectTranslation-meta.xml`

The remaining pieces of the custom object translation are placed in a file called `<objectTranslation>.objectTranslation-meta.xml`.

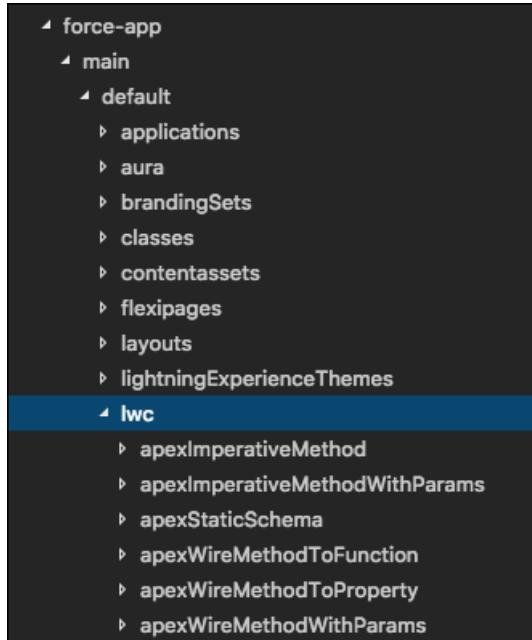
Aura Components

Aura bundles and components must reside in a directory named `aura` under the `<package directory>` directory.



Lightning Web Components

Lightning web components must reside in a directory named `lwc` under the `<package_directory>` directory.

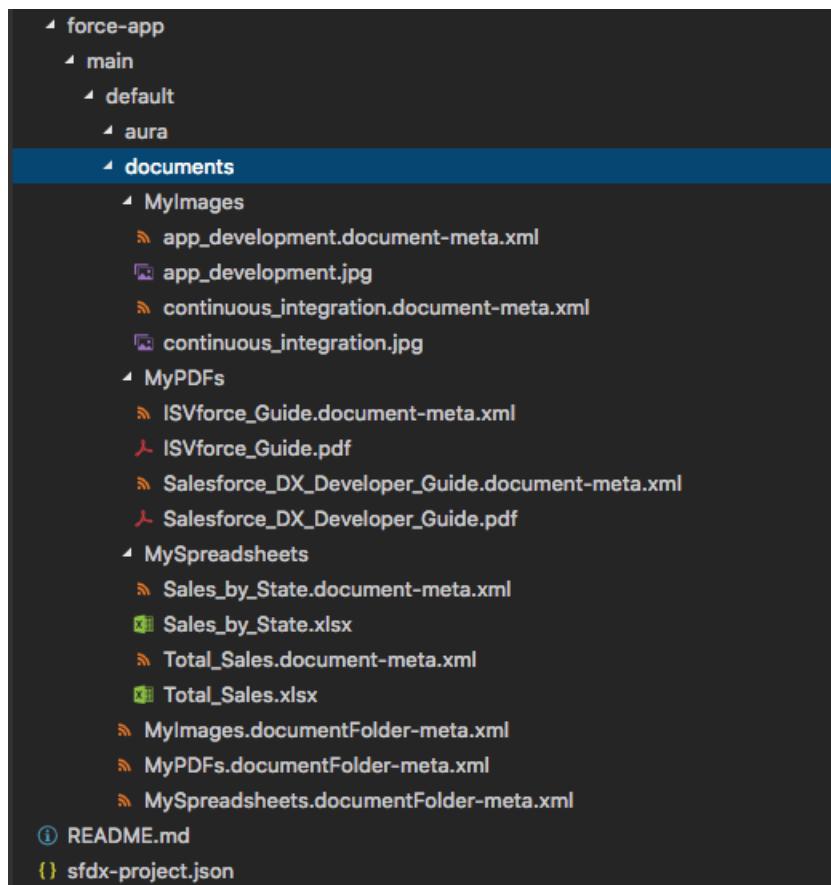


ExperienceBundle for Lightning Communities

The ExperienceBundle metadata type must reside in a directory named `experiences` under the `<package_directory>` directory. The `experiences` directory contains a folder for each Lightning community in your org. See the [Lightning Communities Developer Guide](#) for details.

Documents

Documents must be inside the directories of their parent document folder. The parent document folder must be in a directory called `documents`. Each document has a corresponding metadata XML file that you can view with an XML editor.



Custom Labels

All custom labels are contained in a single file called `CustomLabel.labels-meta.xml` that resides in a directory named `labels` under the `<package_directory>` directory. Each package directory has its own `CustomLabel.labels-meta.xml` file. We think it's easier for you to maintain your custom labels in a single file rather than decomposed into individual files. It also minimizes file I/O performance degradation, especially on Windows.

How to Exclude Source When Syncing or Converting

When syncing metadata between your local file system and a target org, you often have source files you want to exclude. Similarly, you often want to exclude certain files when converting source to Salesforce DX source format. In both cases, you can exclude individual files or all files in a specific directory with a `.forceignore` file.

The `.forceignore` file excludes files when running all the `force:source:*` commands, such as `force:source:push`, `force:source:pull`, `force:source:deploy`, and `force:source:retrieve`.

Structure of the `.forceignore` File

The `.forceignore` file structure mimics the `.gitignore` structure. Each line in `.forceignore` specifies a pattern that corresponds to one or more files. The files typically represent metadata components, but can be any files you want to exclude, such as LWC configuration JSON files or tests.

The `force:source:*` commands, when parsing the `.forceignore` file, use the same rules and patterns as the `.gitignore` file. A few common examples of these rules and patterns include:

- Always use the forward slash (/) as a directory separator, even on operating systems that use back slashes, such as Microsoft Windows.
- An asterisk (*) matches anything except a forward slash (/).
- Two consecutive asterisks (**) in patterns have special meaning, depending on where they're located in the pathname. See [for examples](#).
- For readability, use blank lines as separators in the `.forceignore` file.

There are many more rules and patterns. See the [git documentation](#) for details.

Determine the Exact Filename for a Metadata Component

As you build your `.forceignore` file, you need the exact name of the metadata components that you want to exclude. Use this pattern to determine the filename of a particular metadata component:

```
<component-API-name>.<md-type-file-suffix>-meta.xml
```

- `<component-API-name>` is the unique name of the component used by the Salesforce APIs. API names contain only alphanumeric characters or underscores.
- `<md-type-file-suffix>` is the file suffix for the metadata type. See the *Declarative Metadata File Suffix and Directory Location* section for the specific metadata type in the [Metadata API Developer Guide](#).

For example, the filename for the [profile](#) with API name `NotUsedProfile` is `NotUsedProfile.profile-meta.xml`. To specify that the `force:source:*` commands exclude this component, add this entry to your `.forceignore`:

```
**/NotUsedProfile.profile-meta.xml
```

Another way to determine the exact name of a metadata component is to look at the output of the `force:source:*` commands if you're also using source tracking. For example, if you have either local or remote changes, run the `force:source:status` command to display the full pathname of the changed components. This command output displays the filename of the `Dreamhouse` permission set and the `Settings` custom tab in the PROJECT PATH column:

STATE	FULL NAME	TYPE	PROJECT PATH
Local Add	Settings	CustomTab	./tabs/Settings.tab-meta.xml

```
Local Add     Dreamhouse      PermissionSet  
.permissionsets/Dreamhouse.permissionset-meta.xml
```

Other Files That the Source Commands Ignore

The source commands ignore these files even if they aren't included in your `.forceignore` file:

- Any source file or directory that begins with a "dot", such as `.DS_Store` or `.sfdx`
- Any file that ends in `.dup`
- `package2-descriptor.json`
- `package2-manifest.json`

Exclude Remote Changes Not Yet Synced with Your Local Source

Sometimes, you make a change directly in a scratch org but you don't want to pull that change into your local DX project. To exclude remote metadata changes, add an entry to `.forceignore` that represents the metadata source file that would be created if you *did* retrieve it.

For example, if you have a permission set named `Dreamhouse`, add this entry to `.forceignore`:

```
**/Dreamhouse.permissionset-meta.xml
```

Exclude MetadataWithContent Types

Metadata components that include content, such as `ApexClass` or `EmailTemplate`, extend the [MetadataWithContent](#) type. These components have two source files: one for the content itself, such as the Apex code or email template, and the accompanying metadata file. For example, the source files for the `HelloWorld` Apex class are `HelloWorld.cls` and `HelloWorld.cls-meta.xml`.

To exclude a `MetadataWithContent` component, such as an `ApexClass`, either list both source files in the `.forceignore` file, or use an asterisk. For example:

```
# Explicitly list the HelloWorld source files to be excluded  
helloWorld/main/default/HelloWorld.cls  
helloWorld/main/default/HelloWorld.cls-meta.xml  
  
# Exclude the HelloWorld Apex class using an asterisk  
helloWorld/main/default/HelloWorld.cls*
```

Exclude Bundles and File Groups

Use two consecutive asterisks (`**`) to exclude multiple files with just one `.forceignore` entry.

For example, to exclude all resource files related to a Lightning web component named `myLwcComponent`, add this entry to exclude the entire component bundle:

```
**/lwc/myLwcComponent
```

To exclude all Apex classes:

```
**/classes
```

Metadata with Special Characters

If a metadata name has special characters (such as forward slashes, backslashes, or quotation marks), we encode the file name on the local file system for all operating systems. For example, if you pull a custom profile called Custom: Marketing Profile, the colon is encoded in the resulting file name.

```
Custom%3A Marketing Profile.profile-meta.xml
```

If you reference a file name with special characters in `.forceignore`, use the encoded file name.

Where to Put `.forceignore`

Be sure the paths that you specify in `.forceignore` are relative to the directory containing the `.forceignore` file. For the `.forceignore` file to work its magic, you must put it in the proper location, depending on which command you're running.

- Add the `.forceignore` file to the root of your project for the `force:source:*` tracking commands.
- Add the file to the Metadata retrieve directory (with `package.xml`) for `force:mdapi:convert`.

Sample Syntax

Here are some options for indicating which source to exclude. In this example, all paths are relative to the project root directory.

```
# Specify a relative path to a directory from the project root
helloWorld/main/default/classes

# Specify a wildcard directory - any directory named "classes" is excluded
**classes

# Specify file extensions
**.cls*
**.pdf

# Specify a specific file
helloWorld/main/default/HelloWorld.cls*
```

List the Files and Directories Currently Being Ignored

Use the `force:source:ignored` command to list the files and directories in your project that the `force:source:*` commands are currently ignoring. The `force:source:ignored` command refers to the `.forceignore` file to determine the list of ignored files.

Run the command without any parameters to list all the files in all package directories that are ignored. Use the `--sourcepath` parameter to limit the check to a specific file or directory. If you specify a directory, the command checks all subdirectories recursively.

This example checks if a particular file is ignored.

```
sfdx force:source:ignored:list --sourcepath=package.xml
```

This example gets a list of all ignored files in a specific directory.

```
sfdx force:source:ignored:list --sourcepath=force-app/main/default
```

Sample output if the command finds ignored files:

```
Found the following ignored files:  
force-app/main/default/aura/.eslintrc.json  
force-app/main/default/lwc/.eslintrc.json  
force-app/main/default/lwc/jsconfig.json
```

Sample output if the command doesn't find the specified file:

```
ERROR running force:source:ignored:list: ENOENT: no such file or directory, stat  
'package.xml'
```

Sample output if the file isn't ignored:

```
No ignored files found in paths:  
README.md
```

Create a Salesforce DX Project from Existing Source

If you are already a Salesforce developer or ISV, you likely have existing source in a managed package in your packaging org or some application source in your sandbox or production org. Before you begin using Salesforce DX, retrieve the existing source and convert it to the source format.

-  **Tip:** If your current repo follows the directory structure that is created from a Metadata API retrieve, you can skip to converting the metadata format after you create a Salesforce DX project.
1. Create a Salesforce DX project.
 2. Create a directory for the metadata retrieve. You can create this directory anywhere.

```
mkdir mdapipkg
```

3. Retrieve your metadata source.

Format of Current Source	How to Retrieve Your Source for Conversion
You are a partner who has your source already defined as a managed package in your packaging org.	Retrieve Source from an Existing Managed Package
You have a <code>package.xml</code> file that defines your unpackaged source.	Retrieve Unpackaged Source Defined in a package.xml file

SEE ALSO:

[Convert the Metadata Source to Source Format](#)

[Create a Salesforce DX Project](#)

Retrieve Source from an Existing Managed Package

If you're a partner or ISV who already has a managed package in a packaging org, you're in the right place. You can retrieve that package, unzip it to your local project, and then convert it to source format, all from the CLI.

Before you begin, create a Salesforce DX project.

1. Retrieve the metadata from the source org.

```
sfdx force:mdapi:retrieve -s -r ./mdapipkg -u <username> -p <package name>
```

The username can be a username or alias for the source org (such as a packaging org) from which you're pulling metadata. Indicate the directory where you'd like to store the retrieved package metadata using the `-r` parameter. If this directory doesn't exist, the CLI creates it. The `-s` parameter indicates that you're retrieving a single package. If your package name contains a space, enclose the name in double quotes.

```
-p "Test Package"
```

2. Check the status of the retrieve.

When you run `force:mdapi:retrieve`, the job ID, target username, and retrieve directory are stored, so you don't have to specify these required parameters to check the status. These stored values are overwritten when you run the `force:mdapi:retrieve` again.

```
sfdx force:mdapi:retrieve:report
```

If you want to check the status of a different retrieve operation, specify the retrieve directory and job ID on the command line, which overrides any stored values.

3. Unzip the zip file.

4. (Optional) Delete the zip file.

After you finish, convert the metadata to source format.

SEE ALSO:

[Create a Salesforce DX Project](#)

[Convert the Metadata Source to Source Format](#)

Retrieve Unpackaged Source Defined in a package.xml File

If you already have a `package.xml` file, you can retrieve it, unzip it in your local project, and convert it to source format. You can do all these tasks from the CLI. The `package.xml` file defines the source you want to retrieve.

But what if you don't have a `package.xml` file already created? See [Sample package.xml Manifest Files](#) in the *Metadata API Developer Guide*.



Note: If you already have the source in metadata format, you can skip these steps and go directly to converting it to source format.

1. In the project, create a folder to store what's retrieved from your org, for example, `mdapipkg`.
2. Retrieve the metadata.

```
sfdx force:mdapi:retrieve -r ./mdapipkg -u <username> -k ./package.xml
```

The username can be the scratch org username or an alias. The `-k` parameter indicates the path to the `package.xml` file, which is the unpackaged manifest of components to retrieve.

3. Check the status of the retrieve.

When you run `force:mdapi:retrieve`, the job ID, target username, and retrieve directory are stored, so you don't have to specify these required parameters to check the status. These stored values are overwritten when you run the `force:mdapi:retrieve` again.

```
sfdx force:mdapi:retrieve:report
```

If you want to check the status of a different retrieve operation, specify the retrieve directory and job ID on the command line, which overrides any stored values.

4. Unzip the zip file.

5. (Optional) Delete the zip file.

After you retrieve the source and unzip it, you no longer need the zip file, so you can delete it.

After you finish, convert from metadata format to source format.

SEE ALSO:

[Convert the Metadata Source to Source Format](#)

Convert the Metadata Source to Source Format

After you retrieve the source from your org, you can complete the configuration of your project and convert the metadata source to source format.

The `convert` command ignores all files that start with a "dot," such as `.DS_Store`. To exclude more files from the convert process, add a `.forceignore` file.

1. Convert metadata format to source format. Let's say you created a directory called `mdapi_project` when you retrieved the metadata.

```
sfdx force:mdapi:convert --rootdir mdapi_project --outputdir tmp_convert
```

The `--rootdir` parameter is the name of the directory that contains the metadata source.

If you don't indicate an output directory with the `--outputdir` parameter, the converted source is stored in the default package directory indicated in the `sfdx-project.json` file. If the output directory is located outside of the project, you can indicate its location using an absolute path.

2. To indicate which package directory is the default, update the `sfdx-project.json` file.

If there are two or more files with the same file name yet they contain different contents, the output directory contains duplicate files. Duplicate files can occur if you convert the same set of metadata more than once. The `mdapi:convert` process identifies these files with a `.dup` file extension. The `source:push` and `source:pull` commands ignore duplicate files, so you'll want to resolve them. You have these options:

- Choose which file to keep, then delete the duplicate.
- Merge the files, then delete the other.

Next steps:

- Authorize the Dev Hub org and set it as the default
- Configure the Salesforce DX project

- Create a scratch org

SEE ALSO:

[How to Exclude Source When Syncing or Converting](#)

[Salesforce DX Project Configuration](#)

[Authorization](#)

[Create Scratch Orgs](#)

Salesforce DX Usernames and Orgs

Many CLI commands connect to an org to complete their task. For example, the `force:org:create` command, which creates a scratch org, connects to a Dev Hub org. The `force:source:push|pull` commands synchronize source code between your project and a scratch org. In each case, the CLI command requires a username to determine which org to connect to. Usernames are unique within the entire Salesforce ecosystem and have a one-to-one association with a specific org.



Note: The examples in this topic might refer to CLI commands that you aren't yet familiar with. For now, focus on how to specify the usernames, configure default usernames, and use aliases. The CLI commands are described later.

When you create a scratch org, the CLI generates a username. The username looks like an email address, such as `test-wvkpnfm5z113@example.com`. You don't need a password to connect to or open a scratch org, although you can generate one later with the `force:user:password:generate` command.

Salesforce recommends that you set a default username for the orgs that you connect to the most during development. The easiest way to do this is when you authorize a Dev Hub org or create a scratch org. Specify the `--setdefaultdevhubusername` or `--setdefaultusername` parameter, respectively, from within a project directory. You can also create an alias to give the usernames more readable names. You can use usernames or their aliases interchangeably for all CLI commands that connect to an org.

These examples set the default usernames and aliases when you authorize an org and then when you create a scratch org.

```
sfdx auth:web:login --setdefaultdevhubusername --setalias my-hub-org  
sfdx force:org:create --definitionfile my-org-def.json --setdefaultusername --setalias my-scratch-org
```

To verify whether a CLI command requires an org connection, look at its parameter list with the `--help` parameter. Commands that have the `--targetdevhubusername` parameter connect to the Dev Hub org. Similarly, commands that have `--targetusername` connect to scratch orgs, sandboxes, and so on. This example displays the parameter list and help information about `force:org:create`.

```
sfdx force:org:create --help
```

When you run a CLI command that requires an org connection and you don't specify a username, the command uses the default. To see your default usernames, run `force:org:list` to display all the orgs you've authorized or created. The default Dev Hub and scratch orgs are marked on the left with (D) and (U), respectively.

Let's run through a few examples to see how this works. This example pushes source code to the scratch org that you've set as the default.

```
sfdx force:source:push
```

To specify an org other than the default, use `--targetusername`. For example, let's say you created another scratch org with alias `my-other-scratch-org`. It's not the default but you still want to push source to it.

```
sfdx force:source:push --targetusername my-other-scratch-org
```

This example shows how to use the `--targetdevhubusername` parameter to specify a non-default Dev Hub org when creating a scratch org.

```
sfdx force:org:create --targetdevhubusername jdoe@mydevhub.com --definitionfile my-org-def.json --setalias yet-another-scratch-org
```

More About Setting Default Usernames

If you've already created a scratch org, you can set the default username with the `config:set` command from your project directory.

```
sfdx config:set defaultusername=test-wvkpnfm5z113@example.com
```

The command sets the value locally, so it works only for the current project. To use the default username for all projects on your computer, specify the `--global` parameter. You can run this command from any directory. Local project defaults override global defaults.

```
sfdx config:set defaultusername=test-wvkpnfm5z113@example.com --global
```

The process is similar to set a default Dev Hub org, except you use the `defaultdevhubusername` config value.

```
sfdx config:set defaultdevhubusername=jdoe@mydevhub.com
```

To unset a config value, run the `config:unset` command. Use the `--global` parameter to unset it for all your Salesforce DX projects.

```
sfdx config:unset defaultusername --global
```

More About Aliasing

Use the `alias:set` command to set an alias for an org or after you've authorized an org. You can create an alias for any org: Dev Hub, scratch, production, sandbox, and so on. So when you issue a command that requires the org username, using an alias for the org that you can easily remember can speed up things.

```
sfdx alias:set my-scratch-org=test-wvkpnfm5z113@example.com
```

An alias also makes it easy to set a default username. The previous example of using `config:set` to set `defaultusername` now becomes much more digestible when you use an alias rather than the username.

```
sfdx config:set defaultusername=my-scratch-org
```

Set multiple aliases with a single command by separating the name-value pairs with a space.

```
sfdx alias:set org1=<username> org2=<username>
```

You can associate an alias with only one username at a time. If you set it multiple times, the alias points to the most recent username. For example, if you run the following two commands, the alias `my-org` is set to `test-wvkpnfm5z113@example.com`.

```
sfdx alias:set my-org=test-blahdiblah@whoanellie.net
sfdx alias:set my-org=test-wvkpnfm5z113@example.com
```

To view all aliases that you've set, use one of the following commands.

```
sfdx alias:list
sfdx force:org:list
```

To remove an alias, use the `alias:unset` command.

```
sfdx alias:unset my-org
```

List All Your Orgs

Use the `force:org:list` command to display the usernames for the orgs that you've authorized and the active scratch orgs that you've created.

```
sfdx force:org:list
==== Orgs
  ALIAS      USERNAME      ORG ID      CONNECTED STATUS
  -----      -----      -----
  DD-ORG      jdoe@dd-204.com    00D...OEA  Connected
(D) devhuborg  jdoe@mydevhub.com  00D...MAC  Connected

  ALIAS      SCRATCH ORG NAME  USERNAME      ORG ID      EXPIRATION DATE
  -----      -----      -----      -----
  my-scratch  Your Company    test-wvkm5z113@example.com 00D...UAI 2017-06-13
(U) scratch208  Your Company  test-wvkm5z113@example.com 00D...UAY 2017-06-13
```

The top section of the output lists the non-scratch orgs that you've authorized, including Dev Hub orgs, production orgs, and sandboxes. The output displays the usernames that you specified when you authorized the orgs, their aliases, their IDs, and whether the CLI can connect to it. A (D) on the left points to the default Dev Hub org username.

The lower section lists the active scratch orgs that you've created and their usernames, org IDs, and expiration dates. A (U) on the left points to the default scratch org username.

To view more information about scratch orgs, such as the create date, instance URL, and associated Dev Hub org, use the `--verbose` parameter.

```
sfdx force:org:list --verbose
```

Use the `--clean` parameter to remove non-active scratch orgs from the list. The command prompts you before it does anything.

```
sfdx force:org:list --clean
```

SEE ALSO:

[Authorization](#)

[Build Your Own Scratch Org Definition File](#)

[Create Scratch Orgs](#)

[Generate or Change a Password for a Scratch Org User](#)

[Push Source to the Scratch Org](#)

Override or Add Definition File Options at the Command Line

Some CLI commands, such as `force:org:create` and `force:user:create`, use a JSON definition file to determine the characteristics of the org or user they create. The definition file contains one or more options. You can override some options by specifying them as name-value pairs at the command line. You can also specify options that aren't in the definition file. This technique allows multiple users or continuous integration jobs to share a base definition file and then customize options when they run the command.

Let's say you use the following JSON definition file to create a scratch org. By default, the scratch org definition is named `project-scratch-def.json`.

```
{  
    "orgName": "Acme",  
    "country": "US",  
    "edition": "Enterprise",  
    "hasSampleData": "true",  
    "features": ["MultiCurrency", "AuthorApex"],  
    "settings": {  
        "mobileSettings": {  
            "enableS1EncryptedStoragePref2": true  
        },  
        "chatterSettings": {  
            "enableChatter": true  
        },  
        "nameSettings": {  
            "enableNameSuffix": false  
        }  
    }  
}
```

To create an Enterprise Edition scratch org that uses all the options in the file, run this command.

```
sfdx force:org:create --definitionfile project-scratch-def.json
```

You can then use the same definition file to create a Developer Edition scratch org that doesn't have sample data by overriding the `edition` and `hasSampleData` options.

```
sfdx force:org:create --definitionfile project-scratch-def.json edition=Developer  
hasSampleData=false
```

Use commas to separate multiple array values, and enclose them in double quotes. For example, to change the `features` option:

```
sfdx force:org:create --definitionfile project-scratch-def.json  
features="MultiCurrency,PersonAccounts"
```

This example shows how to add the `adminEmail` option, which doesn't exist in the definition file.

```
sfdx force:org:create --definitionfile project-scratch-def.json adminEmail=john@doe.org
```



Note: You can't override options whose values are JSON objects, such as `settings`.

SEE ALSO:

[Create Scratch Orgs](#)

[Create a Scratch Org User](#)

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, go to Step 1.
- 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. Once 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. Log in to the Developer Edition org in which your namespace is registered using the org's System Administrator's credentials.

You cannot 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** view.

SEE ALSO:

[Create a Developer Edition Org](#)

[Lightning Aura Components Developer Guide: Create a Namespace in Your Org](#)

[Add Salesforce DX Users](#)

[Salesforce Help: My Domain](#)

Salesforce DX Project Configuration

The project configuration file `sfdx-project.json` indicates that the directory is a Salesforce DX project. The configuration file contains project information and facilitates the authentication of orgs and the creation of second-generation packages. It also tells Salesforce CLI where to put files when syncing between the project and org.

We provide sample `sfdx-project.json` files in the sample repos for creating a project using Salesforce CLI or Salesforce Extensions for VS Code.



Note: Are you planning to create second-generation packages? When you're ready, add packaging-specific configuration options to support package creation. See https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev2gp_config_file.htm.

We recommend that you check in this file with your source.

```
{  
  "packageDirectories" : [  
    { "path": "force-app", "default": true},  
    { "path" : "unpackaged" },  
    { "path" : "utils" }  
  ],  
  "namespace": "",  
  "sfcdLoginUrl" : "https://login.salesforce.com",
```

```
"sourceApiVersion": "44.0"  
}
```

You can manually edit these parameters.

name (required for Salesforce Functions)

Salesforce DX or Salesforce Functions project name.

namespace (optional)

The global namespace that is used with a package. The namespace must be registered with an org that is associated with your Dev Hub org. This namespace is assigned to scratch orgs created with the `org:create` command. If you're creating an unlocked package, you have the option to create a package with no namespace.

 **Important:** Register the namespace with Salesforce and then connect the org with the registered namespace to the Dev Hub org.

oauthLocalPort (optional)

By default, the OAuth port is 1717. However, change this port if this port is already in use, and you plan to create a connected app in your Dev Hub org to support JWT-based authorization. Also, don't forget to follow the steps in [Create a Connected App in Your Org](#) to change the callback URL.

packageAliases (optional)

Aliases for package IDs, which can often be cryptic. See [Project Configuration File for Packages](#) for details.

packageDirectories (required)

Package directories indicate which directories to target when syncing source to and from the org. These directories can contain source from your managed package, unmanaged package, or unpackaged source, for example, ant tool or change set. For information on all `packageDirectories` options, see [Project Configuration File for Packages](#).

Keep these things in mind when working with package directories.

- The location of the package directory is relative to the project. Don't specify an absolute path. The following two examples are equivalent.

```
"path": "helloWorld"  
"path" : "./helloWorld"
```

- You can have only one default path (package directory). If you have only one path, we assume it's the default, so you don't have to explicitly set the `default` parameter. If you have multiple paths, you must indicate which one is the default.
- Salesforce CLI uses the default package directory as the target directory when pulling changes in the org to sync the local project. This default path is also used when creating second-generation packages.
- If you don't specify an output directory, the default package directory is also where files are stored during source conversions. Source conversions are both from metadata format to source format, and from source format to metadata format.

plugins (optional)

To use the plug-ins you develop using the [Salesforce Plugin Generator](#) with your Salesforce DX project, add a `plugins` section to the `sfdx-project.json` file. In this section, add configuration values and settings to change your plug-ins' behavior.

```
"plugins": {  
  "yourPluginName": {  
    "timeOutValue": "2"  
  },  
  "yourOtherPluginName": {  
    "yourCustomProperty": true
```

```
    }  
}
```

Store configuration values for only those values that you want to check in to source control for the project. These configuration values affect your whole development team.

pushPackageDirectoriesSequentially (optional)

Set to `true` to push multiple package directories in the order they're listed in `packageDirectories` when using `force:source:push`. The directories are pushed in separate transactions. The default value of this property is `false`, which means that multiple package directories are deployed in a single transaction without regard to order. Example:

```
"packageDirectories": [  
  {  
    "path": "es-base-custom",  
    "default": true  
  },  
  {  
    "path": "es-base-ext"  
  }  
,  
  "pushPackageDirectoriesSequentially": true,
```

 **Note:** This property applies only to `force:source:push` and doesn't affect the behavior of the `force:source:deploy` command.

See [Push Source Sequentially](#) for more information.

replacements (optional)

Automatically replace strings in your metadata source files with specific values right before you deploy the files to an org.

See [Replace Strings in Code Before Deploying](#) for details.

sfdcLoginUrl (optional)

The login URL that the `auth` commands use. If not specified, the default is `login.salesforce.com`. Override the default value if you want users to authorize to a specific Salesforce instance. For example, if you want to authorize into a sandbox org, set this parameter to `test.salesforce.com`.

If you don't specify a default login URL here, or if you run `auth` outside the project, you specify the instance URL when authorizing the org.

sourceApiVersion (optional)

The API version that the source is compatible with. The default is the same version as the Salesforce CLI.

The `sourceApiVersion` determines the fields retrieved for each metadata type during `source:push`, `source:pull`, or `source:convert`. This field is important if you're using a metadata type that has changed in a recent release. You'd want to specify the version of your metadata source. For example, let's say a new field was added to the CustomTab for API version 53.0. If you retrieve components for version 52.0 or earlier, you see errors when running the source commands because the components don't include that field.

Be sure not to confuse this project configuration value with the [apiVersion](#) CLI runtime configuration value, which has a similar name.

SEE ALSO:

- [Link a Namespace to a Dev Hub Org](#)
- [Authorization](#)
- [How to Exclude Source When Syncing or Converting](#)
- [Pull Source from the Scratch Org to Your Project](#)
- [Push Source to the Scratch Org](#)

Multiple Package Directories

When you create your Salesforce DX project, we recommend that you organize your metadata into logical groupings by creating multiple package directories locally. You then define these directories in your `sfdx-project.json` file. You can group similar code and source for an application or customization to better organize your team's repository. Later, if you decide to use second-generation packages (2GP), these directories correspond to the actual 2GP packages.



Note: For clarity in this topic, package directory refers to the local (client-side) directory that contains decomposed metadata files, that is, metadata in source format. This directory doesn't always result in a 2GP package. Package refers to a 2GP package.

Also, the terms *push* and *deploy* are interchangeable; both mean moving metadata components from your local project to your org. Similarly, *pull* and *retrieve* are interchangeable. In this topic, we generally use the terms *deploy* and *retrieve*. However, we use the terms *push* and *pull* when talking about the specific actions of the `force:source:push|pull` commands.

In your `sfdx-project.json` file, list each package directory separately in the `packageDirectories` section. Each local package directory adheres to the standard Salesforce DX project structure.

The multiple package directory structure is client-side (local) only. When you deploy the source to the org with `force:source:deploy` or `force:source:push`, there's no association between its local package directory location and the package in the org. You specify that metadata belongs to a specific 2GP package in an org by explicitly installing the 2GP package.

All of the `force:source:*` commands support multiple package directories.

Considerations

Before setting up multiple package directories, read these considerations.

- By default, both `force:source:deploy` and `force:source:push` deploy metadata to your org in a single transaction, regardless of the order that you list your multiple package directories in `sfdx-project.json`. The `force:source:push` command pushes all changed and new metadata in all package directories; `force:source:deploy` deploys only the metadata that you specify. If you want to use `force:source:push`, but also want to specify the order that the package directories are pushed, use the `pushPackageDirectoriesSequentially` property of `sfdx-project.json`. See [Push Source Sequentially](#) for details.
- By default, the `force:source:deploy|retrieve` commands don't track changes in your source files. Enable source tracking for these commands by specifying the `--tracksource` parameter each time you run the command. The `force:source:deploy|retrieve` commands then use the same internal source tracking files as the `force:source:push|pull` commands. Be sure you use this parameter if you use `force:source:deploy|retrieve` and `force:source:push|pull` together. If you don't, the internal source-tracking files get out of sync. The `force:source:push|pull` commands always track your source changes.

How Do I Set It Up?

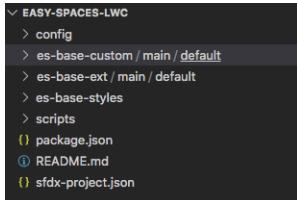
Setting up multiple package directories is easy. How you organize your local source code among these directories takes more thought and planning, and depends on your development environment. Plan how to organize your code before you get started. Keep your source code well organized as your project grows to make it easier and more efficient for your developers to work.

Let's look at a simple example. Say you put the decomposed metadata files for a custom object MyObject in the default package directory. You can then put files for a new field MyField on MyObject in a different "extension" package directory without having to also include the MyObject files. There are many ways to organize your code, and describing all the ways is out of scope of this topic. [These blog posts](#) provide some ideas.

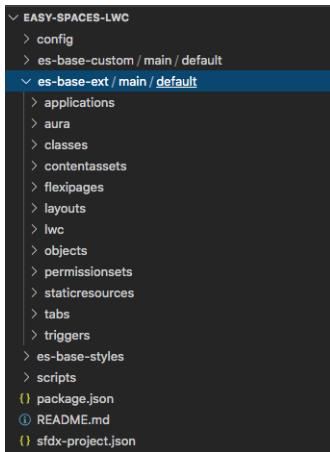
Here's how you set up multiple package directories. Let's first look at a sample `sfdx-project.json` snippet:

```
"packageDirectories": [
  {
    "path": "es-base-custom",
    "default": true
  },
  {
    "path": "es-base-ext"
  },
  {
    "path": "es-base-styles"
  }
],
```

This `sfdx-project.json` snippet defines three package directories: `es-base-custom` (the default), `es-base-ext`, and `es-base-styles`. Let's say your top-level local project directory is called `easy-spaces-lwc`. The directory hierarchy underneath it would look something like this:



Each `es-base-*` directory adheres to the standard Salesforce DX project structure. For example, `es-base-ext` would look something like:



Now add the decomposed metadata source to these multiple package directories in the way that best suits your development environment.

How Does It Work?

Let's go through a few examples to see how `force:source:push|pull` and `force:source:deploy|retrieve` work with multiple package directories.

 **Note:** The examples in this section assume you're using the `force:source:deploy|retrieve` and `force:source:push|pull` commands together. To ensure that source-tracking stays in sync, the `force:source:deploy|retrieve` examples use the `--tracksource` parameter.

For new orgs, the `force:source:push` command pushes all the metadata in all multiple package directories listed in your `sfdx-project.json` file. After that, the command pushes metadata that's new, changed, or marked for delete. By default, the command pushes the metadata in a single transaction, as if you had just one package directory.

```
sfdx force:source:push -u my-org
```

In contrast, use `force:source:deploy` to target the metadata you want to deploy. You can deploy specific package directories, specific metadata components, components listed in a manifest file, and more. This example deploys the metadata in the `es-base-custom` package directory:

```
sfdx force:source:deploy --sourcepath es-base-custom --tracksource -u my-org
```

This example deploys all Apex classes found in all your multiple package directories:

```
sfdx force:source:deploy --metadata ApexClass --tracksource -u my-org
```

When you run `force:source:pull`, the command pulls all remote changes from the org into your local project. For each retrieved component, the command looks in all package directories for a local match. If it finds a match, the command updates it. If it doesn't find a match, the command copies the local component into the default package directory, which in our example is `es-base-custom`.

```
sfdx force:source:pull -u my-org
```

You can then move the pulled files into the package directory that makes sense for your project. After you push the moved files back up to the org with `force:source:push`, Salesforce CLI tracks their new location.

Use `force:source:retrieve` to retrieve targeted metadata from your org. Similar to `force:source:pull`, existing metadata is retrieved into its correct local package directory and new metadata into the default package directory. This example retrieves only the metadata components contained in the local `es-base-custom` package directory:

```
sfdx force:source:retrieve --sourcepath es-base-custom --tracksource -u my-org
```

This example retrieves all Apex classes from your org; new classes go into the default package directory and classes that exist locally go into their corresponding package directory.

```
sfdx force:source:retrieve --metadata ApexClass --tracksource -u my-org
```

Push Source Sequentially

As we've mentioned, `force:source:push`, by default, pushes metadata in all package directories in a single transaction without regard to order. Using a single transaction is much faster for most projects. But sometimes you must specify the exact order that the package directories are pushed. Reasons include:

- The number of recomposed metadata component files in your local project exceeds the Salesforce metadata limit of 10,000 files per retrieve or deploy. One workaround is to split up your metadata into multiple package directories that each contain less than this limit and push each directory sequentially, and thus separately.
- You have dependencies between multiple package directories, which requires that they be pushed in a specific order.

- More than one package directory contains the same metadata component, and you want to specify which one is deployed last so it's not overwritten.

To specify that the multiple package directories are pushed in the order they're listed in `sdfx-project.json`, add `"pushPackageDirectoriesSequentially": true` to the file. For example:

```
"packageDirectories": [
  {
    "path": "es-base-custom",
    "default": true
  },
  {
    "path": "es-base-ext"
  },
  {
    "path": "es-base-styles"
  }
],
"pushPackageDirectoriesSequentially": true,
```

When you run `force:source:push`, the command executes three separate pushes in this order: first the metadata from `es-base-custom`, then `es-base-ext`, and then `es-base-styles`.

! **Important:** The `pushPackageDirectoriesSequentially` property doesn't affect how `force:source:deploy` works. If you need multiple deployments in a specific order when using the `force:source:deploy` command, run the command several times using the `-p` or `-m` parameters in the order you wish.

SEE ALSO:

[Second-Generation Managed Packages](#)

[Salesforce DX Project Structure and Source Format](#)

[Install and Uninstall Second-Generation Managed Packages](#)

[Salesforce Developers Blog: Working with Modular Development and Unlocked Packages](#)

Replace Strings in Code Before Deploying

Automatically replace strings in your metadata source files with specific values right before you deploy the files to an org.

These sample use cases describe scenarios for using pre-deployment string replacement:

- A NamedCredential contains an endpoint that you use for testing. But when you deploy the source to your production org, you want to specify a different endpoint.
- An ExternalDataSource contains a password that you don't want to store in your repository, but you're required to deploy the password along with your metadata.
- You deploy near-identical code to multiple orgs. You want to conditionally swap out some values depending on which org you're deploying to.

String replacement actually occurs when source-formatted files are converted to a ZIP file right before being deployed to the org with the `force:source:deploy` and `force:source:push` commands. The changes that result from string replacement are never written to your project; they apply only to the deployed files.

Configure String Replacement

Configure string replacement by adding a `replacements` property to your `sfdx-project.json` file. The property accepts multiple entries that consist of keys that define the:

- Source file or files that contain the string to be replaced.
- The string to be replaced.
- The replacement value.

Let's look at an example to see how it works. This sample `sfdx-project.json` specifies that when the file `force-app/main/default/classes/myClass.cls` is deployed, all occurrences of the string `replaceMe` are replaced with the value of the `THE_REPLACEMENT` environment variable:

```
{  
  "packageDirectories": [  
    {  
      "path": "force-app",  
      "default": true  
    }  
  ],  
  "name": "myproj",  
  "replacements": [  
    {  
      "filename": "force-app/main/default/classes/myClass.cls",  
      "stringToReplace": "replaceMe",  
      "replaceWithEnv": "THE_REPLACEMENT"  
    }  
  ]  
}
```

You can specify these keys in the `replacements` property.

Location of Files

One of the following properties is required:

- `filename`: Single file that contains the string to be replaced.
- `glob`: Collection of files that contain the string to be replaced. Example: `**/classes/*.cls`.

String to be Replaced

One of the following properties is required:

- `stringToReplace`: The string to be replaced.
- `regexToReplace`: Regular expression (regex) that specifies a string pattern to be replaced.

Replacement Value

One of the following properties is required:

- `replaceWithEnv`: Specifies that the string is replaced with the value of the environment variable.
- `replaceWithFile`: Specifies that the string is replaced with the contents of a file.

Conditional Processing

This property is optional:

- `replaceWhenEnv`: Specifies that a string replacement occur only when a specific environment variable is set to a specific value. Use the property `env` to specify the environment variable and the property `value` to specify the value that triggers the string replacement.

Follow these syntax rules:

- Always use forward slashes (/), even on Windows.
- Both JSON and regular expressions use the backslash (\) as an escape character. As a result, when you use a regular expression to match a dot, which requires escaping, you must use two backslashes for the `regexToReplace` value:

```
"regexToReplace" : "\\".
```

Similarly, to match a single backslash, you must specify three of them.

```
"regexToReplace" : "\\\\"
```

Examples

This example is similar to the previous example but shows how to configure string replacement for two files:

```
"replacements": [
  {
    "filename": "force-app/main/default/classes/FirstApexClass.cls",
    "stringToReplace": "replaceMe",
    "replaceWithEnv": "THE_REPLACEMENT"
  },
  {
    "filename": "force-app/main/default/classes/SecondApexClass.cls",
    "stringToReplace": "replaceMe",
    "replaceWithEnv": "THE_REPLACEMENT"
  }
]
```

This example shows how to specify that the string replacement occur only if an environment variable called DEPLOY_DESTINATION exists and it has a value of PROD.

```
"replacements": [
  {
    "filename": "force-app/main/default/classes/myClass.cls",
    "stringToReplace": "replaceMe",
    "replaceWithEnv": "THE_REPLACEMENT",
    "replaceWhenEnv": [
      {
        "env": "DEPLOY_DESTINATION",
        "value": "PROD"
      }
    ]
}
```

This example specifies that when the Apex class files in the `force-app/main/default` directory are deployed, all occurrences of the string `replaceMe` are replaced with the contents of the file `replacementFiles/copyright.txt`.

```
"replacements": [
  {
    "glob": "force-app/main/default/classes/*.cls",
    "stringToReplace": "replaceMe",
    "replaceWithFile": "replacementFiles/copyright.txt"
  }
]
```

Use a regular expression to specify a search pattern for text rather than the literal text. For example, Apex class XML files always contain an <apiVersion> element that specifies the Salesforce API version, as shown in this snippet.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ApexClass xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>55.0</apiVersion>
    <status>Active</status>
</ApexClass>
```

Let's say you want to test your Apex classes on a more recent API version before you actually update all your classes. This example shows how to use a regular expression to search for the <apiVersion> element. At deploy, the element is then replaced with a specific string, such as <apiVersion>56.0</apiVersion>, which is contained in the replacementFiles/latest-api-version.txt file.

```
"replacements": [
    {
        "glob": "force-app/main/default/classes/*.xml",
        "regexToReplace": "<apiVersion>\d+\.\d+</apiVersion>",
        "replaceWithFile": "replacementFiles/latest-api-version.txt"
    }
]
```

Test String Replacements

Follow these steps to test string replacement without actually deploying files to the org.

1. Set the SF_APPLY_REPLACEMENTS_ON_CONVERT environment variable to true.
2. Run the force:source:convert command, which converts the source files into metadata API format. For example:

```
sfdx force:source:convert --outputdir mdapiOut --sourcedpath force-app
```

3. Inspect the files in the output directory (mdapiOut in our example) for the string replacements and what exactly will be deployed to the org.

 **Warning:** Be careful when writing passwords or secrets to the file system while testing. Also, be sure to reset any environment variables you set during testing so they aren't accidentally applied later.

Tips and Tricks

- (macOS or Linux only) When using the replaceWithEnv or replaceWhenEnv properties, you can specify that the environment variables apply to a single command by prepending the variables before the command execution. For example:

```
THE_REPLACEMENT="some text" DEPLOY_DESTINATION=PROD sfdx force:source:push
```

 **Warning:** Be careful when setting passwords or secrets this way, because they show up in your terminal history.

- If you've configured many string replacements, and are finding it difficult to manage, check out open-source tools that load the contents of one or more files to your environment, such as [dotenv-cli](#). In this example, environment variables configured in two local .env files are loaded before the force:source:push command execution:

```
dotenv -e .env1 -e .env2 sfdx force:source:push
```

 **Warning:** Don't commit passwords or secrets in .env files.

- If you specify `--json` for either `force:source:deploy` or `force:source:push`, the JSON output includes a `replacements` property that lists the affected files and the string that was replaced.

The `force:source:push` human-readable output shows the string replacement information by default; use `--quiet` to remove it, which also removes it from the JSON output.

To view string replacement information in the `force:source:deploy` human-readable output, specify `--verbose`.

Considerations and Limitations

- If you configure multiple string replacements in multiple files, the performance of the deployment can degrade. Consider using the `filename` key when possible, to ensure that you open only one file. If you must use `glob`, try to limit the number of files that are opened by specifying a single directory or metadata type.

For example, `"glob": "force-app/main/default/classes/*.cls"` targets Apex class files in a specific directory, which is better than `"glob": "**/classes/**"`, which searches for all Apex metadata files in all package directories.

- Be careful using string replacement in static resources. When not doing string replacement, Salesforce CLI simply zips up all static resources when it first encounters their directory and deploys them as-is. If you configure string replacement for a large static resource directory, the CLI must inspect a lot more files than usual, which can degrade performance.
- The `force:mdapi:deploy` command doesn't support string replacement.
- If your deployment times out, or you specify the `--wait 0` flag of `force:source:deploy`, and then run `force:source:deploy:report` to see what happened, the deployed files contain string replacements as usual. However, the output of `force:source:deploy:report` doesn't display the same string replacement information as `force:source:deploy --verbose` would have.

CHAPTER 5 Authorization

In this chapter ...

- [Authorize an Org Using the Web Server Flow](#)
- [Authorize an Org Using the JWT Bearer Flow](#)
- [Create a Private Key and Self-Signed Digital Certificate](#)
- [Create a Connected App in Your Org](#)
- [Use the Default Connected App Securely](#)
- [Use an Existing Access Token instead of Authorizing](#)
- [Authorization Information for an Org](#)
- [Log Out of an Org](#)

The Dev Hub org allows you to create, delete, and manage your Salesforce scratch orgs. After you set up your project on your local machine, you authorize with the Dev Hub org before you can create a scratch org.

You can also authorize other existing orgs, such as sandbox or packaging orgs, to provide more flexibility when using CLI commands. For example, after developing and testing an application using scratch orgs, you can deploy the changes to a centralized sandbox. Or, you can export a subset of data from an existing production org and import it into a scratch org for testing purposes.

You authorize an org only once. To switch between orgs during development, specify your username for the org. Use either the `--targetusername` (or `--targetdevhubusername`) CLI command parameter, set a default username, or use an alias.

You have some options when configuring authorization depending on what you're trying to accomplish.

- We provide the OAuth 2.0 web server flow through a global out-of-the-box connected app. This authorization flow implements the [OAuth 2.0 authorization code grant type](#). With this flow, the server hosting the web app must be able to protect the connected app's identity, defined by the client ID and client secret. When you authorize an org from the command line, you enter your credentials and authorize the global connected app through the Salesforce web server authorization flow. See [OAuth 2.0 Web Server Flow for Web App Integration](#).
- For continuous integration or automated environments in which you don't want to manually enter credentials, use the OAuth 2.0 JSON Web Tokens (JWT) bearer flow. This flow is ideal for scenarios where you can't interactively log in to a browser, such as a continuous integration script. However, this flow does require prior approval of the client app. See [OAuth 2.0 JWT Bearer Flow for Server-to-Server Integration](#).

! **Important:** If your Dev Hub org is configured with high assurance (stepped up) authentication, Salesforce prompts the user to verify identity. This verification process means that you can't use the JWT flow and Salesforce CLI for headless authentication.

SEE ALSO:

[Authorize an Org Using the Web Server Flow](#)

[Authorize an Org Using the JWT Bearer Flow](#)

[Salesforce DX Usernames and Orgs](#)

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

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

Authorize an Org Using the Web Server Flow

To authorize an org with the OAuth 2.0 web server flow, all you do is run a CLI command. Enter your credentials in a browser, and you're up and running!

Authorization requires a connected app. We provide a connected app that is used by default. If you need more security or control, such as setting the refresh token timeout or specifying IP ranges, you can create your own connected app.

1. [\(Optional\) Create a connected app if you require more security and control than offered by the provided connected app.](#) Enable OAuth settings for the new connected app. Make note of the consumer key because you need it later.
2. Update your project configuration file (`sfdx-project.json`). Set the `sfdcLoginUrl` parameter to your My Domain login URL. You can find your org's My Domain login URL on the My Domain Setup page. For example:

```
"sfdcLoginUrl" : "https://MyDomainName.my.salesforce.com"
```

```
"sfdcLoginUrl" : "https://MyDomainName-SandboxName.sandbox.my.salesforce.com"
```

Alternatively, to specify the URL, use the `--instanceurl` parameter of the `auth:web:login` command, as shown in the next step.

 **Note:** We recommend that you use your My Domain login URL, as it isn't affected by org migrations that change your org's Salesforce instance.

3. Run the `auth:web:login` CLI command. If you are authorizing a Dev Hub org, use the `--setdefaultdevhubusername` parameter if you want the Dev Hub org to be the default for commands that accept the `--targetdevhubusername` parameter.

```
sfdx auth:web:login --setdefaultdevhubusername --setalias my-hub-org  
sfdx auth:web:login --setalias my-sandbox
```

If you are using your own connected app, use the `--clientid` parameter. For example, if your client identifier (also called the consumer key) is 04580y4051234051 and you are authorizing a Dev Hub org:

```
sfdx auth:web:login --clientid 04580y4051234051 --setdefaultdevhubusername --setalias my-hub-org
```

To specify a login URL other than the default, such as `https://test.salesforce.com`, use the `--instanceurl` parameter:

```
sfdx auth:web:login --setalias my-hub-org --instanceurl https://test.salesforce.com
```

 **Important:** Use the `--setdefaultdevhubusername` parameter only when authorizing a Dev Hub org. Do not use it when authorizing access to other orgs, such as a sandbox.

4. In the browser window that opens, sign in to your org with your credentials.
5. Close the browser window, unless you want to explore the org.

SEE ALSO:

[Create a Connected App in Your Org](#)

[Salesforce DX Project Configuration](#)

[Create, Clone, or Delete a Sandbox](#)

Authorize an Org Using the JWT Bearer Flow

Continuous integration (CI) environments are fully automated and don't support the human interactivity of the OAuth 2.0 web server authorization flow. In these environments, you must use the JSON web tokens (JWT) bearer flow to authorize an org.

The JWT bearer authorization flow requires a digital certificate, also called a digital signature, to sign the JWT request. You can use your own certificate or create a self-signed certificate using OpenSSL. With this flow, explicit user interaction isn't required. However, this flow does require prior approval of the client app. See [OAuth 2.0 JWT Bearer Flow for Server-to-Server Integration](#).

- !** **Important:** If your Dev Hub org is configured with high assurance (stepped up) authentication, Salesforce prompts the user to verify identity. This verification process means that you can't use the JWT bearer flow and Salesforce CLI for headless authentication.

1. If you do not have your own private key and digital certificate, use [OpenSSL to create the key and a self-signed certificate](#). It is assumed in this task that your private key file is named `server.key` and your digital certificate is named `server.crt`.
2. [Create a connected app, and configure it for Salesforce DX](#).

This task includes uploading the `server.crt` digital certificate file. Make note of the consumer key when you save the connected app because you need it later.

3. If the org you are authorizing is not hosted on `https://login.salesforce.com`, update your project configuration file (`sfdx-project.json`).

Set the `sfdcLoginUrl` parameter to the login URL. Examples of other login URLs are your custom subdomain or `https://test.salesforce.com` for sandboxes. For example:

```
"sfdcLoginUrl": "https://test.salesforce.com"
```

- !** **Important:** If you specify a My Domain subdomain for the login URL, use the version that ends in `my.salesforce.com` instead of the URL you see in Lightning Experience (`.lightning.force.com`). To verify the valid My Domain URL, from Setup, enter *My Domain* in the Quick Find box, then select **My Domain**.

Alternatively, you can use the `--instanceurl` parameter of the `auth:jwt:grant` command to specify the URL. This value overrides the login URL you specified in the `sfdx-project.json` file. See the next step for an example.

4. Run the `auth:jwt:grant` CLI command.

Specify the client identifier from your connected app (also called the consumer key), the path to the private key file (`server.key`), and the JWT authentication username. When you authorize a Dev Hub org, set it as the default with the `--setdefaultdevhubusername` parameter. For example:

```
sfdx auth:jwt:grant --clientid 04580y4051234051 \
--jwtkeyfile /Users/jdoe/JWT/server.key --username jdoe@acdxgs0hub.org \
--setdefaultdevhubusername --setalias my-hub-org
```

This example shows how to use the `--instanceurl` parameter to specify an org hosted on `https://test.salesforce.com` rather than the default `https://login.salesforce.com`:

```
sfdx auth:jwt:grant --clientid 04580y4051234051 \
--jwtkeyfile /Users/jdoe/JWT/server.key --username jdoe@acdxgs0hub.org \
--instanceurl https://test.salesforce.com
```

You can authorize a scratch org using the same client identifier (consumer key) and private key file that you used to authorize its associated Dev Hub org. Set the `--instanceurl` parameter to `https://test.salesforce.com` and the `--username` parameter to the administrator user displayed after you create the scratch org.

[Authorize a Scratch Org](#)

If you configured your Dev Hub to support the OAuth 2.0 JWT bearer authorization flow, you can use the same digital certificate and private key to authorize an associated scratch org. This method is useful for continuous integration (CI) systems that must authorize scratch orgs after creating them, but don't have access to the scratch org's access token.

SEE ALSO:

[Create a Private Key and Self-Signed Digital Certificate](#)[Create a Connected App in Your Org](#)[Salesforce DX Project Configuration](#)[Salesforce Help: Set Up Multi-Factor Authentication](#)[Create, Clone, or Delete a Sandbox](#)

Authorize a Scratch Org

If you configured your Dev Hub to support the OAuth 2.0 JWT bearer authorization flow, you can use the same digital certificate and private key to authorize an associated scratch org. This method is useful for continuous integration (CI) systems that must authorize scratch orgs after creating them, but don't have access to the scratch org's access token.

It is assumed in this task that:

- You previously authorized your Dev Hub org using the JWT bearer flow.
 - The private key file you used when authorizing your Dev Hub org is accessible and located in `/Users/jdoe/JWT/server.key`.
 - You've created a scratch org and have its administration user's username, such as `test-wvkpnfm5z113@example.com`.
1. Copy the consumer key from the connected app that you created in your Dev Hub org.
 - a. Log in to your Dev Hub org.
 - b. From Setup, enter `App Manager` in the Quick Find box to get to the Lightning Experience App Manager.
 - c. Locate the connected app in the apps list, then click  and select **View**.
 - d. In the API (Enable OAuth Settings) section, click **Manage Consumer Details**, then verify your identity.
 - e. Copy the Consumer Key to your clipboard. The consumer key is a long string of numbers, letters, and characters, such as `3MVG9szVa2Rx_sqBb444p50Yj` (example shortened for clarity.)
 2. Run the `sfdx auth:jwt:grant` CLI command. The `--clientid` and `--jwtkeyfile` parameter values are the same as when you ran the command to authorize a Dev Hub org. Set `--username` to the scratch org's admin username and set `--instanceurl` to `https://test.salesforce.com`. For example:

```
sfdx auth:jwt:grant --clientid 3MVG9szVa2Rx_sqBb444p50Yj \
--jwtkeyfile /Users/jdoe/JWT/server.key --username test-wvkpnfm5z113@example.com \
--instanceurl https://test.salesforce.com
```

If you get an error that the user is not approved, it means that the scratch org information has not yet been replicated to `https://test.salesforce.com`. Wait a short time and try again.

SEE ALSO:

[Authorize an Org Using the JWT Bearer Flow](#)[Connected Apps](#)[Create Scratch Orgs](#)

Create a Private Key and Self-Signed Digital Certificate

The OAuth 2.0 JWT bearer authorization flow requires a digital certificate and the private key used to sign the certificate. You upload the digital certificate to the custom connected app that is also required for the JWT bearer authorization flow. You can use your own private key and certificate issued by a certification authority. Alternatively, you can use OpenSSL to create a key and a self-signed digital certificate.

This process produces two files:

- `server.key`—The private key. You specify this file when you authorize an org with the `auth:jwt:grant` command.
- `server.crt`—The digital certification. You upload this file when you create the connected app required by the JWT bearer flow.

1. If necessary, install OpenSSL on your computer.

To check whether OpenSSL is installed on your computer, run this command.

```
which openssl
```

2. In Terminal or a Windows command prompt, create a directory to store the generated files, and change to the directory.

```
mkdir /Users/jdoe/JWT
```

```
cd /Users/jdoe/JWT
```

3. Generate a private key, and store it in a file called `server.key`.

```
openssl genpkey -des3 -algorithm RSA -pass pass:SomePassword -out server.pass.key -pkeyopt rsa_keygen_bits:2048
```

```
openssl rsa -passin pass:SomePassword -in server.pass.key -out server.key
```

You can delete the `server.pass.key` file because you no longer need it.

4. Generate a certificate signing request using the `server.key` file. Store the certificate signing request in a file called `server.csr`. Enter information about your company when prompted.

```
openssl req -new -key server.key -out server.csr
```

5. Generate a self-signed digital certificate from the `server.key` and `server.csr` files. Store the certificate in a file called `server.crt`.

```
openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt
```

SEE ALSO:

[OpenSSL: Cryptography and SSL/TLS Tools](#)

[Create a Connected App in Your Org](#)

[Authorize an Org Using the JWT Bearer Flow](#)

Create a Connected App in Your Org

Salesforce CLI requires a connected app in the org that you're authorizing. A connected app is a framework that enables an external application, in this case Salesforce CLI, to integrate with Salesforce using APIs and standard protocols, such as OAuth. We provide a default connected app when you use the OAuth 2.0 web server flow to authorize an org. For extra security, create your own connected app in your org and configure it with the settings of your choice. You're required to create a connected app when using the OAuth 2.0 JWT bearer authorization flow.

Create a connected app using Setup in your org.

The JWT bearer authorization flow requires a digital certificate, also called a digital signature, to sign the JWT request. You can use your own certificate or create a self-signed certificate using OpenSSL. With this flow, explicit user interaction isn't required. However, this flow does require prior approval of the client app. See [OAuth 2.0 JWT Bearer Flow for Server-to-Server Integration](#).

The web server authorization flow implements the [OAuth 2.0 authorization code grant type](#). With this flow, the server hosting the web app must be able to protect the connected app's identity, defined by the client ID and client secret. See [OAuth 2.0 Web Server Flow for Web App Integration](#).



Note: The steps marked *JWT only* are required only if you're creating a connected app for the JWT bearer authorization flow. They're optional for the web server authorization flow.

To learn more about connected apps, see [Connected Apps](#).

1. Log in to your org.
2. From Setup, enter *App Manager* in the Quick Find box, then select **App Manager**.
3. In the top-right corner, click **New Connected App**.
4. Update the [basic information](#) as needed, such as the connected app name and your email address.
5. Select **Enable OAuth Settings**.
6. For the callback URL, enter `http://localhost:1717/OauthRedirect`.

If port 1717 (the default) is already in use on your local machine, specify an available one instead. Make sure to also update your `sfdx-project.json` file by setting the `oauthLocalPort` property to the new port. For example, if you set the callback URL to `http://localhost:1919/OauthRedirect`:

```
"oauthLocalPort" : "1919"
```

7. (JWT only) Select **Use digital signatures**.
 8. (JWT only) Click **Choose File** and upload the `server.crt` file that contains your digital certificate.
 9. Add these OAuth scopes:
 - **Manage user data via APIs (api)**
 - **Manage user data via Web browsers (web)**
 - **Perform requests at any time (refresh_token, offline_access)**
 10. Click **Save**.
- Important:** Make note of the consumer key because you need it later when you run a `auth` command.
11. Click **Manage**.
 12. Click **Edit Policies**.
 13. In the OAuth Policies section, for the Refresh Token Policy field, click **Expire refresh token after:** and enter 90 days or less.

Setting a maximum of 90 days for the refresh token expiration is a security best practice. To continue running CLI commands against an org whose refresh tokens have expired, reauthorize it with the `auth:web:login` or `auth:jwt:grant` command.

14. In the Session Policies section, set **Timeout Value** to *15 minutes*.

Setting a timeout for access tokens is a security best practice. Salesforce CLI automatically handles an expired access token by referring to the refresh token.

15. (JWT only) In the OAuth Policies section, select **Admin approved users are pre-authorized** for permitted users, and click **OK**.

16. (JWT only) Click **Save**.

17. (JWT only) Click **Manage Profiles** and then click **Manage Permission Sets**. Select the profiles and permission sets that are pre-authorized to use this connected app. Create permission sets if necessary.

SEE ALSO:

[Create a Private Key and Self-Signed Digital Certificate](#)

[Connected Apps](#)

[Authorization](#)

[Salesforce Help: Set Up Multi-Factor Authentication](#)

Use the Default Connected App Securely

If you authorize an org with the `auth:web:login` command, but don't specify the `--clientid` parameter, Salesforce CLI creates a default connected app in the org called `Salesforce CLI`. However, its refresh tokens are set to never expire. As a security best practice, Salesforce recommends that refresh tokens in your org expire after 90 days or less. Another security best practice is to set an expiration for the access token to 15 minutes. Similar to refresh tokens, the access token in the default connected app is set to never expire. To continue using this default connected app in a secure way, configure its policies.

1. Log in to your org.
2. From Setup, enter `OAuth` in the Quick Find box, then select **Connected Apps OAuth Usage**.
3. Select the `Salesforce CLI` app and click **Install**. Confirm by clicking **Install** again.
4. Click **Edit Policies**.
5. In the OAuth Policies section, for the Refresh Token Policy field, click **Expire refresh token after:** and enter *90 Days* or less.
6. In the Session Policies section, set **Timeout Value** to *15 minutes*.
7. Click **Save**.

If you run a CLI command against an org whose refresh token has expired, you get an error. For example:

```
ERROR running force:org:open: Error authenticating with the refresh token due to: expired  
access/refresh token
```

The `force:org:list` command also displays expired refresh token information in the CONNECTED STATUS column. To continue using the org, reauthorize it with the `auth:web:login` or `auth:jwt:grant` command.

Salesforce CLI automatically handles an expired access token by referring to the refresh token.

Use an Existing Access Token instead of Authorizing

When you authorize into an org using the `auth` commands, Salesforce CLI takes care of generating and refreshing all tokens, such as the access token. But sometimes you want to run a few CLI commands against an existing org without going through the entire authorization process. In this case, you provide the access token and URL of the Salesforce instance that hosts the org to which you want to connect.

Almost all CLI commands that have the `--targetusername` | `-u` parameter accept an access token. The only exception is `force:user:display`.

1. To get the instance URL and access token for the org to connect to, run the `force:org:display` command. See [Authorization Information for an Org](#).

2. Use `config:set` to set the `instanceUrl` config value.

```
sfdx config:set instanceUrl=https://na35.salesforce.com
```

3. When you run the CLI command, use the org's access token as the value for the `--targetusername` parameter rather than the org's username.

```
sfdx force:source:deploy --sourcepath <source-dir> --targetusername <access-token>
```

The CLI doesn't store the access token in its internal files. It uses it only for this CLI command run.

Authorization Information for an Org

You can view information for all orgs that you have authorized and the scratch orgs that you have created.

Use this command to view authentication information about an org.

```
sfdx force:org:display --targetusername <username>
```

If you have set a default username, you don't have to specify the `--targetusername` parameter. To display the usernames for all the active orgs that you've authorized or created, use `force:org:list`.

If you have set an alias for an org, you can specify it with the `--targetusername` parameter. This example uses the `my-scratch` alias.

```
sfdx force:org:display --targetusername my-scratch-org
```

==== Org Description	
KEY	VALUE
Access Token	<long-string>
Alias	my-scratch-org
Client Id	SalesforceDevelopmentExperience
Created By	joe@mydevhub.org
Created Date	2017-06-07T00:51:59.000+0000
Dev Hub Id	jdoe@fabdevhub.org
Edition	Developer
Expiration Date	2017-06-14
Id	00D9A0000008cKm
Instance Url	https://page-power-5849-dev-ed.my.salesforce.com
Org Name	Your Company

Status	Active
Username	test-apraqvkwhcml@example.com

To get more information, such as the Salesforce DX authentication URL, include the `--verbose` parameter. However, `--verbose` displays the Sfdx Auth Url only if you authenticated to the org using `auth:web:login` and not `auth:jwt:grant`.

```
sfdx force:org:display -u my-scratch-org --verbose
```

==== Org Description

KEY	VALUE
-----	-------

Access Token	<long-string>
Alias	my-scratch-org
Client Id	SalesforceDevelopmentExperience
Created By	joe@mydevhub.org
Created Date	2017-06-07T00:51:59.000+0000
Dev Hub Id	jdoe@fabdevhub.org
Edition	Developer
Expiration Date	2017-06-14
Id	00D9A0000008cKm
Instance Url	https://page-power-5849-dev-ed.my.salesforce.com
Org Name	Your Company
Sfdx Auth Url	<code>force://SalesforceDevelopmentExperience:xxx@xxx.my.salesforce.com</code>
Status	Active
Username	test-apraqvkwhcml@example.com



Note: To help prevent security breaches, the `force:org:display` output doesn't include the org's client secret or refresh token. If you need these values, perform an OAuth flow outside of the Salesforce CLI.

SEE ALSO:

[OAuth 2.0 Web Server Authentication Flow](#)

[Salesforce DX Usernames and Orgs](#)

Log Out of an Org

For security purposes, you can use the Salesforce CLI to log out of any org you've previously authorized. This practice prevents other users from accessing your orgs if you don't want them to.



Important: The only way to access an org after you log out of it is with a password. By default, new scratch orgs contain one administrator with no password. Therefore, to access a scratch org again after you log out of it, set a password for at least one user. Otherwise, you lose all access to the scratch org. If you don't want to access the scratch org again, rather than log out of it, we recommend that you delete it with `force:org:delete`.

To log out of an org, use `auth:logout`. This example uses the alias `my-hub-org` to log out.

```
sfdx auth:logout --targetusername my-hub-org
```

To log out of all your orgs, including scratch orgs, use the `--all` parameter.

```
sfdx auth:logout --all
```

To access an org again, other than a scratch org, reauthorize it.

When you log out of an org, it no longer shows up in the `force:org:list` output. If you log out of a Dev Hub org, the associated scratch orgs show up only if you specify the `--all` parameter.

CHAPTER 6 Metadata Coverage

Launch the Metadata Coverage report to determine supported metadata for scratch org source tracking purposes. The Metadata Coverage report is the ultimate source of truth for metadata coverage across several channels. These channels include Metadata API, scratch org source tracking, unlocked packages, second-generation managed packages, classic managed packages, and more.

View the [Metadata Coverage report](#).

For more information, see [Metadata Types](#) in the *Metadata API Developer Guide*.

We've moved the information on [Hard-Deleted Components in Unlocked Packages](#).

SEE ALSO:

[Components Available in Managed Packages](#)

CHAPTER 7 Scratch Orgs

In this chapter ...

- [Supported Scratch Org Editions and Allocations](#)
- [Build Your Own Scratch Org Definition File](#)
- [Create a Scratch Org Based on an Org Shape](#)
- [Create Scratch Orgs](#)
- [Select the Salesforce Release for a Scratch Org](#)
- [Push Source to the Scratch Org](#)
- [Pull Source from the Scratch Org to Your Project](#)
- [Scratch Org Users](#)
- [Manage Scratch Orgs from Dev Hub](#)
- [Scratch Org Error Codes](#)

The scratch org is a source-driven and disposable deployment of Salesforce code and metadata. A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with different features and preferences. You can share the scratch org configuration file with other team members, so you all have the same basic org in which to do your development. In addition to code and metadata, developers can install packages and deploy synthetic or dummy data for testing. Scratch orgs should never contain personal data.

Scratch orgs drive developer productivity and collaboration during the development process, and facilitate automated testing and continuous integration. You can use the CLI or IDE to open your scratch org in a browser without logging in. Spin up a new scratch org when you want to:

- Start a new project.
- Start a new feature branch.
- Test a new feature.
- Start automated testing.
- Perform development tasks directly in an org.
- Start from “scratch” with a fresh new org.

Scratch Org Creation Methods

By default, scratch orgs are empty. They don’t contain much of the sample metadata that you get when you sign up for an org, such as a Developer Edition org, the traditional way. Some of the things not included in a scratch org are:

- Custom objects, fields, indexes, tabs, and entity definitions
- Sample data
- Sample Chatter feeds
- Dashboards and reports
- Workflows
- Picklists
- Profiles and permission sets
- Apex classes, triggers, and pages

Before creating a scratch org, you must configure it so it has the features, settings, licenses, and limits that mirror a source org, often your production org. The combination of features, settings, edition, licenses, and limits are what we refer to as the org’s shape.

We offer these methods for configuring scratch orgs:

- [Build Your Own Scratch Org Definition File](#)

- [Create a Scratch Org Based on an Org Shape \(Beta\)](#)

On Which Salesforce Instances Are Scratch Orgs Created?

Scratch orgs are created on sandbox instances. The sandbox instance depends on the country information used when creating the Dev Hub org.

Scratch orgs for Government Cloud and Public Cloud are created in the region where the Dev Hub org is physically located.

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

If you notice that your scratch orgs aren't located in the expected region, create a Salesforce Support case.

Scratch Org Expiration Policy

A scratch org is temporary and is deleted along with the associated ActiveScratchOrgs records from the Dev Hub after their expiration. This expiration process ensures that teams frequently sync their changes with their version control system and are working with the most recent version of their project.

Scratch orgs have a maximum 30 days lifespan. You can select a duration from 1 through 30 days at the time of creation, with the default set at 7 days. After the scratch org has expired, you can't restore it.

 **Note:** Deleting a scratch org doesn't terminate your scratch org subscription. If your subscription is still active, you can create a new scratch org. Creating a new scratch org counts against your daily and active scratch org limits.

Supported Scratch Org Editions and Allocations

Your Dev Hub org is often your production org, and you can enable Dev Hub in these editions: Developer, Enterprise, Unlimited, or Performance. Your Dev Hub edition determines how many scratch orgs you can create. You choose one of the supported scratch org editions each time you create a scratch org.

Supported Scratch Org Editions

The Salesforce edition of the scratch org. Possible values are:

- Developer
- Enterprise
- Group
- Professional



Note: Partners can create partner edition scratch orgs: Partner Developer, Partner Enterprise, Partner Group, and Partner Professional.

This feature is available only if creating scratch orgs from a Dev Hub in a partner business org. See [Supported Scratch Org Editions for Partners](#) in the *ISVforce Guide* for details.

Scratch orgs have these storage limits:

- 200 MB for data
- 50 MB for files

Supported Dev Hub Editions and Associated Scratch Org Allocations

To ensure optimal performance, your Dev Hub org edition determines your scratch org allocations. These allocations determine how many scratch orgs you can create daily, and how many can be active at a given point.

To try out scratch orgs, sign up for a [Developer Edition org](#) on Salesforce Developers, then [enable Dev Hub](#) on page 7.



Note: If you are a partner or ISV, your scratch org allocations are likely different. See the *ISVforce Guide* for details.

Edition	Active Scratch Org Allocation	Daily Scratch Org Allocation
Developer Edition or trial	3	6
Enterprise Edition	40	80
Unlimited Edition	100	200
Performance Edition	100	200

Active Scratch Org Allocation

The maximum number of scratch orgs you can have at any given time based on the edition type. Allocation becomes available if you delete a scratch org or if a scratch org expires.

Daily Scratch Org Allocation

The maximum number of successful scratch org creations you can initiate in a rolling (sliding) 24-hour window. Allocations are determined based on the number of scratch orgs created in the preceding 24 hours.

List Active and Daily Scratch Orgs

-  **Note:** If your Salesforce admin provided access to the Dev Hub org using the Free Limited Access license, you don't have permission to run this command. Contact your admin to provide this information.

To view how many scratch orgs you have allocated, and how many you have remaining:

```
sfdx force:limits:api:display -u <Dev Hub username or alias>
```

NAME	REMAINING	MAXIMUM
ActiveScratchOrgs	25	40
ConcurrentAsyncGetReportInstances	200	200
ConcurrentSyncReportRuns	20	20
DailyApiRequests	14994	100000
DailyAsyncApexExecutions	250000	250000
DailyBulkApiRequests	10000	10000
DailyDurableGenericStreamingApiEvents	10000	10000
DailyDurableStreamingApiEvents	10000	10000
DailyGenericStreamingApiEvents	10000	10000
DailyScratchOrgs	80	80
DailyStreamingApiEvents	10000	10000
DailyWorkflowEmails	75	75
DataStorageMB	1073	1073
DurableStreamingApiConcurrentClients	20	20
FileStorageMB	1073	1073
HourlyAsyncReportRuns	1200	1200
HourlyDashboardRefreshes	200	200
HourlyDashboardResults	5000	5000
HourlyDashboardStatuses	999999999	999999999
HourlyODataCallout	10000	10000
HourlySyncReportRuns	500	500
HourlyTimeBasedWorkflow	50	50
MassEmail	10	10
PermissionSets	1489	1500
SingleEmail	15	15
StreamingApiConcurrentClients	20	20

Build Your Own Scratch Org Definition File

The scratch org definition file is a blueprint for a scratch org. It mimics the shape of an org that you use in the development life cycle, such as sandbox, packaging, or production.

The settings and configuration options associated with a scratch org determine its shape, including:

- Edition—The Salesforce edition of the scratch org, such as Developer, Enterprise, Group, or Professional.
- Add-on features—Functionality that is not included by default in an edition, such as multi-currency.
- Settings—Org and feature settings used to configure Salesforce products, such as Chatter and Communities.

Setting up different scratch org definition files allows you to easily create scratch orgs with different shapes for testing. For example, you can turn Chatter on or off in a scratch org by setting the ChatterEnabled org preference in the definition file. If you want a scratch org with sample data and metadata like you're used to, add this option: `hasSampleData`.

We recommend that you keep this file in your project and check it in to your version control system. For example, create a team version that you check in for all team members to use. Individual developers could also create their own local version that includes the scratch org definition parameters. Examples of these parameters include email and last name, which identify who is creating the scratch org.

Scratch Org Definition File Name

You indicate the path to the scratch org configuration file when you create a scratch org with the `force:org:create` CLI command.

- If you’re using Salesforce CLI on the command line, you can name this file whatever you like and locate it anywhere the CLI can access.
- If you’re using Salesforce Extensions for VS Code, make sure that the scratch org definition file is located in the `config` folder and its name ends in `scratch-def.json`.

If you’re using a sample repo or creating a Salesforce DX project, the sample scratch org definition files are located in the `config` directory. You can create different configuration files for different org shapes or testing scenarios. For easy identification, name the file something descriptive, such as `devEdition-scratch-def.json` or `packaging-org-scratch-def.json`.

Scratch Org Definition File Options

Here are the options you can specify in the scratch org definition file:

Name	Required?	Default If Not Specified
orgName	No	Company
country	No	Dev Hub's country. If you want to override this value, enter the two-character, upper-case ISO-3166 country code (Alpha-2 code). You can find a full list of these codes at several sites, such as: https://www.iso.org/obp/ui/#search . This value sets the locale of the scratch org.
username	No	<code>test-unique_identifier@example.com</code>
adminEmail	No	Email address of the Dev Hub user making the scratch org creation request
edition	Yes	None. Valid entries are Developer, Enterprise, Group, or Professional
description	No	None. 2000-character free-form text field. The description is a good way to document the scratch org’s purpose. You can view or edit the description in the Dev Hub. From App Launcher, select Scratch Org Info or Active Scratch Orgs , then click the scratch org number.
hasSampleData	No	Valid values are <code>true</code> and <code>false</code> . False is the default, which creates an org without sample data.
language	No	Default language for the country. To override the language set by the Dev Hub locale, see Supported Languages for the codes to use in this field.
features	No	None

Name	Required?	Default If Not Specified
release	No	Same Salesforce release as the Dev Hub org. Options are <code>preview</code> or <code>previous</code> . Can use only during Salesforce release transition periods.
settings	No	None
objectSettings	No	<p>None. Use objectSettings to specify object-level sharing settings and default record types. To successfully install in a scratch org, some packages require that you define object-level sharing settings and default record types. The objectSettings option is a map. Each key is the lowercase name of an object, such as opportunity or account. The definition for each key is also a map with two possible values:</p> <ul style="list-style-type: none"> • <code>sharingModel</code>—Sets a sharing model. Different objects support different sharing models. Possible values of sharing models are: <ul style="list-style-type: none"> – <code>private</code> – <code>read</code> – <code>readWrite</code> – <code>readWriteTransfer</code> – <code>fullAccess</code> – <code>controlledByParent</code> – <code>controlledByCampaign</code> – <code>controlledByLeadOrContent</code> • <code>defaultRecordType</code>—Creates a record type. This setting is required before installing a package that creates record types. Specify an alphanumeric string that starts with a lowercase letter.
<code><custom field API name></code>	No	None. Useful for Dev Ops use cases where you want to track extra information on the <code>ScratchOrgInfo</code> object. First, create the custom field , then reference it in the scratch org definition by its API name.
sourceOrg	No	None. 15-character source org ID. Use only if you are using Org Shape for Scratch orgs to create your scratch org. See Create a Scratch Org Based on an Org Shape .

Sample Scratch Org Definition File

Here's what the scratch org definition JSON file looks like. For more information on features and settings, see [Scratch Org Features](#).

```
{
  "orgName": "Acme",
  "edition": "Enterprise",
  "features": ["Communities", "ServiceCloud", "Chatbot"],
```

```

"settings": {
    "communitiesSettings": {
        "enableNetworksEnabled": true
    },
    "mobileSettings": {
        "enableS1EncryptedStoragePref2": true
    },
    "omniChannelSettings": {
        "enableOmniChannel": true
    },
    "caseSettings": {
        "systemUserEmail": "support@acme.com"
    }
}
}

```

Some features, such as Communities, can require a combination of a feature and a setting to work correctly for scratch orgs. This code snippet sets both the feature and associated setting.

```

"features": ["Communities"],
"settings": {
    "communitiesSettings": {
        "enableNetworksEnabled": true
    },
    ...
}

```

Create a Custom Field for ScratchOrgInfo

You can add more options to the scratch org definition to manage your Dev Ops process. To do so, create a custom field on the [ScratchOrgInfo](#) object. (ScratchOrgInfo tracks scratch org creation and deletion.)

! **Important:** If you're making these changes directly in your production org, proceed with the appropriate levels of caution. The ScratchOrgInfo object is not available in sandboxes or scratch orgs.

1. In the Dev Hub org, create the custom field.
 - a. From Setup, enter *Object Manager* in the Quick Find box, then select **Object Manager**.
 - b. Click **Scratch Org Info**.
 - c. In Fields & Relationships, click **New**.
 - d. Define the custom field, then click **Save**.
2. After you create the custom field, you can pass it a value in the scratch org definition file by referencing it with its API name.

Let's say you create two custom fields called `workitem` and `release`. Add the custom fields and associated values to the scratch org definition:

```

{
    "orgName": "MyCompany",
    "edition": "Developer",
    "workitem__c": "W-12345678",
    "release__c": "June 2018 pilot",

    "settings": {
        "omniChannelSettings": {
}

```

```

        "enableOmniChannel": true
    }
}
}
```

3. Create the scratch org.

Set Object-Level Sharing Settings and Default Record Types

To install successfully, some packages require that you define object-level sharing settings and default record types before installation. Set the sharing settings and default record types with objectSettings. In this sample scratch org definition file, we set a sharing model and a default record type for opportunity, and a default record type for account.

```
{
  "orgName": "MyCompany",
  "edition": "Developer",
  "features": ["Communities", "ServiceCloud", "Chatbot"],
  "settings": {
    "communitiesSettings": {
      "enableNetworksEnabled": true
    }
  },
  "objectSettings": {
    "opportunity": {
      "sharingModel": "private",
      "defaultRecordType": "default"
    },
    "account": {
      "defaultRecordType": "default"
    }
  }
}
```

Scratch Org Features

The scratch org definition file contains the configuration values that determine the shape of the scratch org. You can enable these supported add-on features in a scratch org.

Scratch Org Settings

In Winter '19 and later, scratch org settings are the format for defining org preferences in the scratch org definition. Because you can use all Metadata API settings, they're the most comprehensive way to configure a scratch org. If a setting is supported in Metadata API, it's supported in scratch orgs. Settings provide you with fine-grained control because you can define values for all fields for a setting, rather than just enabling or disabling it.

Scratch Org Features

The scratch org definition file contains the configuration values that determine the shape of the scratch org. You can enable these supported add-on features in a scratch org.

Supported Features

Features aren't case-sensitive. You can indicate them as all-caps, or as we define them here for readability. If a feature is followed by <value>, you must specify a value as an incremental allocation or limit.

You can specify multiple feature values in a comma-delimited list in the scratch org definition file.

```
"features": ["MultiCurrency", "AuthorApex"],
```

[AccountingSubledgerUser](#)

Enables organization-wide access to Accounting Subledger Growth features when the package is installed.

[AddCustomApps:<value>](#)

Increases the maximum number of custom apps allowed in an org. Indicate a value from 1–30.

[AddCustomObjects:<value>](#)

Increases the maximum number of custom objects allowed in the org. Indicate a value from 1–30.

[AddCustomRelationships:<value>](#)

Increases the maximum number of custom relationships allowed on an object. Indicate a value from 1–10.

[AddCustomTabs:<value>](#)

Increases the maximum number of custom tabs allowed in an org. Indicate a value from 1–30.

[AddDataComCRMRecordCredit:<value>](#)

Increases record import credits assigned to a user in your scratch org. Indicate a value from 1–30.

[AddInsightsQueryLimit:<value>](#)

Increases the size of your CRM Analytics query results. Indicate a value from 1–30 (multiplier is 10). Setting the quantity to 6 increases the query results to 60.

[AdditionalFieldHistory:<value>](#)

Increases the number of fields you can track history for beyond the default, which is 20 fields. Indicate a value between 1–40.

[AIAttribution](#)

Provides access to Einstein Attribution for Marketing Cloud Account Engagement. Einstein Attribution uses AI modeling to dynamically assign attribution percentages to multiple campaign touchpoints.

[AnalyticsAdminPerms](#)

Enables all permissions required to administer the CRM Analytics platform, including permissions to enable creating CRM Analytics templated apps and CRM Analytics Apps.

[AnalyticsAppEmbedded](#)

Provides one CRM Analytics Embedded App license for the CRM Analytics platform.

[API](#)

Even in the editions (Professional, Group) that don't provide API access, REST API is enabled by default. Use this scratch org feature to access additional APIs (SOAP, Streaming, Bulk, Bulk 2.0).

[Assessments](#)

Enables dynamic Assessments features, which enables both Assessment Questions and Assessment Question Sets.

[AssetScheduling:<value>](#)

Enables Asset Scheduling license. Asset Scheduling makes it easier to book rooms and equipments. Indicate a value between 1–10.

[AuthorApex](#)

Enables you to access and modify Apex code in a scratch org. Enabled by default in Enterprise and Developer Editions.

B2BCommerce

Provides the B2B License. B2BCommerce enables business-to-business (B2B) commerce in your org. Create and update B2B stores. Create and manage buyer accounts. Sell products to other businesses.

B2BLoyaltyManagement

Enables the B2B Loyalty Management license. Create loyalty programs and set up loyalty program-specific processes that allow you to recognize, rewards, and retain customers.

B2CCommerceGMV

Provides the B2B2C Commerce License. B2B2C Commerce allows you to quickly stand up an ecommerce site to promote brands and sell products into multiple digital channels. You can create and update retail storefronts in your org, and create and manage person accounts.

B2CLoyaltyManagement

Enables the Loyalty Management - Growth license. Create loyalty programs and set up loyalty program-specific processes that allow you to recognize, rewards, and retain customers.

B2CLoyaltyManagementPlus

Enables the Loyalty Management - Advanced license. Create loyalty programs and set up loyalty program-specific processes that allow you to recognize, rewards, and retain customers.

BatchManagement

Enables the Batch Management license. Batch Management allows you to process a high volume of records in manageable batches.

BigObjectsBulkAPI

Enables the scratch org to use BigObjects in the Bulk API.

Briefcase

Enables the use of Briefcase Builder in a scratch org, which allows you to create offline briefcases that make selected records available for viewing offline.

BudgetManagement

Gives users access to budget management features and objects. To enable budget management, add this feature to your scratch org definition file.

BusinessRulesEngine

Enables Business Rules Engine, which enables both expression sets and lookup tables.

CacheOnlyKeys

Enables the cache-only keys service. This feature allows you to store your key material outside of Salesforce, and have the Cache-Only Key Service fetch your key on demand from a key service that you control.

CalloutSizeMB:<value>

Increases the maximum size of an Apex callout. Indicate a value between 3–12.

CampaignnInfluence2

Provides access to Customizable Campaign Influence for Sales Cloud and Marketing Cloud Account Engagement. Customizable Campaign Influence can auto-associate or allow manual creation of relationships among campaigns and opportunities to track attribution.

CascadeDelete

Provides lookup relationships with the same cascading delete functionality previously only available to master-detail relationships. To prevent records from being accidentally deleted, cascade-delete is disabled by default.

CaseClassification

Enables Einstein Case Classification. Case Classification offers recommendations to your agents so they can select the best value. You can also automatically save the best recommendation and route the case to the right agent.

[CaseWrapUp](#)

Enables Einstein Case Wrap-Up. To help agents complete cases quickly, Einstein Case Wrap-Up recommends case field values based on past chat transcripts.

[ChangeDataCapture](#)

Enables Change Data Capture, if the scratch org edition doesn't automatically enable it.

[Chatbot](#)

Enables deployment of Bot metadata into a scratch org, and allows you to create and edit bots.

[ChatterEmailFooterLogo](#)

ChatterEmailFooterLogo allows you to use the Document ID of a logo image, which you can use to customize chatter emails.

[ChatterEmailFooterText](#)

ChatterEmailFooterText allows you to use footer text in customized Chatter emails.

[ChatterEmailSenderName](#)

ChatterEmailSenderName allows you to customize the name that appears as the sender's name in the email notification. For example, your company's name.

[CloneApplication](#)

CloneApplication allows you to clone an existing custom Lightning app and make required customizations to the new app. This way, you don't have to start from scratch, especially when you want to create apps with simple variations.

[CMSMaxContType](#)

Limits the number of distinct content types you can create within Salesforce CMS to 21.

[CMSMaxNodesPerContType](#)

Limits the maximum number of child nodes (fields) you can create for a particular content type to 15.

[CMSUnlimitedUse](#)

Enables unlimited content records, content types, and bandwidth usage in Salesforce CMS.

[Communities](#)

Allows the org to create a customer community. To use Communities, you must also include communitiesSettings > enableNetworksEnabled in the settings section of your scratch org definition file.

[ConAppPluginExecuteAsUser](#)

Enables the pluginExecutionUser field in the ConnectedApp Metadata API object.

[ConStreamingClients:<value>](#)

Increases the maximum number of concurrent clients (subscribers) across all channels and for all event types for API version 36.0 and earlier. Indicate a value between 20–4,000.

[ConnectedAppCustomNotifSubscription](#)

Enables connected apps to subscribe to custom notification types, which are used to send custom desktop and mobile notifications.

[ConnectedAppToolingAPI](#)

Enables the use of connected apps with the Tooling API.

[ConsentEventStream](#)

Enables the Consent Event Stream permission for the org.

[ConsolePersistenceInterval:<value>](#)

Increases how often console data is saved, in minutes. Indicate a value between 0–500. To disable auto save, set the value to 0.

[ContactsToMultipleAccounts](#)

Enables the contacts to multiple accounts feature. This feature lets you relate a contact to two or more accounts.

[ContractApprovals](#)

Enables contract approvals, which allow you to track contracts through an approval process.

[CPQ](#)

Enables the licensed features required to install the Salesforce CPQ managed package. Doesn't install the package automatically.

[CustomFieldDataTranslation](#)

Enables translation of custom field data for Work Type Group, Service Territory, and Service Resource objects. You can enable data translation for custom fields with Text, Text Area, Text Area (Long), Text Area (Rich), and URL types.

[CustomNotificationType](#)

Allows the org to create custom notification types, which are used to send custom desktop and mobile notifications.

[DataComDnbAccounts](#)

Provides a license to Data.com account features.

[DataComFullClean](#)

Provides a license to Data.com cleaning features, and allows users to turn on auto fill clean settings for jobs.

[DataMaskUser](#)

Provides 30 Data Mask permission set licenses. This permission set enables access to an installed Salesforce Data Mask package.

[DataProcessingEngine](#)

Enables the Data Processing Engine license. Data Processing Engine helps transform data that's available in your Salesforce org and write back the transformation results as new or updated records.

[DebugApex](#)

Enables Apex Interactive Debugger. You can use it to debug Apex code by setting breakpoints and checkpoints, and inspecting your code to find bugs.

[DecisionTable](#)

Enables Decision Table license. Decision tables read business rules and decide the outcome for records in your Salesforce org or for the values that you specify.

[DefaultWorkflowUser](#)

Sets the scratch org admin as the default workflow user.

[DeferSharingCalc](#)

Allows admins to suspend group membership and sharing rule calculations and to resume them later.

[DevelopmentWave](#)

Enables CRM Analytics development in a scratch org. It assigns five platform licenses and five CRM Analytics platform licenses to the org, along with assigning the permission set license to the admin user. It also enables the CRM Analytics Templates and Einstein Discovery features.

[DeviceTrackingEnabled](#)

Enables Device Tracking.

[DevOpsCenter](#)

Enables DevOps Center in scratch orgs so that partners can create second-generation managed packages that extend or enhance the functionality in the DevOps Center application (base) package.

[DisableManageConfAPI](#)

Limits access to the LoginIP and ClientBrowser API objects to allow view or delete only.

[DisclosureFramework](#)

Provides the permission set licenses and permission sets required to configure Disclosure and Compliance Hub.

Division

Turns on the Manage Divisions feature under Company Settings. Divisions let you segment your organization's data into logical sections, making searches, reports, and list views more meaningful to users. Divisions are useful for organizations with extremely large amounts of data.

DocumentChecklist

Enables Document Tracking and Approval features, and adds the Document Checklist permission set. Document tracking features let you define documents to upload and approve, which supports processes like loan applications or action plans.

DocumentReaderPageLimit

Limits the number of pages sent for data extraction to 5.

DSARPortability

Enables an org to access the DSARPortability feature in Privacy Center. Also, provides one seat each of the PrivacyCenter and PrivacyCenterAddOn licenses.

DurableClassicStreamingAPI

Enables Durable PushTopic Streaming API for API version 37.0 and later.

DurableGenericStreamingAPI

Enables Durable Generic Streaming API for API version 37.0 and later.

DynamicClientCreationLimit

Allows the org to register up to 100 OAuth 2.0 connected apps through the dynamic client registration endpoint.

EinsteinAnalyticsPlus

Provides one CRM Analytics Plus license for the CRM Analytics platform.

EinsteinArticleRecommendations

Provides licenses for Einstein Article Recommendations. Einstein Article Recommendations uses data from past cases to identify Knowledge articles that are most likely to help your customer service agents address customer inquiries.

EinsteinBuilderFree

Provides a license that allows admins to create one enabled prediction with Einstein Prediction Builder. Einstein Prediction Builder is custom AI for admins

EinsteinDocReader

Provides the license required to enable and use Intelligent Form Reader in a scratch org. Intelligent Form Reader uses optical character recognition to automatically extract data with Amazon Textract.

EinsteinRecommendationBuilder

Provides a license to create recommendations with Einstein Recommendation Builder. Einstein Recommendation Builder lets you build custom AI recommendations.

EinsteinRecommendationBuilderMetadata

Enables Einstein Recommendation Builder to use the required metadata APIs. Enabling this feature lets you build custom AI recommendations.

EinsteinSearch

Provides the license required to use and enable Einstein Search features in a scratch org.

EinsteinVisits

Enables Consumer Goods Cloud. With Consumer Goods cloud, transform the way you collaborate with your retail channel partners. Empower your sales managers to plan visits and analyze your business's health across stores. Also, allow your field reps to track inventory, take orders, and capture visit details using the Retail Execution mobile app.

[EinsteinVisitsED](#)

Enables Einstein Discovery, which can be used to get store visit recommendations. With Einstein Visits ED, you can create a visit frequency strategy that allows Einstein to provide optimal store visit recommendations.

[EmbeddedLoginForIE](#)

Provides JavaScript files that support Embedded Login in IE11.

[EmpPublishRateLimit:<value>](#)

Increases the maximum number of standard-volume platform event notifications published per hour. Indicate a value between 1,000–10,000.

[EnablePRM](#)

Enables the partner relationship management permissions for the org.

[EnableManageIdConfUI](#)

Enables access to the LoginIP and ClientBrowser API objects to verify a user's identity in the UI.

[EnableSetPasswordInApi](#)

Enables you to use `sfdx force:user:password:generate:` to change a password without providing the old password.

[EncryptionStatisticsInterval:<value>](#)

Defines the interval (in seconds) between encryption statistics gathering processes. The maximum value is 604,800 seconds (7 days). The default is once per 86,400 seconds (24 hours).

[EncryptionSyncInterval:<value>](#)

Defines how frequently (in seconds) the org can synchronize data with the active key material. The default and maximum value is 604,800 seconds (7 days). To synchronize data more frequently, indicate a value, in seconds, equal to or larger than 0.

[Entitlements](#)

Enables entitlements. Entitlements are units of customer support in Salesforce, such as phone support or web support that represent terms in service agreements.

[EventLogFile](#)

Enables API access to your org's event log files. The event log files contain information about your org's operational events that you can use to analyze usage trends and user behavior.

[EntityTranslation](#)

Enables translation of field data for Work Type Group, Service Territory, and Service Resource objects.

[ExpressionSetMaxExecPerHour](#)

Enables an org to run a maximum of 500,000 expression sets per hour by using Connect REST API.

[ExternalIdentityLogin](#)

Allows the scratch org to use Salesforce Customer Identity features associated with your External Identity license.

[FieldAuditTrail](#)

Enables Field Audit Trail for the org and allows a total 60 tracked fields. By default, 20 fields are tracked for all orgs, and 40 more are tracked with Field Audit Trail.

[FieldService:<value>](#)

Provides the Field Service license. Indicate a value between 1–25.

[FieldServiceDispatcherUser:<value>](#)

Adds the Field Service Dispatcher permission set license. Indicate a value between 1–25.

[FieldServiceMobileUser:<value>](#)

Adds the Field Service Mobile permission set license. Indicate a value between 1–25.

[**FieldServiceSchedulingUser:<value>**](#)

Adds the Field Service Scheduling permission set license. Indicate a value between 1–25.

[**FinanceLogging**](#)

Adds Finance Logging objects to a scratch org. This feature is required for Finance Logging.

[**FinancialServicesCommunityUser:<value>**](#)

Adds the Financial Services Insurance Community permission set license, and enables access to Financial Services insurance community components and objects. Indicate a value between 1–10.

[**FinancialServicesInsuranceUser:<value>**](#)

Adds the Financial Services Insurance permission set license, and enables access to Financial Services insurance components and objects. Indicate a value between 1–10.

[**FinancialServicesUser:<value>**](#)

Adds the Financial Services Cloud Standard permission set license. This permission set enables access to Lightning components and the standard version of Financial Services Cloud. Also provides access to the standard Salesforce objects and custom Financial Services Cloud objects. Indicate a value between 1–10.

[**FlowSites**](#)

Enables the use of flows in Salesforce Sites and customer portals.

[**ForceComPlatform**](#)

Adds one Salesforce Platform user license.

[**FSCServiceProcess**](#)

Enables the Service Process Studio feature of Financial Service Cloud. Provides 10 seats each of the IndustriesServiceExcellenceAddOn and FinancialServicesCloudStandardAddOn licenses. To enable the feature, you must also turn on the StandardServiceProcess setting in Setup and grant users the AccessToServiceProcess permission.

[**GenericStreaming**](#)

Enables Generic Streaming API for API version 36.0 and earlier.

[**GenStreamingEventsPerDay:<value>**](#)

Increases the maximum number of delivered event notifications within a 24-hour period, shared by all CometD clients, with generic streaming for API version 36.0 and earlier. Indicate a value between 10,000–50,000.

[**Grantmaking**](#)

Gives users access to Grantmaking features and objects in Salesforce and Experience Cloud.

[**HealthCloudAddOn**](#)

Enables use of Health Cloud.

[**HealthCloudELOOverride**](#)

Salesforce retired the Health Cloud CandidatePatient object in Spring '22 to focus on the more robust Lead object. This scratch org feature allows you to override that retirement and access the object.

[**HealthCloudForCmty**](#)

Enables use of Health Cloud for Experience Cloud Sites.

[**HealthCloudMedicationReconciliation**](#)

Allows Medication Management to support Medication Reconciliation.

[**HealthCloudPNMAddOn**](#)

Enables use of Provider Network Management.

[HealthCloudUser](#)

This enables the scratch org to use the Health Cloud objects and features equivalent to the Health Cloud permission set license for one user.

[HighVelocitySales](#)

Provides Sales Engagement licenses and enables Salesforce Inbox. Sales Engagement optimizes the inside sales process with a high-productivity workspace. Sales managers can create custom sales processes that guide reps through handling different types of prospects. And sales reps can rapidly handle prospects with a prioritized list and other productivity-boosting features. The Sales Engagement feature can be deployed in scratch orgs, but the settings for the feature can't be updated through the scratch org definition file. Instead, configure settings directly in the Sales Engagement app.

[HoursBetweenCoverageJob:<value>](#)

The frequency in hours when the sharing inheritance coverage report can be run for an object. Indicate a value between 1–24.

[IdentityProvisioningFeatures](#)

Enables use of Salesforce Identity User Provisioning.

[IndustriesActionPlan](#)

Provides a license for Action Plans. Action Plans allow you to define the tasks or document checklist items for completing a business process.

[IndustriesMfgTargets](#)

Enables Sales Agreements. With Sales Agreements, you can negotiate purchase and sale of products over a continued period. You can also get insights into products, prices, discounts, and quantities. And you can track your planned and actual quantities and revenues with real-time updates from orders and contracts.

[IndustriesManufacturingCmty](#)

Provides the Manufacturing Sales Agreement for the Community permission set license, which is intended for the usage of partner community users. It also provides access to the Manufacturing community template for admins users to create communities.

[IndustriesMfgAccountForecast](#)

Enables Account Forecast. With Account Forecast, you can generate forecasts for your accounts based on orders, opportunities, and sales agreements. You can also create formulas to calculate your forecasts per the requirements of your company.

[InsightsPlatform](#)

Enables the CRM Analytics Plus license for CRM Analytics.

[InsuranceCalculationUser](#)

Enables the calculation feature of Insurance. Provides 10 seats each of the BRERuntimeAddOn and OmniStudioRuntime licenses. Also, provides one seat each of the OmniStudio and BREPlatformAccess licenses.

[InsuranceClaimMgmt](#)

Enables claim management features. Provides one seat of the InsuranceClaimMgmtAddOn license.

[InsurancePolicyAdmin](#)

Enables policy administration features. Provides one seat of the InsurancePolicyAdministrationAddOn license.

[IntelligentDocumentReader](#)

Provides the license required to enable and use Intelligent Document Reader in a scratch org. Intelligent Document Reader uses optical character recognition to automatically extract data with Amazon Textract by using your AWS account.

[Interaction](#)

Enables flows. A flow is the part of Salesforce Flow that collects data and performs actions in your Salesforce org or an external system. Salesforce Flow provides two types of flows: screen flows and autolaunched flows.

[IoT](#)

Enables IoT so the scratch org can consume platform events to perform business and service workflows using orchestrations and contexts.

[JigsawUser](#)

Provides one license to Jigsaw features.

[Knowledge](#)

Enables Salesforce Knowledge and gives your website visitors, clients, partners, and service agents the ultimate support tool. Create and manage a knowledge base with your company information, and securely share it when and where it's needed. Build a knowledge base of articles that can include information on process, like how to reset your product to its defaults, or frequently asked questions.

[LegacyLiveAgentRouting](#)

Enables legacy Live Agent routing for Chat. Use Live Agent routing to chat in Salesforce Classic. Chats in Lightning Experience must be routed using Omni-Channel.

[LightningSalesConsole](#)

Adds one Lightning Sales Console user license.

[LightningScheduler](#)

Enables Lightning Scheduler. Lightning Scheduler gives you tools to simplify appointment scheduling in Salesforce. Create a personalized experience by scheduling customer appointments—in person, by phone, or by video—with the right person at the right place and time.

[LightningServiceConsole](#)

Assigns the Lightning Service Console License to your scratch org so you can use the Lightning Service Console and access features that help manage cases faster.

[LiveAgent](#)

Enables Chat for Service Cloud. Use web-based chat to quickly connect customers to agents for real-time support.

[LiveMessage](#)

Enables Messaging for Service Cloud. Use Messaging to quickly support customers using apps such as SMS text messaging and Facebook Messenger.

[LongLayoutSectionTitles](#)

Allows page layout section titles to be up to 80 characters.

[LoyaltyAnalytics](#)

Enables Analytics for Loyalty license. The Analytics for Loyalty app gives you actionable insights into your loyalty programs.

[LoyaltyEngine](#)

Enables Loyalty Management Promotion Setup license. Promotion setup allows loyalty program managers to create loyalty program processes. Loyalty program processes help you decide how incoming and new Accrual and Redemption-type transactions are processed.

[LoyaltyManagementStarter](#)

Enables the Loyalty Management - Starter license. Create loyalty programs and set up loyalty program-specific processes that allow you to recognize, rewards, and retain customers.

[LoyaltyMaximumPartners:<value>](#)

Increases the number of loyalty program partners that can be associated with a loyalty program in an org where the Loyalty Management - Starter license is enabled. The default and maximum value is 1.

[LoyaltyMaximumPrograms:<value>](#)

Increases the number of loyalty programs that can be created in an org where the Loyalty Management - Starter license is enabled. The default and maximum value is 1.

LoyaltyMaxOrderLinePerHour:<value>

Increases the number of order lines that can be cumulatively processed per hour by loyalty program processes. Indicate a value between 1–3,500,000.

LoyaltyMaxProcExecPerHour:<value>

Increases the number of transaction journals that can be processed by loyalty program processes per hour. Indicate a value between 1–500,000.

LoyaltyMaxTransactions:<value>

Increases the number of Transaction Journal records that can be processed. Indicate a value between 1–50,000,000.

LoyaltyMaxTrxnJournals:<value>

Increases the number of Transaction Journal records that can be stored in an org that has the Loyalty Management - Start license enabled.

Macros

Enables macros in your scratch org. After enabling macros, add the macro browser to the Lightning Console so you can configure predefined instructions for commonly used actions and apply them to multiple posts at the same time.

MarketingUser

Provides access to the Campaigns object. Without this setting, Campaigns are read-only.

MaxActiveDPEDefs:<value>

Increases the number of Data Processing Engine definitions that can be activated in the org. Indicate a value between 1–50.

MaxApexCodeSize:<value>

Limits the non-test, unmanaged Apex code size (in MB). To use a value greater than the default value of 10, contact Salesforce Customer Support.

MaxAudTypeCriterionPerAud

Limits the number of audience type criteria available per audience. The default value is 10.

MaxCustomLabels:<value>

Limits the number of custom labels (measured in thousands). Setting the limit to 10 enables the scratch org to have 10,000 custom labels. Indicate a value between 1–15.

MaxDatasetLinksPerDT:<value>

Increases the number of dataset links that can be associated with a decision table. Indicate a value between 1–3.

MaxDataSourcesPerDPE:<value>

Increases the number of Source Object nodes a Data Processing Engine definition can contain. Indicate a value between 1–50.

MaxDecisionTableAllowed:<value>

Increases the number of decision tables rules that can be created in the org. Indicate a value between 1–30.

MaxFavoritesAllowed:<value>

Increases the number of Favorites allowed. Favorites allow users to create a shortcut to a Salesforce Page. Users can view their Favorites by clicking the Favorites list dropdown in the header. Indicate a value between 0–200.

MaxFieldsPerNode:<value>

Increases the number of fields a node in a Data Processing Engine definition can contain. Indicate a value between 1–500.

MaxInputColumnsPerDT:<value>

Increases the number of input fields a decision table can contain. Indicate a value between 1–10.

MaxLoyaltyProcessRules:<value>

Increases the number of loyalty program process rules that can be created in the org. Indicate a value between 1–20.

[**MaxNodesPerDPE:<value>**](#)

Increases the number of nodes that a Data Processing Engine definition can contain. Indicate a value between 1–500.

[**MaxNoOfLexThemesAllowed:<value>**](#)

Increases the number of Themes allowed. Themes allow users to configure colors, fonts, images, sizes, and more. Access the list of Themes in Setup, under Themes and Branding. Indicate a value between 0–300.

[**MaxOutputColumnsPerDT:<value>**](#)

Increases the number of output fields a decision table can contain. Indicate a value between 1–5.

[**MaxSourceObjectPerDSL:<value>**](#)

Increases the number of source objects that can be selected in a dataset link of a decision table. Indicate a value between 1–5.

[**MaxStreamingTopics:<value>**](#)

Increases the maximum number of delivered PushTopic event notifications within a 24-hour period, shared by all CometD clients. Indicate a value between 40–100.

[**MaxUserNavItemsAllowed:<value>**](#)

Increases the number of navigation items a user can add to the navigation bar. Indicate a value between 0–500.

[**MaxUserStreamingChannels:<value>**](#)

Increases the maximum number of user-defined channels for generic streaming. Indicate a value between 20–1,000.

[**MaxWritebacksPerDPE:<value>**](#)

Increases the number of Writeback Object nodes a Data Processing Engine definition can contain. Indicate a value between 1–50.

[**MedVisDescriptorLimit:<value>**](#)

Increases the number of sharing definitions allowed per record for sharing inheritance to be applied to an object. Indicate a value between 150–1,600.

[**MinKeyRotationInterval**](#)

Sets the encryption key material rotation interval at once per 60 seconds. If this feature isn't specified, the rotation interval defaults to once per 604,800 seconds (7 days) for Search Index key material, and once per 86,400 seconds (24 hours) for all other key material.

[**MobileExtMaxFileSizeMB:<value>**](#)

Increases the file size (in megabytes) for Field Service Mobile extensions. Indicate a value between 1–2,000.

[**MobileSecurity**](#)

Enables Enhanced Mobile Security. With Enhanced Mobile Security, you can control a range of policies to create a security solution tailored to your org's needs. You can limit user access based on operating system versions, app versions, and device and network security. You can also specify the severity of a violation.

[**MultiCurrency**](#)

Enables the scratch org to set up and use multiple currencies in opportunities, forecasts, quotes, reports, and other data.

[**MultiLevelMasterDetail**](#)

Allows the creation a special type of parent-child relationship between one object, the child, or detail, and another object, the parent, or master.

[**MutualAuthentication**](#)

Requires client certificates to verify inbound requests for mutual authentication.

[**MyTrailhead**](#)

Enables access to a myTrailhead enablement site in a scratch org.

[**NonprofitCloudCaseManagementUser**](#)

Provides the permission set license required to use and configure the Salesforce.org Nonprofit Cloud Case Management managed package. You can then install the package in the scratch org.

[NumPlatformEvents:<value>](#)

Increases the maximum number of platform event definitions that can be created. Indicate a value between 5–20.

[ObjectLinking](#)

Create rules to quickly link channel interactions to objects such as contacts, leads, or person accounts for customers (Beta).

[OrderManagement](#)

Provides the Salesforce Order Management license. Order Management is your central hub for handling all aspects of the order lifecycle, including order capture, fulfillment, shipping, payment processing, and servicing.

[OrderSaveLogicEnabled](#)

Enables scratch org support for New Order Save Behavior.

[OrderSaveBehaviorBoth](#)

Enables scratch org support for both New Order Save Behavior and Old Order Save Behavior.

[OutboundMessageHTTPSession](#)

Enables using HTTP endpoint URLs in outbound message definitions that have the Send Session ID option selected.

[PardotScFeaturesCampaignnInfluence](#)

Enables additional campaign influence models, first touch, last touch, and even distribution for Pardot users.

[PersonAccounts](#)

Enables person accounts in your scratch org.

[PipelineInspection](#)

Enables the Pipeline Inspection. Pipeline Inspection is a consolidated pipeline view with metrics, opportunities, and highlights of recent changes.

[PlatformCache](#)

Enables Platform Cache and allocates a 3 MB cache. The Lightning Platform Cache layer provides faster performance and better reliability when caching Salesforce session and org data.

[PlatformConnect:<value>](#)

Enables Salesforce Connect and allows your users to view, search, and modify data that's stored outside your Salesforce org. Indicate a value from 1–5.

[PlatformEncryption](#)

Shield Platform Encryption encrypts data at rest. You can manage key material and encrypt fields, files, and other data.

[PlatformEventsPerDay:<value>](#)

Increases the maximum number of delivered standard-volume platform event notifications within a 24-hour period, shared by all CometD clients. Indicate a value between 10,000–50,000.

[ProcessBuilder](#)

Enables Process Builder, a Salesforce Flow tool that helps you automate your business processes.

[ProductsAndSchedules](#)

Enables product schedules in your scratch org. Enabling this feature lets you create default product schedules on products. Users can also create schedules for individual products on opportunities.

[ProgramManagement](#)

Enables access to all Program Management and Case Management features and objects.

[ProviderFreePlatformCache](#)

Provides 3 MB of free Platform Cache capacity for AppExchange-certified and security-reviewed managed packages. This feature is made available through a capacity type called Provider Free capacity and is automatically enabled in Developer Edition orgs. Allocate the Provider Free capacity to a Platform Cache partition and add it to your managed package.

[PublicSectorAccess](#)

Enables access to all Public Sector features and objects.

[PublicSectorApplicationUsageCreditsAddOn](#)

Enables additional usage of Public Sector applications based on their pricing.

[PublicSectorSiteTemplate](#)

Allows Public Sector users access to build an Experience Cloud site from the templates available.

[RecordTypes](#)

Enables Record Type functionality. Record Types let you offer different business processes, picklist values, and page layouts to different users.

[RefreshOnInvalidSession](#)

Enables automatic refreshes of Lightning pages when the user's session is invalid. If, however, the page detects a new token, it tries to set that token and continue without a refresh.

[RevSubscriptionManagement](#)

Enables Subscription Management. Subscription Management is an API-first, product-to-cash solution for B2B subscriptions and one-time sales.

[S1ClientComponentCacheSize](#)

Allows the org to have up to 5 pages of caching for Lightning Components.

[SalesCloudEinstein](#)

Enables Sales Cloud Einstein features and Salesforce Inbox. Sales Cloud Einstein brings AI to every step of the sales process.

[SalesforceContentUser](#)

Enables access to Salesforce content features.

[SalesforceFeedbackManagementStarter](#)

Provides a license to use the Salesforce Feedback Management - Starter features.

[SalesforceIdentityForCommunities](#)

Adds Salesforce Identity components, including login and self-registration, to Experience Builder. This feature is required for Aura components.

[SalesUser](#)

Provides a license for Sales Cloud features.

[SAML20SingleLogout](#)

Enables usage of SAML 2.0 single logout.

[SCIMProtocol](#)

Enables access support for the SCIM protocol base API.

[SecurityEventEnabled](#)

Enables access to security events in Event Monitoring.

[SentimentInsightsFeature](#)

Provides the license required to enable and use Sentiment Insights in a scratch org. Use Sentiment Insights to analyze the sentiment of your customers and get actionable insights to improve it.

[ServiceCatalog](#)

Enables Employee Service Catalog so you can create a catalog of products and services for your employees. It can also turn your employees' requests for these products and services into approved and documented orders.

[ServiceCloud](#)

Assigns the Service Cloud license to your scratch org, so you can choose how your customers can reach you, such as by email, phone, social media, online communities, chat, and text.

[ServiceCloudVoicePartnerTelephony](#)

Assigns the Service Cloud Voice with Partner Telephony add-on license to your scratch org, so you can set up a Service Cloud Voice contact center that integrates with supported telephony providers. Indicate a value from 1–50.

[ServiceUser](#)

Adds one Service Cloud User license, and allows access to Service Cloud features.

[SessionIdInLogEnabled](#)

Enables Apex debug logs to include session IDs. If disabled, session IDs are replaced with "SESSION_ID_REMOVED" in debug logs.

[SFDOInsightsDataIntegrityUser](#)

Provides a license to Salesforce.org Insights Platform Data Integrity managed package. You can then install the package in the scratch org.

[SharedActivities](#)

Allow users to relate multiple contacts to tasks and events.

[Sites](#)

Enables Salesforce Sites, which allows you to create public websites and applications that are directly integrated with your Salesforce org. Users aren't required to log in with a username and password.

[SocialCustomerService](#)

Enables Social Customer Service, sets post defaults, and either activates the Starter Pack or signs into your Social Studio account.

[StateAndCountryPicklist](#)

Enables state and country/territory picklists. State and country/territory picklists let users select states and countries from predefined, standardized lists, instead of entering state, country, and territory data into text fields.

[StreamingAPI](#)

Enables Streaming API.

[StreamingEventsPerDay:<value>](#)

Increases the maximum number of delivered PushTopic event notifications within a 24-hour period, shared by all CometD clients (API version 36.0 and earlier). Indicate a value between 10,000–50,000.

[SubPerStreamingChannel:<value>](#)

Increases the maximum number of concurrent clients (subscribers) per generic streaming channel (API version 36.0 and earlier). Indicate a value between 20–4,000.

[SubPerStreamingTopic:<value>](#)

Increases the maximum number of concurrent clients (subscribers) per PushTopic streaming channel (API version 36.0 and earlier). Indicate a value between 20–4,000.

[SurveyAdvancedFeatures](#)

Enables a license for the features available with the Salesforce Feedback Management - Growth license.

[SustainabilityCloud](#)

Provides the permission set licenses and permission sets required to install and configure Sustainability Cloud. To enable or use CRM Analytics and CRM Analytics templates, include the DevelopmentWave scratch org feature.

[SustainabilityApp](#)

Provides the permission set licenses and permission sets required to configure Net Zero Cloud. To enable or use Tableau CRM and Tableau CRM templates, include the DevelopmentWave scratch org feature.

[TCRMforSustainability](#)

Enables all permissions required to manage the Net Zero Analytics app by enabling Tableau CRM. You can create and share the analytics app for your users to bring your environmental accounting in line with your financial accounting.

[TimelineConditionsLimit](#)

Limits the number of timeline record display conditions per event type to 3.

[TimelineEventLimit](#)

Limits the number of event types displayed on a timeline to 5.

[TimelineRecordTypeLimit](#)

Limits the number of related object record types per event type to 3.

[TimeSheetTemplateSettings](#)

Time Sheet Templates let you configure settings to create time sheets automatically. For example, you can create a template that sets start and end dates. Assign templates to user profiles so that time sheets are created for the right users.

[TransactionFinalizers](#)

Enables you to implement and attach Apex Finalizers to Queueable Apex jobs.

[WaveMaxCurrency](#)

Increases the maximum number of supported currencies for CRM Analytics. Indicate a value between 1–5.

[WavePlatform](#)

Enables the Wave Platform license.

[Workflow](#)

Enables Workflow so you can automate standard internal procedures and processes.

[WorkflowFlowActionFeature](#)

Allows you to launch a flow from a workflow action.

[WorkplaceCommandCenterUser](#)

Enables access to Workplace Command Center features including access to objects such as Employee, Crisis, and EmployeeCrisisAssessment.

[WorkThanksPref](#)

Enables the give thanks feature in Chatter.

AccountingSubledgerUser

Enables organization-wide access to Accounting Subledger Growth features when the package is installed.

More Information

Requires that you install the Accounting Subledger or Accounting Subledger for Industries managed package. If you install the Accounting Subledger package, also set up the Opportunity object. See [Accounting Subledger Legacy Documentation](#) in Salesforce Help.

AddCustomApps:<value>

Increases the maximum number of custom apps allowed in an org. Indicate a value from 1–30.

Supported Quantities

1–30, Multiplier: 1

AddCustomObjects:<value>

Increases the maximum number of custom objects allowed in the org. Indicate a value from 1–30.

Supported Quantities

1–30, Multiplier: 1

AddCustomRelationships:<value>

Increases the maximum number of custom relationships allowed on an object. Indicate a value from 1–10.

Supported Quantities

1–10, Multiplier: 5

AddCustomTabs:<value>

Increases the maximum number of custom tabs allowed in an org. Indicate a value from 1–30.

Supported Quantities

1–30, Multiplier: 1

AddDataComCRMRecordCredit:<value>

Increases record import credits assigned to a user in your scratch org. Indicate a value from 1–30.

Supported Quantities

1–30, Multiplier: 1

AddInsightsQueryLimit:<value>

Increases the size of your CRM Analytics query results. Indicate a value from 1–30 (multiplier is 10). Setting the quantity to 6 increases the query results to 60.

Supported Quantities

1–30, Multiplier: 10

AdditionalFieldHistory:<value>

Increases the number of fields you can track history for beyond the default, which is 20 fields. Indicate a value between 1–40.

Supported Quantities

1–40, Multiplier: 1

More Information

Previous name: AddHistoryFieldsPerEntity.

AIAttribution

Provides access to Einstein Attribution for Marketing Cloud Account Engagement. Einstein Attribution uses AI modeling to dynamically assign attribution percentages to multiple campaign touchpoints.

Sample Scratch Org Definition File

Before enabling Einstein Attribution, make sure that `enableAIAttribution` and `enableCampaignInfluence2` are set to `true`.

```
{  
    "orgName": "NTOutfitters",  
    "edition": "Enterprise",  
    "features": ["AIAttribution"],  
    "settings": {  
        "campaignSettings": {  
            "enableAIAttribution": true  
            "enableCampaignInfluence2": true  
        }  
    }  
}
```

More Information

This feature is available in Account Engagement Advanced and Premium editions.

Optional configuration steps are accessible in Setup in the scratch org. For more information, see *Salesforce Help: Einstein Attribution*.

AnalyticsAdminPerms

Enables all permissions required to administer the CRM Analytics platform, including permissions to enable creating CRM Analytics templated apps and CRM Analytics Apps.

More Information

See [Set Up the CRM Analytics Platform](#) in Salesforce Help for more information.

AnalyticsAppEmbedded

Provides one CRM Analytics Embedded App license for the CRM Analytics platform.

API

Even in the editions (Professional, Group) that don't provide API access, REST API is enabled by default. Use this scratch org feature to access additional APIs (SOAP, Streaming, Bulk, Bulk 2.0).

More Information

See [Salesforce editions with API access](#) for more information.

Assessments

Enables dynamic Assessments features, which enables both Assessment Questions and Assessment Question Sets.

More Information

Add these options to your scratch org feature definition file. For "edition," you can indicate any of the supported scratch org feature editions.

```
{  
    "orgName": "Sample Org",  
    "edition": "Developer",  
}
```

```
"features": ["Assessments"],  
"settings": {  
    "industriesSettings": {  
        "enableIndustriesAssessment": true,  
        "enableDiscoveryFrameworkMetadata": true  
    }  
}
```

Add the Assessment to the page layout. See [Page Layouts](#) in Salesforce Help for more information.

AssetScheduling:<value>

Enables Asset Scheduling license. Asset Scheduling makes it easier to book rooms and equipments. Indicate a value between 1–10.

Supported Quantities

1–10

More Information

See [Enable Asset Scheduling in Salesforce Scheduler](#) in Salesforce Help for more information.

AuthorApex

Enables you to access and modify Apex code in a scratch org. Enabled by default in Enterprise and Developer Editions.

More Information

For Group and Professional Edition orgs, this feature is disabled by default. Enabling the AuthorApex feature lets you edit and test your Apex classes.

B2BCommerce

Provides the B2B License. B2BCommerce enables business-to-business (B2B) commerce in your org. Create and update B2B stores. Create and manage buyer accounts. Sell products to other businesses.

More Information

Requires that you also include the Communities scratch org feature in your scratch org definition file to create a store using B2B Commerce. Not available in Professional, Partner Professional, Group, or Partner Group Edition orgs.

B2BLoyaltyManagement

Enables the B2B Loyalty Management license. Create loyalty programs and set up loyalty program-specific processes that allow you to recognize, rewards, and retain customers.

More Information

See [Loyalty Management](#) in Salesforce Help for more information.

B2CCommerceGMV

Provides the B2B2C Commerce License. B2B2C Commerce allows you to quickly stand up an ecommerce site to promote brands and sell products into multiple digital channels. You can create and update retail storefronts in your org, and create and manage person accounts.

More Information

Also requires the Communities feature in your scratch org definition file.

Not available in Professional, Partner Professional, Group, or Partner Group Edition orgs.

For more information, see Salesforce Help at [Salesforce B2B Commerce and B2B2C Commerce](#).

B2CLoyaltyManagement

Enables the Loyalty Management - Growth license. Create loyalty programs and set up loyalty program-specific processes that allow you to recognize, rewards, and retain customers.

More Information

See [Loyalty Management](#) in Salesforce Help for more information.

B2CLoyaltyManagementPlus

Enables the Loyalty Management - Advanced license. Create loyalty programs and set up loyalty program-specific processes that allow you to recognize, rewards, and retain customers.

More Information

See [Loyalty Management](#) in Salesforce Help for more information.

BatchManagement

Enables the Batch Management license. Batch Management allows you to process a high volume of records in manageable batches.

More Information

See [Batch Management](#) in Salesforce Help for more information.

BigObjectsBulkAPI

Enables the scratch org to use BigObjects in the Bulk API.

More Information

See [Big Objects Implementation Guide](#) for more information.

Briefcase

Enables the use of Briefcase Builder in a scratch org, which allows you to create offline briefcases that make selected records available for viewing offline.

BudgetManagement

Gives users access to budget management features and objects. To enable budget management, add this feature to your scratch org definition file.

More Information

See [Budget Management](#) in Salesforce Help for more information.

BusinessRulesEngine

Enables Business Rules Engine, which enables both expression sets and lookup tables.

More Information

Provides 10 Business Rules Engine Designer and 10 Business Rules Engine Runtime licenses.

For more information, see [Business Rules Engine](#) in Salesforce Help.

CacheOnlyKeys

Enables the cache-only keys service. This feature allows you to store your key material outside of Salesforce, and have the Cache-Only Key Service fetch your key on demand from a key service that you control.

More Information

Requires enabling [PlatformEncryption](#) and configuration using the Setup menu in the scratch org. See [Which User Permissions Does Shield Platform Encryption Require?](#), [Generate a Tenant Secret with Salesforce](#), and [Cache-Only Key Service](#) in Salesforce Help.

CalloutSizeMB:<value>

Increases the maximum size of an Apex callout. Indicate a value between 3–12.

Supported Quantities

3–12, Multiplier: 1

CampaignInfluence2

Provides access to Customizable Campaign Influence for Sales Cloud and Marketing Cloud Account Engagement. Customizable Campaign Influence can auto-associate or allow manual creation of relationships among campaigns and opportunities to track attribution.

Sample Scratch Org Definition File

To enable Customizable Campaign Influence, set `enableCampaignInfluence2` to `true`.

```
{
  "orgName": "NTOutfitters",
  "edition": "Enterprise",
  "features": ["CampaignInfluence2"],
  "settings": {
    "campaignSettings": {
      "enableCampaignInfluence2": true
    }
  }
}
```

More Information

This feature is available in Salesforce Enterprise, Performance, Unlimited, and Developer Editions.

Optional configuration steps are accessible in Setup in the scratch org. For more information, see [Salesforce Help: Customizable Campaign Influence](#).

CascadeDelete

Provides lookup relationships with the same cascading delete functionality previously only available to master-detail relationships. To prevent records from being accidentally deleted, cascade-delete is disabled by default.

CaseClassification

Enables Einstein Case Classification. Case Classification offers recommendations to your agents so they can select the best value. You can also automatically save the best recommendation and route the case to the right agent.

CaseWrapUp

Enables Einstein Case Wrap-Up. To help agents complete cases quickly, Einstein Case Wrap-Up recommends case field values based on past chat transcripts.

More Information

Available in Enterprise Edition scratch orgs.

Requires configuration using the Setup menu in the scratch org.

See [Set Up Einstein Classification Apps](#) in Salesforce Help for more information.

ChangeDataCapture

Enables Change Data Capture, if the scratch org edition doesn't automatically enable it.

Chatbot

Enables deployment of Bot metadata into a scratch org, and allows you to create and edit bots.

More Information

To use this feature, turn on **Enable Einstein Features** in the Dev Hub org to accept the Terms of Service.

See [Einstein Bots](#) in Salesforce Help for more information.

ChatterEmailFooterLogo

ChatterEmailFooterLogo allows you to use the Document ID of a logo image, which you can use to customize chatter emails.

More Information

See [Add Your Custom Brand to Email Notifications](#) in Salesforce Help for more information.

ChatterEmailFooterText

ChatterEmailFooterText allows you to use footer text in customized Chatter emails.

More Information

See [Add Your Custom Brand to Email Notifications](#) in Salesforce Help for more information.

ChatterEmailSenderName

ChatterEmailSenderName allows you to customize the name that appears as the sender's name in the email notification. For example, your company's name.

More Information

See [Chatter Email Settings and Branding](#) in Salesforce Help for more information.

CloneApplication

CloneApplication allows you to clone an existing custom Lightning app and make required customizations to the new app. This way, you don't have to start from scratch, especially when you want to create apps with simple variations.

More Information

See [Create Lightning Apps](#) in Salesforce Help for more information.

CMSMaxContType

Limits the number of distinct content types you can create within Salesforce CMS to 21.

CMSMaxNodesPerContType

Limits the maximum number of child nodes (fields) you can create for a particular content type to 15.

CMSUnlimitedUse

Enables unlimited content records, content types, and bandwidth usage in Salesforce CMS.

Communities

Allows the org to create a customer community. To use Communities, you must also include communitiesSettings > enableNetworksEnabled in the settings section of your scratch org definition file.

More Information

Available in Enterprise and Developer scratch orgs.

ConAppPluginExecuteAsUser

Enables the pluginExecutionUser field in the ConnectedApp Metadata API object.

ConcStreamingClients:<value>

Increases the maximum number of concurrent clients (subscribers) across all channels and for all event types for API version 36.0 and earlier. Indicate a value between 20–4,000.

Supported Quantities

20–4,000, Multiplier: 1

ConnectedAppCustomNotifSubscription

Enables connected apps to subscribe to custom notification types, which are used to send custom desktop and mobile notifications.

More Information

Sending custom notifications requires both [CustomNotificationType](#) on page 85 to create notification types and [ConnectedAppCustomNotifSubscription](#) to subscribe to notification types. See [Manage Your Notifications with Notification Builder](#) in Salesforce Help for more information on custom notifications.

ConnectedAppToolingAPI

Enables the use of connected apps with the Tooling API.

ConsentEventStream

Enables the Consent Event Stream permission for the org.

More Information

See [Use the Consent Event Stream](#) in Salesforce Help for more information.

ConsolePersistenceInterval:<value>

Increases how often console data is saved, in minutes. Indicate a value between 0–500. To disable auto save, set the value to 0.

Supported Quantities

0–500, Multiplier: 1

ContactsToMultipleAccounts

Enables the contacts to multiple accounts feature. This feature lets you relate a contact to two or more accounts.

ContractApprovals

Enables contract approvals, which allow you to track contracts through an approval process.

CPQ

Enables the licensed features required to install the Salesforce CPQ managed package. Doesn't install the package automatically.

More Information

For additional information and configuration steps, see [Manage Your Quotes with CPQ](#) in Salesforce Help.

CustomFieldDataTranslation

Enables translation of custom field data for Work Type Group, Service Territory, and Service Resource objects. You can enable data translation for custom fields with Text, Text Area, Text Area (Long), Text Area (Rich), and URL types.

More Information

Requires that you also include the EntityTranslation scratch org feature in your scratch org definition file. Not available in Professional, Partner Professional, Group, or Partner Group Edition orgs.

CustomNotificationType

Allows the org to create custom notification types, which are used to send custom desktop and mobile notifications.

More Information

Sending custom notifications requires both CustomNotificationType to create notification types and [ConnectedAppCustomNotifSubscription](#) on page 83 to subscribe to notification types. See [Manage Your Notifications with Notification Builder](#) in Salesforce Help for more information on custom notifications.

DataComDnbAccounts

Provides a license to Data.com account features.

DataComFullClean

Provides a license to Data.com cleaning features, and allows users to turn on auto fill clean settings for jobs.

DataMaskUser

Provides 30 Data Mask permission set licenses. This permission set enables access to an installed Salesforce Data Mask package.

More Information

For additional installation and configuration steps, see [Install the Managed Package](#) in Salesforce Help.

DataProcessingEngine

Enables the Data Processing Engine license. Data Processing Engine helps transform data that's available in your Salesforce org and write back the transformation results as new or updated records.

More Information

See [Data Processing Engine](#) in Salesforce Help for more information.

DebugApex

Enables Apex Interactive Debugger. You can use it to debug Apex code by setting breakpoints and checkpoints, and inspecting your code to find bugs.

DecisionTable

Enables Decision Table license. Decision tables read business rules and decide the outcome for records in your Salesforce org or for the values that you specify.

More Information

See [Decision Table](#) in Salesforce Help for more information.

DefaultWorkflowUser

Sets the scratch org admin as the default workflow user.

DeferSharingCalc

Allows admins to suspend group membership and sharing rule calculations and to resume them later.

More Information

Requires configuration using the Setup menu in the scratch org. See [Defer Sharing Calculations](#) in Salesforce Help.

DevelopmentWave

Enables CRM Analytics development in a scratch org. It assigns five platform licenses and five CRM Analytics platform licenses to the org, along with assigning the permission set license to the admin user. It also enables the CRM Analytics Templates and Einstein Discovery features.

DeviceTrackingEnabled

Enables Device Tracking.

DevOpsCenter

Enables DevOps Center in scratch orgs so that partners can create second-generation managed packages that extend or enhance the functionality in the DevOps Center application (base) package.

Dev Hub Org

Ask a Salesforce admin to enable DevOps Center in the Dev Hub org. From Setup, enter *DevOps Center* in the Quick Find box, then select **DevOps Center**. You can create scratch orgs after the org preference is enabled.

Scratch Org Definition File

Add these options to your scratch org definition file:

```
{  
    "orgName": "Acme",  
    "edition": "Enterprise",  
    "features": ["DevOpsCenter"],  
    "settings": {  
        "devHubSettings": {  
            "enableDevOpsCenterGA": true  
        }  
    }  
}
```

Scratch Org Definition File For Scratch Orgs Created from an Org Shape

If you create a scratch org based on an org shape with DevOps Center enabled, we still require that you add the DevOps Center feature and setting to the scratch org definition for legal reasons as part of the DevOps Center terms and conditions.

```
{  
    "orgName": "Acme",  
    "sourceOrg": "00DB1230400Ifx5",  
    "features": ["DevOpsCenter"],  
    "settings": {  
        "devHubSettings": {  
            "enableDevOpsCenterGA": true  
        }  
    }  
}
```

More Information

Salesforce Help: [Build an Extension Package for DevOps Center](#)

DisableManageConfAPI

Limits access to the LoginIP and ClientBrowser API objects to allow view or delete only.

DisclosureFramework

Provides the permission set licenses and permission sets required to configure Disclosure and Compliance Hub.

Scratch Org Definition File

Add these options to your scratch org definition file:

```
{  
    "orgName": "dch org",  
    "edition": "Developer",  
    "features": ["DisclosureFramework"],  
    "settings": {  
        "industriesSettings":{  
            "enableGnrcDisclsFrmwrk": true,  
            "enableIndustriesAssessment" : true  
        }  
    }  
}
```

More Information

For configuration steps, see [Disclosure and Compliance Hub](#) in the Set Up and Maintain Net Zero Cloud guide in Salesforce Help.

Division

Turns on the Manage Divisions feature under Company Settings. Divisions let you segment your organization's data into logical sections, making searches, reports, and list views more meaningful to users. Divisions are useful for organizations with extremely large amounts of data.

DocumentChecklist

Enables Document Tracking and Approval features, and adds the Document Checklist permission set. Document tracking features let you define documents to upload and approve, which supports processes like loan applications or action plans.

More Information

See [Enable Document Tracking and Approvals](#) in the Financial Services Cloud Administrator Guide for more information.

DocumentReaderPageLimit

Limits the number of pages sent for data extraction to 5.

More Information

See [Intelligent Form Reader](#) in Salesforce Help for more information.

DSARPortability

Enables an org to access the DSARPortability feature in Privacy Center. Also, provides one seat each of the PrivacyCenter and PrivacyCenterAddOn licenses.

More Information

See [Portability](#) in the Salesforce REST API Developer Guide for more information.

DurableClassicStreamingAPI

Enables Durable PushTopic Streaming API for API version 37.0 and later.

More Information

Available in Enterprise and Developer Edition scratch orgs.

DurableGenericStreamingAPI

Enables Durable Generic Streaming API for API version 37.0 and later.

More Information

Available in Enterprise and Developer Edition scratch orgs.

DynamicClientCreationLimit

Allows the org to register up to 100 OAuth 2.0 connected apps through the dynamic client registration endpoint.

EinsteinAnalyticsPlus

Provides one CRM Analytics Plus license for the CRM Analytics platform.

EinsteinArticleRecommendations

Provides licenses for Einstein Article Recommendations. Einstein Article Recommendations uses data from past cases to identify Knowledge articles that are most likely to help your customer service agents address customer inquiries.

More Information

Available in Enterprise Edition scratch orgs.

Requires configuration using the Setup menu in the scratch org.

See [Set Up Einstein Article Recommendations](#) in Salesforce Help for more information.

EinsteinBuilderFree

Provides a license that allows admins to create one enabled prediction with Einstein Prediction Builder. Einstein Prediction Builder is custom AI for admins

More Information

For configuration steps, see [Einstein Prediction Builder](#) in Salesforce Help.

EinsteinDocReader

Provides the license required to enable and use Intelligent Form Reader in a scratch org. Intelligent Form Reader uses optical character recognition to automatically extract data with Amazon Textract.

More Information

For information about Intelligent Form Reader, see [Intelligent Form Reader](#) in Salesforce Help.

EinsteinRecommendationBuilder

Provides a license to create recommendations with Einstein Recommendation Builder. Einstein Recommendation Builder lets you build custom AI recommendations.

More Information

Enabled in Developer and Enterprise Editions.

Requires configuration using the Setup menu in the scratch org. You also need the EinsteinRecommendationBuilderMetadata feature to use Einstein Recommendation Builder in scratch org.

See [Einstein Recommendation Builder](#) in Salesforce Help for more information.

EinsteinRecommendationBuilderMetadata

Enables Einstein Recommendation Builder to use the required metadata APIs. Enabling this feature lets you build custom AI recommendations.

More Information

Enabled in Developer and Enterprise Editions.

Requires configuration using the Setup menu in the scratch org. You also need the EinsteinRecommendationBuilderMetadata feature to use the Einstein Recommendation Builder in scratch org.

See [Einstein Recommendation Builder](#) in Salesforce Help for more information.

EinsteinSearch

Provides the license required to use and enable Einstein Search features in a scratch org.

More Information

Available in Professional and Enterprise Edition scratch orgs.

Requires configuration using the Setup menu in the scratch org.

See [Manage Einstein Search Settings](#) in Salesforce Help for more information.

EinsteinVisits

Enables Consumer Goods Cloud. With Consumer Goods cloud, transform the way you collaborate with your retail channel partners. Empower your sales managers to plan visits and analyze your business's health across stores. Also, allow your field reps to track inventory, take orders, and capture visit details using the Retail Execution mobile app.

EinsteinVisitsED

Enables Einstein Discovery, which can be used to get store visit recommendations. With Einstein Visits ED, you can create a visit frequency strategy that allows Einstein to provide optimal store visit recommendations.

More Information

See [Create a Visit Frequency Next Best Action Strategy](#) in Salesforce Help.

EmbeddedLoginForIE

Provides JavaScript files that support Embedded Login in IE11.

EmpPublishRateLimit:<value>

Increases the maximum number of standard-volume platform event notifications published per hour. Indicate a value between 1,000–10,000.

Supported Quantities

1,000–10,000, Multiplier: 1

EnablePRM

Enables the partner relationship management permissions for the org.

EnableManageIdConfUI

Enables access to the LoginIP and ClientBrowser API objects to verify a user's identity in the UI.

EnableSetPasswordInApi

Enables you to use `sfdx force:user:password:generate:` to change a password without providing the old password.

EncryptionStatisticsInterval:<value>

Defines the interval (in seconds) between encryption statistics gathering processes. The maximum value is 604,800 seconds (7 days). The default is once per 86,400 seconds (24 hours).

Supported Quantities

0–60,4800, Multiplier: 1

More Information

Requires enabling [PlatformEncryption](#) and some configuration using the Setup menu in the scratch org. See [Which User Permissions Does Shield Platform Encryption Require?](#), and [Generate a Tenant Secret with Salesforce](#) in Salesforce Help.

EncryptionSyncInterval:<value>

Defines how frequently (in seconds) the org can synchronize data with the active key material. The default and maximum value is 604,800 seconds (7 days). To synchronize data more frequently, indicate a value, in seconds, equal to or larger than 0.

Supported Quantities

0–604,800, Multiplier: 1

More Information

Requires enabling [PlatformEncryption](#) and some configuration using the Setup menu in the scratch org. See [Which User Permissions Does Shield Platform Encryption Require?](#), and [Generate a Tenant Secret with Salesforce](#) in Salesforce Help.

Entitlements

Enables entitlements. Entitlements are units of customer support in Salesforce, such as phone support or web support that represent terms in service agreements.

EventLogFile

Enables API access to your org's event log files. The event log files contain information about your org's operational events that you can use to analyze usage trends and user behavior.

EntityTranslation

Enables translation of field data for Work Type Group, Service Territory, and Service Resource objects.

More Information

To translate custom field data, also include the CustomFieldDataTranslation scratch org feature in your scratch org definition file. Not available in Professional, Partner Professional, Group, or Partner Group Edition orgs.

ExpressionSetMaxExecPerHour

Enables an org to run a maximum of 500,000 expression sets per hour by using Connect REST API.

For more information, see [Expression Set](#) in Salesforce developer documentation.

ExternalIdentityLogin

Allows the scratch org to use Salesforce Customer Identity features associated with your External Identity license.

FieldAuditTrail

Enables Field Audit Trail for the org and allows a total 60 tracked fields. By default, 20 fields are tracked for all orgs, and 40 more are tracked with Field Audit Trail.

More Information

Previous name: RetainFieldHistory

FieldService:<value>

Provides the Field Service license. Indicate a value between 1–25.

Supported Quantities

1–25, Multiplier: 1

More Information

Available in Enterprise Edition. Enabled by default in Developer Edition. See [Enable Field Service](#) in Salesforce Help for more information.

FieldServiceDispatcherUser:<value>

Adds the Field Service Dispatcher permission set license. Indicate a value between 1–25.

Supported Quantities

1–25, Multiplier: 1

More Information

See [Assign Field Service Permissions](#) in Salesforce Help for more information.

FieldServiceMobileUser:<value>

Adds the Field Service Mobile permission set license. Indicate a value between 1–25.

Supported Quantities

1–25, Multiplier: 1

More Information

See [Assign Field Service Permissions](#) in Salesforce Help for more information.

FieldServiceSchedulingUser:<value>

Adds the Field Service Scheduling permission set license. Indicate a value between 1–25.

Supported Quantities

1–25, Multiplier: 1

More Information

See [Assign Field Service Permissions](#) in Salesforce Help for more information.

FinanceLogging

Adds Finance Logging objects to a scratch org. This feature is required for Finance Logging.

FinancialServicesCommunityUser:<value>

Adds the Financial Services Insurance Community permission set license, and enables access to Financial Services insurance community components and objects. Indicate a value between 1–10.

Supported Quantities

1–10, Multiplier: 1

FinancialServicesInsuranceUser:<value>

Adds the Financial Services Insurance permission set license, and enables access to Financial Services insurance components and objects. Indicate a value between 1–10.

Supported Quantities

1–10, Multiplier: 1

FinancialServicesUser:<value>

Adds the Financial Services Cloud Standard permission set license. This permission set enables access to Lightning components and the standard version of Financial Services Cloud. Also provides access to the standard Salesforce objects and custom Financial Services Cloud objects. Indicate a value between 1–10.

Supported Quantities

1–10, Multiplier: 1

FlowSites

Enables the use of flows in Salesforce Sites and customer portals.

ForceComPlatform

Adds one Salesforce Platform user license.

FSCServiceProcess

Enables the Service Process Studio feature of Financial Service Cloud. Provides 10 seats each of the IndustriesServiceExcellenceAddOn and FinancialServicesCloudStandardAddOn licenses. To enable the feature, you must also turn on the StandardServiceProcess setting in Setup and grant users the AccessToServiceProcess permission.

GenericStreaming

Enables Generic Streaming API for API version 36.0 and earlier.

More Information

Available in Enterprise and Developer Edition scratch orgs.

GenStreamingEventsPerDay:<value>

Increases the maximum number of delivered event notifications within a 24-hour period, shared by all CometD clients, with generic streaming for API version 36.0 and earlier. Indicate a value between 10,000–50,000.

Supported Quantities

10,000–50,000, Multiplier: 1

Grantmaking

Gives users access to Grantmaking features and objects in Salesforce and Experience Cloud.

More Information

See [Grantmaking](#) in Salesforce Help for more information. To enable Grantmaking, add these settings to your scratch org definition file.

```
{  
    "features": ["Grantmaking"],  
    "settings": {  
        "IndustriesSettings": {  
            "enableGrantmaking": true  
        }  
    }  
}
```

```
    }  
}  
}
```

HealthCloudAddOn

Enables use of Health Cloud.

More Information

See [Administer Health Cloud](#) in Salesforce Help for more information.

HealthCloudELOOverride

Salesforce retired the Health Cloud CandidatePatient object in Spring '22 to focus on the more robust Lead object. This scratch org feature allows you to override that retirement and access the object.

More Information

See [Candidate Patient Data Entity Retirement](#) in Salesforce Help for more information.

HealthCloudForCmty

Enables use of Health Cloud for Experience Cloud Sites.

More Information

See [Experience Cloud Sites](#) in Salesforce Help for more information.

HealthCloudMedicationReconciliation

Allows Medication Management to support Medication Reconciliation.

More Information

See [Enable Medication Management to Perform Medication Reconciliation](#) in Salesforce Help for more information.

HealthCloudPNMAddOn

Enables use of Provider Network Management.

More Information

See [Provider Network Management](#) in Salesforce Help for more information.

HealthCloudUser

This enables the scratch org to use the Health Cloud objects and features equivalent to the Health Cloud permission set license for one user.

More Information

See [Assign Health Cloud Permission Sets and Permission Set Licenses](#) in Salesforce Help for more information.

HighVelocitySales

Provides Sales Engagement licenses and enables Salesforce Inbox. Sales Engagement optimizes the inside sales process with a high-productivity workspace. Sales managers can create custom sales processes that guide reps through handling different types of prospects. And sales reps can rapidly handle prospects with a prioritized list and other productivity-boosting features. The Sales Engagement feature can be deployed in scratch orgs, but the settings for the feature can't be updated through the scratch org definition file. Instead, configure settings directly in the Sales Engagement app.

HoursBetweenCoverageJob:<value>

The frequency in hours when the sharing inheritance coverage report can be run for an object. Indicate a value between 1–24.

Supported Quantities

1–24, Multiplier: 1

IdentityProvisioningFeatures

Enables use of Salesforce Identity User Provisioning.

IndustriesActionPlan

Provides a license for Action Plans. Action Plans allow you to define the tasks or document checklist items for completing a business process.

More Information

Previous name: ActionPlan.

For more information and configuration steps, see [Enable Actions Plans](#) in Salesforce Help.

IndustriesMfgTargets

Enables Sales Agreements. With Sales Agreements, you can negotiate purchase and sale of products over a continued period. You can also get insights into products, prices, discounts, and quantities. And you can track your planned and actual quantities and revenues with real-time updates from orders and contracts.

More Information

See [Track Sales Compliance with Sales Agreements](#) in Salesforce Help for more information.

IndustriesManufacturingCmt

Provides the Manufacturing Sales Agreement for the Community permission set license, which is intended for the usage of partner community users. It also provides access to the Manufacturing community template for admins users to create communities.

More Information

See [Improve Partner Collaboration with Communities](#) in Salesforce Help for more information.

IndustriesMfgAccountForecast

Enables Account Forecast. With Account Forecast, you can generate forecasts for your accounts based on orders, opportunities, and sales agreements. You can also create formulas to calculate your forecasts per the requirements of your company.

More Information

See [Create Account Forecasts to Enhance Your Planning](#) in Salesforce Help for more information.

InsightsPlatform

Enables the CRM Analytics Plus license for CRM Analytics.

InsuranceCalculationUser

Enables the calculation feature of Insurance. Provides 10 seats each of the BRERuntimeAddOn and OmniStudioRuntime licenses. Also, provides one seat each of the OmniStudio and BREPlatformAccess licenses.

InsuranceClaimMgmt

Enables claim management features. Provides one seat of the InsuranceClaimMgmtAddOn license.

More Information

See [Manage Claims](#) in Salesforce Help for more information.

InsurancePolicyAdmin

Enables policy administration features. Provides one seat of the InsurancePolicyAdministrationAddOn license.

More Information

See [Manage Insurance Policies](#) in Salesforce Help for more information.

IntelligentDocumentReader

Provides the license required to enable and use Intelligent Document Reader in a scratch org. Intelligent Document Reader uses optical character recognition to automatically extract data with Amazon Textract by using your AWS account.

More Information

For information about Intelligent Document Reader, see [Intelligent Document Reader](#) in Salesforce Help.

Interaction

Enables flows. A flow is the part of Salesforce Flow that collects data and performs actions in your Salesforce org or an external system. Salesforce Flow provides two types of flows: screen flows and autolaunched flows.

More Information

Requires configuration in the Setup menu of the scratch org.

IoT

Enables IoT so the scratch org can consume platform events to perform business and service workflows using orchestrations and contexts.

More Information

Requires configuration in the Setup menu of the scratch org.

JigsawUser

Provides one license to Jigsaw features.

Knowledge

Enables Salesforce Knowledge and gives your website visitors, clients, partners, and service agents the ultimate support tool. Create and manage a knowledge base with your company information, and securely share it when and where it's needed. Build a knowledge base of articles that can include information on process, like how to reset your product to its defaults, or frequently asked questions.

More Information

See [Salesforce Knowledge](#) in Salesforce Help for more information.

LegacyLiveAgentRouting

Enables legacy Live Agent routing for Chat. Use Live Agent routing to chat in Salesforce Classic. Chats in Lightning Experience must be routed using Omni-Channel.

LightningSalesConsole

Adds one Lightning Sales Console user license.

LightningScheduler

Enables Lightning Scheduler. Lightning Scheduler gives you tools to simplify appointment scheduling in Salesforce. Create a personalized experience by scheduling customer appointments—in person, by phone, or by video—with the right person at the right place and time.

More Information

See [Manage Appointments with Lightning Scheduler](#) in Salesforce Help for more information.

LightningServiceConsole

Assigns the Lightning Service Console License to your scratch org so you can use the Lightning Service Console and access features that help manage cases faster.

More Information

See [Lightning Service Console](#) in Salesforce Help for more information.

LiveAgent

Enables Chat for Service Cloud. Use web-based chat to quickly connect customers to agents for real-time support.

LiveMessage

Enables Messaging for Service Cloud. Use Messaging to quickly support customers using apps such as SMS text messaging and Facebook Messenger.

LongLayoutSectionTitles

Allows page layout section titles to be up to 80 characters.

More Information

To turn on this feature, contact Salesforce Customer Support.

LoyaltyAnalytics

Enables Analytics for Loyalty license. The Analytics for Loyalty app gives you actionable insights into your loyalty programs.

More Information

See [Analytics for Loyalty](#) in Salesforce Help for more information.

LoyaltyEngine

Enables Loyalty Management Promotion Setup license. Promotion setup allows loyalty program managers to create loyalty program processes. Loyalty program processes help you decide how incoming and new Accrual and Redemption-type transactions are processed.

More Information

See [Create Processes with Promotion Setup](#) in Salesforce Help for more information.

LoyaltyManagementStarter

Enables the Loyalty Management - Starter license. Create loyalty programs and set up loyalty program-specific processes that allow you to recognize, rewards, and retain customers.

More Information

See [Loyalty Management](#) in Salesforce Help for more information.

LoyaltyMaximumPartners:<value>

Increases the number of loyalty program partners that can be associated with a loyalty program in an org where the Loyalty Management - Starter license is enabled. The default and maximum value is 1.

Supported Quantities

0–1, Multiplier: 1

LoyaltyMaximumPrograms:<value>

Increases the number of loyalty programs that can be created in an org where the Loyalty Management - Starter license is enabled. The default and maximum value is 1.

Supported Quantities

0–1, Multiplier: 1

LoyaltyMaxOrderLinePerHour:<value>

Increases the number of order lines that can be cumulatively processed per hour by loyalty program processes. Indicate a value between 1–3,500,000.

Supported Quantities

1–3,500,000, Multiplier: 1

LoyaltyMaxProcExecPerHour:<value>

Increases the number of transaction journals that can be processed by loyalty program processes per hour. Indicate a value between 1–500,000.

Supported Quantities

1–500,000, Multiplier: 1

LoyaltyMaxTransactions:<value>

Increases the number of Transaction Journal records that can be processed. Indicate a value between 1–50,000,000.

Supported Quantities

1–50,000,000, Multiplier: 1

LoyaltyMaxTrxnJournals:<value>

Increases the number of Transaction Journal records that can be stored in an org that has the Loyalty Management - Start license enabled.

Supported Quantities

1–25,000,000, Multiplier: 1

More Information

See [Transaction Journal Limits](#) in Salesforce Help for more information.

Macros

Enables macros in your scratch org. After enabling macros, add the macro browser to the Lightning Console so you can configure predefined instructions for commonly used actions and apply them to multiple posts at the same time.

More Information

See [Set Up and Use Macros](#) in Salesforce Help for more information.

MarketingUser

Provides access to the Campaigns object. Without this setting, Campaigns are read-only.

MaxActiveDPEDefs:<value>

Increases the number of Data Processing Engine definitions that can be activated in the org. Indicate a value between 1–50.

Supported Quantities

1–50, Multiplier: 1

MaxApexCodeSize:<value>

Limits the non-test, unmanaged Apex code size (in MB). To use a value greater than the default value of 10, contact Salesforce Customer Support.

MaxAudTypeCriterionPerAud

Limits the number of audience type criteria available per audience. The default value is 10.

MaxCustomLabels:<value>

Limits the number of custom labels (measured in thousands). Setting the limit to 10 enables the scratch org to have 10,000 custom labels. Indicate a value between 1–15.

Supported Quantities

1–15, Multiplier: 1,000

MaxDatasetLinksPerDT:<value>

Increases the number of dataset links that can be associated with a decision table. Indicate a value between 1–3.

Supported Quantities

1–3, Multiplier: 1

MaxDataSourcesPerDPE:<value>

Increases the number of Source Object nodes a Data Processing Engine definition can contain. Indicate a value between 1–50.

Supported Quantities

1–50, Multiplier: 1

MaxDecisionTableAllowed:<value>

Increases the number of decision tables rules that can be created in the org. Indicate a value between 1–30.

Supported Quantities

1–30, Multiplier: 1

MaxFavoritesAllowed:<value>

Increases the number of Favorites allowed. Favorites allow users to create a shortcut to a Salesforce Page. Users can view their Favorites by clicking the Favorites list dropdown in the header. Indicate a value between 0–200.

Supported Quantities

0–200, Multiplier: 1

MaxFieldsPerNode:<value>

Increases the number of fields a node in a Data Processing Engine definition can contain. Indicate a value between 1–500.

Supported Quantities

1–500, Multiplier: 1

MaxInputColumnsPerDT:<value>

Increases the number of input fields a decision table can contain. Indicate a value between 1–10.

Supported Quantities

1–10, Multiplier: 1

MaxLoyaltyProcessRules:<value>

Increases the number of loyalty program process rules that can be created in the org. Indicate a value between 1–20.

Supported Quantities

1–20, Multiplier: 1

MaxNodesPerDPE:<value>

Increases the number of nodes that a Data Processing Engine definition can contain. Indicate a value between 1–500.

Supported Quantities

1–500, Multiplier: 1

MaxNoOfLexThemesAllowed:<value>

Increases the number of Themes allowed. Themes allow users to configure colors, fonts, images, sizes, and more. Access the list of Themes in Setup, under Themes and Branding. Indicate a value between 0–300.

Supported Quantities

0–300, Multiplier: 1

MaxOutputColumnsPerDT:<value>

Increases the number of output fields a decision table can contain. Indicate a value between 1–5.

Supported Quantities

1–5, Multiplier: 1

MaxSourceObjectPerDSL:<value>

Increases the number of source objects that can be selected in a dataset link of a decision table. Indicate a value between 1–5.

Supported Quantities

1–5, Multiplier: 1

MaxStreamingTopics:<value>

Increases the maximum number of delivered PushTopic event notifications within a 24-hour period, shared by all CometD clients. Indicate a value between 40–100.

Supported Quantities

40–100, Multiplier: 1

MaxUserNavItemsAllowed:<value>

Increases the number of navigation items a user can add to the navigation bar. Indicate a value between 0–500.

Supported Quantities

0–500, Multiplier: 1

MaxUserStreamingChannels:<value>

Increases the maximum number of user-defined channels for generic streaming. Indicate a value between 20–1,000.

Supported Quantities

20–1,000, Multiplier: 1

MaxWritebacksPerDPE:<value>

Increases the number of Writeback Object nodes a Data Processing Engine definition can contain. Indicate a value between 1–50.

Supported Quantities

1–10, Multiplier: 1

MedVisDescriptorLimit:<value>

Increases the number of sharing definitions allowed per record for sharing inheritance to be applied to an object. Indicate a value between 150–1,600.

Supported Quantities

150–1,600, Multiplier: 1

MinKeyRotationInterval

Sets the encryption key material rotation interval at once per 60 seconds. If this feature isn't specified, the rotation interval defaults to once per 604,800 seconds (7 days) for Search Index key material, and once per 86,400 seconds (24 hours) for all other key material.

More Information

Requires enabling [PlatformEncryption](#) and some configuration using the Setup menu in the scratch org. See [Which User Permissions Does Shield Platform Encryption Require?](#) and [Generate a Tenant Secret with Salesforce](#) in Salesforce Help.

MobileExtMaxFileSizeMB:<value>

Increases the file size (in megabytes) for Field Service Mobile extensions. Indicate a value between 1–2,000.

Supported Quantities

1–2,000, Multiplier: 1

MobileSecurity

Enables Enhanced Mobile Security. With Enhanced Mobile Security, you can control a range of policies to create a security solution tailored to your org's needs. You can limit user access based on operating system versions, app versions, and device and network security. You can also specify the severity of a violation.

MultiCurrency

Enables the scratch org to set up and use multiple currencies in opportunities, forecasts, quotes, reports, and other data.

More Information

See [Considerations for Enabling Multiple Currencies](#) in Salesforce Help.

MultiLevelMasterDetail

Allows the creation a special type of parent-child relationship between one object, the child, or detail, and another object, the parent, or master.

MutualAuthentication

Requires client certificates to verify inbound requests for mutual authentication.

MyTrailhead

Enables access to a myTrailhead enablement site in a scratch org.

Scratch Org Definition File

Add these options to your scratch org definition file:

```
{  
    "orgName": "Acme",  
    "edition": "Enterprise",  
    "features": ["MyTrailhead"],  
    "settings": {  
        "trailheadSettings": {  
            "enableMyTrailheadPref": true  
        }  
    }  
}
```

More Information

Salesforce Help: [Enablement Sites \(myTrailhead\)](#)

NonprofitCloudCaseManagementUser

Provides the permission set license required to use and configure the Salesforce.org Nonprofit Cloud Case Management managed package. You can then install the package in the scratch org.

More Information

For installation and configuration steps, see [Salesforce.org Nonprofit Cloud Case Management](#).

NumPlatformEvents:<value>

Increases the maximum number of platform event definitions that can be created. Indicate a value between 5–20.

Supported Quantities

5–20, Multiplier: 1

ObjectLinking

Create rules to quickly link channel interactions to objects such as contacts, leads, or person accounts for customers (Beta).

OrderManagement

Provides the Salesforce Order Management license. Order Management is your central hub for handling all aspects of the order lifecycle, including order capture, fulfillment, shipping, payment processing, and servicing.

More Information

Available in Enterprise and Developer Edition scratch orgs.

If you want to configure Order Management to use any of these features, enable it in your scratch org:

- MultiCurrency
- PersonAccounts
- ProcessBuilder
- StateAndCountryPicklist

Requires configuration using the Setup menu in the scratch org. For installation and configuration steps, see *Salesforce Help: Salesforce Order Management*.

 **Note:** The implementation process includes turning on several Order and Order Management feature toggles in Setup. In a scratch org, you can turn them on by including metadata settings in your scratch org definition file. For details about these settings, see [OrderSettings](#) and [OrderManagementSettings](#) in the *Metadata API Developer Guide*.

OrderSaveLogicEnabled

Enables scratch org support for New Order Save Behavior.

More Information

OrderSaveLogicEnabled supports only New Order Save Behavior. If your scratch org needs both Old and New Order Save Behavior, use OrderSaveBehaviorBoth.

To enable OrderSaveLogicEnabled, update your scratch org definitions file.

```
{  
  "features": ["OrderSaveLogicEnabled"],  
  "settings": {  
    "orderSettings": {  
      "enableOrders": true  
    }  
  }  
}
```

OrderSaveBehaviorBoth

Enables scratch org support for both New Order Save Behavior and Old Order Save Behavior.

More Information

To enable OrderSaveLogicEnabled, update your scratch org definitions file.

```
{  
  "features": ["OrderSaveBehaviorBoth"],  
  "settings": {  
    "orderSettings": {  
      "enableOrders": true  
    }  
  }  
}
```

```
}
```

OutboundMessageHTTPSession

Enables using HTTP endpoint URLs in outbound message definitions that have the Send Session ID option selected.

PardotScFeaturesCampaignInfluence

Enables additional campaign influence models, first touch, last touch, and even distribution for Pardot users.

PersonAccounts

Enables person accounts in your scratch org.

More Information

Available in Enterprise and Developer Edition scratch orgs.

PipelineInspection

Enables the Pipeline Inspection. Pipeline Inspection is a consolidated pipeline view with metrics, opportunities, and highlights of recent changes.

More Information

Requires enabling PipelineInspection and some configuration using the Setup menu in the scratch org. See [Turn On Pipeline Inspection](#) in Salesforce Help for more information.

Editions

Available in: Enterprise Edition scratch orgs.

PlatformCache

Enables Platform Cache and allocates a 3 MB cache. The Lightning Platform Cache layer provides faster performance and better reliability when caching Salesforce session and org data.

More Information

See [Platform Cache](#) in the Apex Developer Guide for more information.

PlatformConnect:<value>

Enables Salesforce Connect and allows your users to view, search, and modify data that's stored outside your Salesforce org. Indicate a value from 1–5.

Supported Quantities

1–5, Multiplier: 1

PlatformEncryption

Shield Platform Encryption encrypts data at rest. You can manage key material and encrypt fields, files, and other data.

PlatformEventsPerDay:<value>

Increases the maximum number of delivered standard-volume platform event notifications within a 24-hour period, shared by all CometD clients. Indicate a value between 10,000–50,000.

Supported Quantities

10,000–50,000, Multiplier: 1

ProcessBuilder

Enables Process Builder, a Salesforce Flow tool that helps you automate your business processes.

More Information

Requires configuration in the Setup menu of the scratch org.

See [Process Builder](#) in Salesforce Help for more information.

ProductsAndSchedules

Enables product schedules in your scratch org. Enabling this feature lets you create default product schedules on products. Users can also create schedules for individual products on opportunities.

ProgramManagement

Enables access to all Program Management and Case Management features and objects.

More Information

To enable ProgramManagement, add these settings to your scratch org definition file.

```
{
  "orgName": "Sample Org",
  "edition": "Enterprise",
  "features": ["ProgramManagement"],
  "settings": {
    "IndustriesSettings": {
      "enableBenefitManagementPreference": true,
      "enableBenefitAndGoalSharingPref": true,
      "enableCarePlansPreference": true
    }
  }
}
```

Alternatively, enable the settings in the org manually. See [Enable Program Management](#) in Salesforce Help.

ProviderFreePlatformCache

Provides 3 MB of free Platform Cache capacity for AppExchange-certified and security-reviewed managed packages. This feature is made available through a capacity type called Provider Free capacity and is automatically enabled in Developer Edition orgs. Allocate the Provider Free capacity to a Platform Cache partition and add it to your managed package.

More Information

See [Set Up a Platform Cache Partition with Provider Free Capacity](#) in Salesforce Help for more information.

PublicSectorAccess

Enables access to all Public Sector features and objects.

PublicSectorApplicationUsageCreditsAddOn

Enables additional usage of Public Sector applications based on their pricing.

PublicSectorSiteTemplate

Allows Public Sector users access to build an Experience Cloud site from the templates available.

RecordTypes

Enables Record Type functionality. Record Types let you offer different business processes, picklist values, and page layouts to different users.

RefreshOnInvalidSession

Enables automatic refreshes of Lightning pages when the user's session is invalid. If, however, the page detects a new token, it tries to set that token and continue without a refresh.

RevSubscriptionManagement

Enables Subscription Management. Subscription Management is an API-first, product-to-cash solution for B2B subscriptions and one-time sales.

More Information

Available in Enterprise and Developer scratch orgs. To enable Subscription Management in your scratch org, add this setting in your scratch org definition file.

```
"settings": {  
    ...  
    "subscriptionManagementSettings": {  
        "enableSubscriptionManagement": true  
    },  
    ...  
}
```

For more information about Subscription Management, see <https://developer.salesforce.com/docs/revenue/subscription-management/overview>.

S1ClientComponentCacheSize

Allows the org to have up to 5 pages of caching for Lightning Components.

SalesCloudEinstein

Enables Sales Cloud Einstein features and Salesforce Inbox. Sales Cloud Einstein brings AI to every step of the sales process.

More Information

Available in Enterprise Edition scratch orgs.

See [Sales Cloud Einstein](#) in Salesforce Help for more information.

SalesforceContentUser

Enables access to Salesforce content features.

SalesforceFeedbackManagementStarter

Provides a license to use the Salesforce Feedback Management - Starter features.

More Information

Available in Enterprise and Developer edition scratch orgs. To use the Salesforce Feedback Management - Starter features, enable Surveys and assign the Salesforce Advanced Features Starter user permission to the scratch org user. For additional information on how to enable Surveys and configuration steps, see [Enable Surveys and Configure Survey Settings](#) and [Assign User Permissions](#) in Salesforce Help.

SalesforceIdentityForCommunities

Adds Salesforce Identity components, including login and self-registration, to Experience Builder. This feature is required for Aura components.

SalesUser

Provides a license for Sales Cloud features.

SAML20SingleLogout

Enables usage of SAML 2.0 single logout.

SCIMProtocol

Enables access support for the SCIM protocol base API.

SecurityEventEnabled

Enables access to security events in Event Monitoring.

SentimentInsightsFeature

Provides the license required to enable and use Sentiment Insights in a scratch org. Use Sentiment Insights to analyze the sentiment of your customers and get actionable insights to improve it.

More Information

For information about Sentiment Insights, see [Sentiment Insights](#) in Salesforce Help.

ServiceCatalog

Enables Employee Service Catalog so you can create a catalog of products and services for your employees. It can also turn your employees' requests for these products and services into approved and documented orders.

More Information

To learn more, see [Employee Service Catalog](#).

ServiceCloud

Assigns the Service Cloud license to your scratch org, so you can choose how your customers can reach you, such as by email, phone, social media, online communities, chat, and text.

ServiceCloudVoicePartnerTelephony

Assigns the Service Cloud Voice with Partner Telephony add-on license to your scratch org, so you can set up a Service Cloud Voice contact center that integrates with supported telephony providers. Indicate a value from 1–50.

Supported Quantities

1–50, Multiplier: 1

More Information

For setup and configuration steps, see [Service Cloud Voice with Partner Telephony](#) in Salesforce Help.

ServiceUser

Adds one Service Cloud User license, and allows access to Service Cloud features.

SessionIdInLogEnabled

Enables Apex debug logs to include session IDs. If disabled, session IDs are replaced with "SESSION_ID_REMOVED" in debug logs.

SFDOInsightsDataIntegrityUser

Provides a license to Salesforce.org Insights Platform Data Integrity managed package. You can then install the package in the scratch org.

More Information

For installation and configuration steps, see the [Salesforce.org Insights Platform Data Integrity](#) help.

SharedActivities

Allow users to relate multiple contacts to tasks and events.

More Information

For additional installation and configuration steps, see [Considerations for Enabling Shared Activities](#) in Salesforce Help.

Sites

Enables Salesforce Sites, which allows you to create public websites and applications that are directly integrated with your Salesforce org. Users aren't required to log in with a username and password.

More Information

You can create sites and communities in a scratch org, but custom domains, such as www.example.com, aren't supported.

SocialCustomerService

Enables Social Customer Service, sets post defaults, and either activates the Starter Pack or signs into your Social Studio account.

StateAndCountryPicklist

Enables state and country/territory picklists. State and country/territory picklists let users select states and countries from predefined, standardized lists, instead of entering state, country, and territory data into text fields.

StreamingAPI

Enables Streaming API.

More Information

Available in Enterprise and Developer Edition scratch orgs.

StreamingEventsPerDay:<value>

Increases the maximum number of delivered PushTopic event notifications within a 24-hour period, shared by all CometD clients (API version 36.0 and earlier). Indicate a value between 10,000–50,000.

Supported Quantities

10,000–50,000, Multiplier: 1

SubPerStreamingChannel:<value>

Increases the maximum number of concurrent clients (subscribers) per generic streaming channel (API version 36.0 and earlier). Indicate a value between 20–4,000.

Supported Quantities

20–4,000, Multiplier: 1

SubPerStreamingTopic:<value>

Increases the maximum number of concurrent clients (subscribers) per PushTopic streaming channel (API version 36.0 and earlier). Indicate a value between 20–4,000.

Supported Quantities

20–4,000, Multiplier: 1

SurveyAdvancedFeatures

Enables a license for the features available with the Salesforce Feedback Management - Growth license.

More Information

Available in Enterprise and Developer edition scratch orgs. To use the Salesforce Feedback Management - Growth features, enable Surveys and assign the Salesforce Surveys Advanced Features user permission to the scratch org user. For additional information on

how to enable Surveys and configuration steps, see [Enable Surveys and Configure Survey Settings](#) and [Assign User Permissions](#) in Salesforce Help.

SustainabilityCloud

Provides the permission set licenses and permission sets required to install and configure Sustainability Cloud. To enable or use CRM Analytics and CRM Analytics templates, include the DevelopmentWave scratch org feature.

More Information

For installation and configuration steps, see [Sustainability Cloud Legacy Documentation](#) in the Set Up and Maintain Net Zero Cloud guide in Salesforce Help.

SustainabilityApp

Provides the permission set licenses and permission sets required to configure Net Zero Cloud. To enable or use Tableau CRM and Tableau CRM templates, include the DevelopmentWave scratch org feature.

Scratch Org Definition File

Add these options to your scratch org definition file:

```
{  
    "orgName": "net zero scratch org",  
    "edition": "Developer",  
    "features": ["SustainabilityApp"],  
    "settings": {  
        "industriesSettings": {  
            "enableSustainabilityCloud": true,  
            "enableSCCarbonAccounting" : true  
        }  
    }  
}
```

More Information

For configuration steps, see [Configure Net Zero Cloud](#) in the Set Up and Maintain Net Zero Cloud guide in Salesforce Help.

TCRMforSustainability

Enables all permissions required to manage the Net Zero Analytics app by enabling Tableau CRM. You can create and share the analytics app for your users to bring your environmental accounting in line with your financial accounting.

More Information

For more information, see [Deploy Net Zero Analytics](#) in the Set Up and Maintain Net Zero Cloud guide in Salesforce Help.

TimelineConditionsLimit

Limits the number of timeline record display conditions per event type to 3.

More Information

See [Provide Holistic Patient Care with Enhanced Timeline](#) in Salesforce Help for more information.

TimelineEventLimit

Limits the number of event types displayed on a timeline to 5.

More Information

See [Provide Holistic Patient Care with Enhanced Timeline](#) in Salesforce Help for more information.

TimelineRecordTypeLimit

Limits the number of related object record types per event type to 3.

More Information

See [Provide Holistic Patient Care with Enhanced Timeline](#) in Salesforce Help for more information.

TimeSheetTemplateSettings

Time Sheet Templates let you configure settings to create time sheets automatically. For example, you can create a template that sets start and end dates. Assign templates to user profiles so that time sheets are created for the right users.

More Information

For configuration steps, see [Create Time Sheet Templates](#) in Salesforce Help.

TransactionFinalizers

Enables you to implement and attach Apex Finalizers to Queueable Apex jobs.

More Information



Note: This functionality is currently in open pilot and subject to restrictions.

See the [Transaction Finalizers \(Pilot\)](#) in Apex Developer Guide for more information.

WaveMaxCurrency

Increases the maximum number of supported currencies for CRM Analytics. Indicate a value between 1–5.

WavePlatform

Enables the Wave Platform license.

Workflow

Enables Workflow so you can automate standard internal procedures and processes.

More Information

Requires configuration in the Setup menu of the scratch org.

WorkflowFlowActionFeature

Allows you to launch a flow from a workflow action.

More Information

This setting is supported only if you enabled the pilot program in your org for flow trigger workflow actions. If you enabled the pilot, you can continue to create and edit flow trigger workflow actions.

If you didn't enable the pilot, use the Flows action in the ProcessBuilder scratch org feature instead.

WorkplaceCommandCenterUser

Enables access to Workplace Command Center features including access to objects such as Employee, Crisis, and EmployeeCrisisAssessment.

More Information

For additional installation and configuration steps, see [Set Up Your Work.com Development Org](#) in the *Workplace Command Center for Work.com Developer Guide*.

WorkThanksPref

Enables the give thanks feature in Chatter.

Scratch Org Settings

In Winter '19 and later, scratch org settings are the format for defining org preferences in the scratch org definition. Because you can use all Metadata API settings, they're the most comprehensive way to configure a scratch org. If a setting is supported in Metadata API, it's supported in scratch orgs. Settings provide you with fine-grained control because you can define values for all fields for a setting, rather than just enabling or disabling it.

For information on Metadata API settings and their supported fields, see [Settings](#) in *Metadata API Developer Guide*.

! **Important:** Although the Settings are upper camel case in the Metadata API Developer Guide, be sure to indicate them as lower camel case in the scratch org definition.

```
{
    "orgName": "Acme",
    "edition": "Enterprise",
    "features": ["Communities", "ServiceCloud", "Chatbot"],
    "settings": {
        "communitiesSettings": {
            "enableNetworksEnabled": true
        },
        "lightningExperienceSettings": {
            "enableS1DesktopEnabled": true
        },
        "mobileSettings": {
            "enableS1EncryptedStoragePref2": true
        },
        "omniChannelSettings": {
            "enableOmniChannel": true
        },
        "caseSettings": {
            "systemUserEmail": "support@acme.com"
        }
    }
}
```

Here's an example of how to configure SecuritySettings in your scratch org. In this case, to define session timeout, you nest the field values.

```
{
  "orgName": "Acme",
  "edition": "Enterprise",
  "features": [],
  "settings": {
    "mobileSettings": {
      "enableS1EncryptedStoragePref2": true
    },
    "securitySettings": {
      "sessionSettings": {
        "sessionTimeout": "TwelveHours"
      }
    }
  }
}
```

Here's an example of how to configure the IoT feature in your scratch org. It requires a combination of indicating the IoT feature and IoT scratch org settings.

```
{
  "orgName": "Acme",
  "edition": "Enterprise",
  "features": ["IoT"],
  "settings": {
    "mobileSettings": {
      "enableS1EncryptedStoragePref2": true
    },
    "iotSettings": {
      "enableIoT": true,
      "iotInsights": true
    }
  }
}
```

Create a Scratch Org Based on an Org Shape

We know it's not easy to build a scratch org definition that mirrors the features and settings in your production org. With Org Shape for Scratch Orgs, you can leave building the scratch org definition to us. After you capture the org's shape, you can spin up scratch orgs based on it.

Available in: Developer, Group, Professional, Unlimited, and Enterprise editions. The scratch org created from the org shape is the same edition as the source org.

Not available in: Scratch orgs and sandboxes

What's Included in Org Shape?

Features, Metadata API settings, edition, limits, and licenses determine what we refer to as an org's shape. For further clarification, org shape includes:

- Metadata API settings with boolean fields.

- Licenses associated with installed packages, but not the packages themselves. To use the associated package, install it in the scratch org created from the org shape.

What's Not Included in Org Shape?

- Metadata API settings with `integer` or `string` fields. However, you can manually add non-Boolean settings or other settings not included in the source org to your scratch org definition. See [Scratch Org Definition for Org Shape](#) for examples.
- Metadata types
- Data

Org Shapes Are Specific to a Release

Scratch org shapes are associated with a specific Salesforce release. Be sure to recreate the org shape after the source org is upgraded to the new Salesforce release. During a Salesforce major release transition, your Dev Hub org and source org can be on different release versions. See [Scratch Org Definition for Org Shape](#) for options during the transition period.

[Enable Org Shape for Scratch Orgs](#)

Enable Org Shape for Scratch Orgs in the org whose shape you want to capture (source org).

[Org Shape Permissions](#)

A Salesforce admin for the Dev Hub org must assign permissions to users who plan to create org shapes, or create scratch orgs based on an org shape. If you already have a permission set for Salesforce DX users, you can update it to include access.

[Create and Manage Org Shapes](#)

Create an org shape to mimic the baseline setup (features, limits, edition, and Metadata API settings) of a source org without the extraneous data and metadata. If the features, settings, or licenses of that org change, you can capture those updates by recreating the org shape. You can have only one active org shape at a time.

[Scratch Org Definition for Org Shape](#)

During org shape creation, we capture the features, settings, edition, licenses, and limits of the specified source org. This way, you don't have to manually include these items in the scratch org definition file. You can create a scratch org based solely on the source org shape. Or you can add more features and settings in the scratch org definition file to include functionality not present in the source org.

[Troubleshooting for Org Shape](#)

Here are some issues you can encounter when using Org Shape for Scratch Orgs.

SEE ALSO:

[Metadata API Developer Guide: Settings](#)

Enable Org Shape for Scratch Orgs

Enable Org Shape for Scratch Orgs in the org whose shape you want to capture (source org).

Available in: Developer, Group, Professional, Unlimited, and Enterprise editions

Not available in: Scratch orgs and sandboxes

Be sure to:

- Enable Org Shape for Scratch Orgs in both the source org and the Dev Hub org, if you want to capture the shape of an org that isn't also your Dev Hub org.
 - When entering the org ID, use only the first 15 characters rather than the full 18-character org ID.
- Enable Org Shape for Scratch Orgs in the Dev Hub org that you use to create scratch orgs. Contact a Salesforce admin if you require assistance.
 - From Setup, enter *Org Shape* in the Quick Find box, then select **Org Shape**.
 - Click the toggle for **Enable Org Shape for Scratch Orgs**.
 - In the text box, enter the 15-character org ID for the Dev Hub, then click **Save**.
 - (Optional) If the source org is different from the Dev Hub org, enable Org Shape for Scratch Orgs in the source org.
 - Log in to the source org.
 - From Setup, enter *Org Shape* in the Quick Find box, then select **Org Shape**.
 - Click the toggle for **Enable Org Shape for Scratch Orgs**.
 - Enter the 15-character Dev Hub org ID that you're using to create scratch orgs.

You can specify up to 50 Dev Hub org IDs to address these common use cases:

- You have multiple production orgs but your development team has access to only one. For the customization they're building, they require the shape of another production org.
- Your developers use their own Dev Hub orgs and don't have access to the production org. However, they want to create scratch orgs based on the shape of the production org.
- You're an ISV who uses your production org to create scratch orgs. You want to capture the shape of your first-generation packaging org so you can build second-generation packages.

Org Shape Permissions

A Salesforce admin for the Dev Hub org must assign permissions to users who plan to create org shapes, or create scratch orgs based on an org shape. If you already have a permission set for Salesforce DX users, you can update it to include access.

Access	Permissions
Create an org shape	Object Settings > Shape Representation > Create, Edit
Delete an org shape	Object Settings > Shape Representation > Delete
Use an org shape to create a scratch org	No additional permissions are required besides the ones for creating scratch orgs.

You don't require the "Modify All" permission to delete shapes created by others because there can be only one active shape in the org at a time.

Supported Licenses

In addition to providing users with appropriate permissions, be sure to assign the Salesforce license to Org Shape users. Other user licenses aren't supported at this time.

SEE ALSO:

[Add Salesforce DX Users](#)

[SOAP API Developer Guide: ShapeRepresentation](#)

Create and Manage Org Shapes

Create an org shape to mimic the baseline setup (features, limits, edition, and Metadata API settings) of a source org without the extraneous data and metadata. If the features, settings, or licenses of that org change, you can capture those updates by recreating the org shape. You can have only one active org shape at a time.

An org shape captures Metadata API settings, not all metadata types. For example, customizations that appear in the org, such as Lightning Experience Themes, aren't included as part of org shape. See [Settings](#) in the *Metadata API Guide* for the complete list.

An org shape includes org preference and permissions. It doesn't include data entries such as [AddressSettings](#).

! **Important:** Scratch org shapes are associated with a specific Salesforce release. Be sure to recreate the org shape after the source org is upgraded to the new Salesforce release.

1. Authorize both your Dev Hub org and the source org. Run this command for each org.

```
sfdx auth:web:login -a <alias>
```

2. Create the org shape for the source org. This command kicks off an asynchronous process to create the org shape.

```
sfdx force:org:shape:create -u <source org username/alias>
Successfully created org shape for 3SRB0000000TXbnOCG.
```

3. Check the status of the `shape:create` command.

```
sfdx force:org:shape:list
```

== Org Shapes					
ALIAS	USERNAME	ORG ID	SHAPE	STATUS	CREATED BY
					CREATED DATE
SrcOrg	me@my.org	00DB1230000Ifx5MAC	InProgress	me@my.org	2020-08-06

You can use the org shape after the status is Active:

== Org Shapes					
ALIAS	USERNAME	ORG ID	SHAPE	STATUS	CREATED BY
					CREATED DATE
SrcOrg	me@my.org	00DB1230000Ifx5MAC	Active	me@my.org	2020-08-06

Delete Org Shapes

If you run the `shape:create` command again for this org, the previous shape is marked inactive and replaced by a new active shape. If you don't want to create scratch orgs based on this shape, you can delete the org shape.

```
sfdx force:org:shape:delete -u <username/alias>
```

Scratch Org Definition for Org Shape

During org shape creation, we capture the features, settings, edition, licenses, and limits of the specified source org. This way, you don't have to manually include these items in the scratch org definition file. You can create a scratch org based solely on the source org shape. Or you can add more features and settings in the scratch org definition file to include functionality not present in the source org.

! **Important:** In the scratch org definition, indicate the 15-character `sourceOrg` instead of `edition`. The `sourceOrg` is the org ID for the org whose shape you created. Use only the first 15 characters rather than the full 18-character org ID.

Simple Scratch Org Definition File

If your Dev Hub org, source org, and org shape are all on the same Salesforce version, you can use the simple scratch org definition.

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

Scratch Org Definition File during Salesforce Release Transitions

During the Salesforce major release transition, your Dev Hub org and source org can be on different versions. If your Dev Hub org is on a different version than your source org, add the `release` option to your scratch org definition file to create scratch orgs using the org shape.

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

Source Org/Org Shape Version	Dev Hub Version	Supported Scratch Org Version	Release Option to Use in Scratch Org Definition File
Current	Preview	Current version only	<code>"release": "previous"</code>
Preview	Current	Preview version only	<code>"release": "preview"</code>

Scratch Org Definition File for DevOps Center

If you create a scratch org based on an org shape with DevOps Center enabled, we still require that you add the DevOps Center feature and setting to the scratch org definition. We require that customers explicitly enable it for legal reasons as part of the DevOps Center terms and conditions.

```
{
  "orgName": "Acme",
  "sourceOrg": "00DB1230400Ifx5",
  "features": ["DevOpsCenter"],
  "settings": {
    "devHubSettings": {
      "enableDevOpsCenterGA": true
    }
  }
}
```

```

        }
    }
}
```

Scratch Org Definition File with Other Features and Settings

To test features that your source org doesn't have, you can add more scratch org features and Metadata API settings. Settings refer to the Settings metadata type, not all metadata types.

```
{
    "orgName": "Acme",
    "sourceOrg": "00DB1230000Ifx5",
    "features": ["Communities", "ServiceCloud", "Chatbot"],
    "settings": {
        "communitiesSettings": {
            "enableNetworksEnabled": true
        },
        "mobileSettings": {
            "enableS1EncryptedStoragePref2": true
        },
        "omniChannelSettings": {
            "enableOmniChannel": true
        },
        "caseSettings": {
            "systemUserEmail": "support@acme.com"
        }
    }
}
```

Next: Create a scratch org using the org shape scratch org definition file.

SEE ALSO:

[Metadata API Developer Guide: Settings](#)

Troubleshooting for Org Shape

Here are some issues you can encounter when using Org Shape for Scratch Orgs.

Some Field Service Features Aren't Enabled in Scratch Orgs Created from Org Shape

Description: Even when the Field Service Enhanced Scheduling and Optimization, and Field Service Integration features are enabled in the source org in which the org shape is created, these features aren't enabled when creating a scratch org based on the org shape.

Workaround: Manually add the missing Field Service Metadata API settings to the scratch org definition depending on which features are enabled in the source org.

Scenario1: If the org shape included both the Field Service Enhanced Scheduling and Optimization, and Field Service Integration features, manually add the Field Service Enhanced Scheduling and Optimization Metadata API setting, o2EngineEnabled, in the scratch org definition file, which enables both features.

```
"settings":
{
    "fieldServiceSettings":
```

```
{
  "fieldServiceOrgPref": true,
  "o2EngineEnabled": true

}
}
```

Scenario 2: If the org shape included only the Field Service Integration feature, manually add the Field Service Enhanced Scheduling and Optimization Metadata API setting, `optimizationServiceAccess`, to the scratch org definition file.

```
"settings":
{
  "fieldServiceSettings":
  {
    "fieldServiceOrgPref": true,
    "optimizationServiceAccess": true

  }
}
```

DevOps Center Isn't Enabled in a Scratch Org Based on an Org Shape

Description: Although DevOps Center is enabled in the source org, the scratch org created from the source org's shape doesn't have DevOps Center enabled. The DevOps Center org preference is purposely toggled off. We require that customers explicitly enable it by indicating the feature and setting in the scratch org definition file for legal reasons as part of the DevOps Center terms and conditions.

Workaround: Add the DevOps Center feature and setting to the scratch org definition file. See [Scratch Org Definition for Org Shape](#) for details.

ERROR running force:org:shape:list

Description: A trial org from which you created the org shape has expired. You could see either of these errors:

```
ERROR running force:org:shape:list: Error authenticating with the refresh token due to:
inactive user
ERROR running force:org:shape:list: Error authenticating with the refresh token due to:
expired access/refresh token
```

Workaround:

- Use `sfdx force:auth:logout` to log out and remove the expired org.
- Run `sfdx force:org:shape:list` again.

Can't create a Digital Experience Cloud Site Using Org Shape

Description: When you try to create a scratch org from an org shape that contains an Experience Cloud Site, you get an error.

```
Required fields are missing: [Welcome Email Template, Change Password Email Template, Lost
Password Template]
```

Workaround: None.

Error While Creating Scratch Org Using a Shape

Description: You see this error when creating a scratch org using a shape.

```
ERROR running force:org:create: A fatal signup error occurred. Please try again.  
If you still see this error, contact Salesforce Support for assistance.
```

Workaround: Generate a new shape using the `force:org:shape:create` command, then try again.

Shift Status Picklists Aren't Populated When Using a Shape With Field Service

Description: When you create a scratch org from a shape with Field Service enabled, the Status field picklist for Shifts is empty.

Workaround: Use an org shape with field service disabled, then enable field service in the scratch org definition file settings.

```
{  
    "orgName": "Acme",  
    "sourceOrg": "00DB1230000Ifx5",  
    "settings": {  
        "fieldServiceSettings": {  
            "fieldServiceOrgPref": true  
        }  
    }  
}
```

Org Shape Feature Accepts Only 15-Character Org IDs

Description: You can use only 15-character org IDs when enabling Org Shape for Scratch Orgs and specifying the source org in the scratch org definition file. Org IDs are usually 18 characters long, which is what the `force:org:list` command displays.

Workaround: Use only the first 15 characters of a standard 18-character org ID when working with the Org Shape feature.

Create Scratch Orgs

After you create the scratch org definition file, you can easily spin up a scratch org and open it directly from the command line.

Before you create a scratch org:

- Set up your Salesforce DX project
- Authorize the Dev Hub org
- Create the scratch org definition file

You can create scratch orgs for different functions, such as for feature development, for development of packages that contain a namespace, or for user acceptance testing.

 **Tip:** Delete any unneeded or malfunctioning scratch orgs in the Dev Hub org or via the command line so that they don't count against your active scratch org allocations.

Indicate the path to the scratch definition file relative to your current directory. For sample repos and new projects, this file is located in the `config` directory.

Ways to Create Scratch Orgs

Create a scratch org for development using a scratch org definition file, give the scratch org an alias, and indicate that this scratch org is the default.

```
sfdx force:org:create -f project-scratch-def.json -a MyScratchOrg --setdefaultusername
```

Specify scratch org definition values on the command line using key=value pairs.

```
sfdx force:org:create adminEmail=me@email.com edition=Developer \
username=admin_user@orgname.org
```

```
sfdx force:org:create sourceOrg=00DB1230000Ifx5
```

Create a scratch org for user acceptance testing or to test installations of packages. In this case, you don't want to create a scratch org with a namespace. You can use this command to override the namespace value in the scratch org definition file.

```
sfdx force:org:create -f project-scratch-def.json --nonamespace
```

Specify the scratch org's duration, which indicates when the scratch org expires (in 1-30 days). The default duration is 7 days.

```
sfdx force:org:create -f config/project-scratch-def.json --durationdays 30
```

Specify the Salesforce release for the scratch org. During the Salesforce release transition, you can specify the release (preview or previous) when creating a scratch org. See [Select the Salesforce Release for a Scratch Org](#).

If Scratch Org Creation Is Successful

Stdout displays two important pieces of information: the org ID and the username.

```
Successfully created scratch org: 00D3D000000PE5UAM,
username: test-b4agup43oxmu@example.com
```

You can now open the org.

```
sfdx force:org:open -u <username/alias>
```

Troubleshooting Tips

If the create command times out before the scratch org is created (the default wait time is 6 minutes), you see an error. Issue this command to see if it returns the scratch org ID, which confirms the existence of the scratch org:

```
sfdx force:data:soql:query -q "SELECT ID, Name, Status FROM ScratchOrgInfo \
WHERE CreatedBy.Name = '<your name>' \
AND CreatedDate = TODAY" -u <Dev Hub org>
```

This example assumes that your name is Jane Doe, and you created an alias for your Dev Hub org called DevHub:

```
sfdx force:data:soql:query -q "SELECT ID, Name, Status FROM ScratchOrgInfo \
WHERE CreatedBy.Name = 'Jane Doe' AND CreatedDate = TODAY" -u DevHub
```

If that doesn't work, create another scratch org and increase the timeout value using the `--wait` parameter. Don't forget to delete the malfunctioning scratch org.

SEE ALSO:

- [Project Setup](#)
- [Authorization](#)
- [Build Your Own Scratch Org Definition File](#)
- [Push Source to the Scratch Org](#)

Select the Salesforce Release for a Scratch Org

During the Salesforce release transition, you can specify the release (preview or previous) when creating a scratch org.

What Is Salesforce Preview?

During every major Salesforce release, you can get early access to the upcoming release in your scratch orgs and sandboxes to test new customizations and features before your production org is upgraded. This window is called the Salesforce Preview, and scratch orgs created on the upcoming release are called preview scratch orgs.

Normally, you create scratch orgs that are the same version as the Dev Hub. However, during the major Salesforce release transition that happens three times a year, you can select the Salesforce release version, `Preview`, or `Previous`, based on the version of your Dev Hub.

To try out new features in an upcoming release, you no longer have to create a trial Dev Hub on the upcoming version to create preview scratch orgs. You can use your existing Dev Hub that includes your existing scratch org active and daily limits.

For example, you can select a version over the next three releases during these release transition dates. Preview start date is when sandbox instances are upgraded. Preview end date is when all instances are on the GA release.

Release Version	Preview Start Date	Preview End Date
Spring '23	January 8, 2023	February 11, 2023
Summer '23	May 7, 2023	June 10, 2023
Winter '24	September 10, 2023	October 14, 2023

Because previous and preview are relative terms, your Dev Hub org version during the release transition determines their relative significance. Here's what happens when you try to create a scratch org with one of the release values.

Dev Hub Version	Preview	Previous
Dev Hub has upgraded to the latest version	Error (Dev Hub is already on the latest version)	Prior Dev Hub version
Dev Hub is still on the GA version	Version following the Dev Hub version (newly released Salesforce version)	Error (Dev Hub is on the GA version; previous version unavailable)

 **Note:** If you don't specify a release value, the scratch org version is the same version as the Dev Hub org.

Create a Scratch Org for a Specific Release

You can specify the release version in the scratch org definition file or directly on the command line. Any option you issue on the command line overrides what you have defined in your scratch definition file.

1. Find out which instance your Dev Hub org is on: <https://status.salesforce.com>.
2. Add the release option (lowercase) to your scratch org definition file.

```
{  
    "orgName": "Dreamhouse",  
    "edition": "Developer",  
    "release": "preview",  
    "settings": {  
        "mobileSettings": {  
            "enableS1EncryptedStoragePref2": true  
        }  
    }  
}
```

Alternatively, you can specify the release value directly on the command line. Any values you specify on the command line override the values in the scratch org definition.

3. Create the scratch org.

In this example, we're creating a scratch org on the preview release.

```
sfdx force:org:create -f config/project-scratch-def.json -a PreviewOrg -v DevHub  
release=Preview
```

Be sure to set the `apiVersion` to match the scratch org version.

To set it globally for all DX projects:

```
sfdx config:set apiVersion=50.0 --global
```

To set it on the command line:

```
SFDX_API_VERSION=50.0 sfdx force:org:create -f config/project-scratch-def.json -a PreviewOrg  
-v DevHub release=Preview
```

 **Note:** Regardless of the release version of your Dev Hub, you can use scratch org features that are available in the release (preview or previous) of the scratch org you create.

What If I Want to Create a Pre-Release Scratch Org?

Pre-release is a very early build of the latest version of Salesforce that's available before Salesforce Preview. It's not built to handle scale and doesn't come with any Salesforce Support service-level agreements (SLAs). For this reason, the only way to create a pre-release scratch org is to sign up for a [pre-release trial Dev Hub org](#) (subject to availability).

Push Source to the Scratch Org

After changing the source, you can sync the changes to your scratch org by pushing the changed source to it.

The first time you push metadata to the org, all source in the folders you indicated as package directories is pushed to the scratch org to complete the initial setup. At this point, we start change-tracking locally on the file system and remotely in the scratch org to determine which metadata has changed. Let's say you pushed an Apex class to a scratch org and then decide to modify the class in the scratch org instead of your local file system. The CLI tracks in which local package directory the class was created, so when you pull it back to your project, it knows where it belongs.

During development, you change files locally in your file system and change the scratch org directly using the builders and editors that Salesforce supplies. Usually, these changes don't cause a conflict and involve unique files.

The push command doesn't handle merges. Projects and scratch orgs are meant to be used by one developer. Therefore, we don't anticipate file conflicts or the need to merge. However, if the push command detects a conflict, it terminates the operation and displays the conflict information to the terminal. You can rerun the push command and force the changes in your project to the scratch org.

Before running the push command, you can get a list of what's new, changed, and the conflicts between your local file system and the scratch org by using `force:source:status`. This way you can choose ahead of time which version you want to keep and manually address the conflict.

Pushing Source to a Scratch Org

To push changed source to your default scratch org:

STATE	FULL NAME	TYPE	PROJECT PATH
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls-meta.xml
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls

To push changed source to a scratch org that's not the default, you can indicate it by its username or alias:

```
sfdx force:source:push --targetusername test-b4agup43oxmu@example.com  
sfdx force:source:push -u test-b4agup43oxmu@example.com  
sfdx force:source:push -u MyGroovyScratchOrg
```



Tip: You can create an alias for an org using `alias:set`. Run `force:org:list` to display the usernames of all the scratch orgs you have created.

Selecting Files to Ignore During Push

It's likely that you have some files that you don't want to sync between the project and scratch org. You can have the push command ignore the files you indicate in `.forceignore`.

If Push Detects Warnings

If you run `force:source:push`, and warnings occur, the CLI doesn't push the source. Warnings can occur, for example, if your project source is using an outdated version. If you want to ignore these warnings and push the source to the scratch org, run:

```
sfdx force:source:push --ignorewarnings
```

 **Tip:** Although you can successfully push using this option, we recommend addressing the issues in the source files. For example, if you see a warning because a Visualforce page is using an outdated version, consider updating your page to the current version of Visualforce. This way, you can take advantage of new features and performance improvements.

If Push Detects File Conflicts

If you run `force:source:push`, and conflicts are detected, the CLI doesn't push the source.

STATE	FULL NAME	TYPE	PROJECT PATH
Conflict	NewClass	ApexClass	/classes/CoolClass.cls-meta.xml
Conflict	NewClass	ApexClass	/classes/CoolClass.cls

Notice that you have a conflict. CoolClass exists in your scratch org but not in the local file system. In this new development paradigm, the local project is the source of truth. Consider if it makes sense to overwrite the conflict in the scratch org.

If conflicts have been detected and you want to override them, here's how you use the power of the force (overwrite) to push the source to a scratch org.

```
sfdx force:source:push --forceoverwrite
```

If Push Detects a Username Reference in the Source

Some metadata types include a username in their source. When you run `force:source:push` to push this source to a scratch org, the push command replaces the username with the scratch org's administrator username. This behavior ensures that the push succeeds, even if the scratch org does not contain the original username.

For example, let's say that you create a scratch org and use Lightning Experience to create a report folder. You then create a report and save it to the new folder. You run `force:source:pull` to pull down the source from the scratch org to your project. The `*.reportFolder-meta.xml` source file for the new ReportFolder is similar to this example; note the `<sharedTo>` element that contains the username `test-ymmlqf5@example.com`.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReportFolder xmlns="http://soap.sforce.com/2006/04/metadata">
    <folderShares>
        <accessLevel>Manage</accessLevel>
        <sharedTo>test-ymmlqf5@example.com</sharedTo>
        <sharedToType>User</sharedToType>
    </folderShares>
    <name>TestFolder</name>
</ReportFolder>
```

You then create a different scratch org whose administrator's username is `test-zuw1xy321@example.com`. If you push the ReportFolder's source file to the new scratch org, `force:source:push` replaces the `test-ymmlqf5@example.com` username with `test-zuw1xy321@example.com`.

This behavior applies only to `force:source:push` and scratch orgs. If you use `force:mdapi:deploy` to deploy metadata to a regular production org, for example, the deploy uses the username referenced in the source.

Next steps:

- Verify that the source was uploaded successfully to the scratch org, open the org in a browser.
- Add some sample test data.

SEE ALSO:

[How to Exclude Source When Syncing or Converting](#)

[Assign a Permission Set](#)

[Ways to Add Data to Your Org](#)

[Pull Source from the Scratch Org to Your Project](#)

[Track Changes Between Your Project and Org](#)

Pull Source from the Scratch Org to Your Project

After you do an initial push, your changes are tracked between your local file system and your scratch org. If you change your scratch org, you usually want to pull those changes to your local project to keep both in sync.

During development, you change files locally in your file system and change the scratch org using builders and editors. Usually, these changes don't cause a conflict and involve unique files.

By default, only changed source is synced back to your project.

The pull command does not handle merges. Projects and scratch orgs are meant to be used by one developer. Therefore, we don't anticipate file conflicts or the need to merge. However, if the pull command detects a conflict, it terminates the operation and displays the conflict information to the terminal. You can rerun the command with the force option if you want to pull changes from your scratch org to the project despite any detected conflicts.

Before you run the pull command, you can get a list of what's new, changed, and any conflicts between your local file system and the scratch org by using `force:source:status`. This way you can choose ahead of time which files to keep.

To pull changed source from the scratch org to the project:

```
sfdx force:source:pull
```

You can indicate either the full scratch org username or an alias. The terminal displays the results of the pull command. This example adds two Apex classes to the scratch org. The classes are then pulled to the project in the default package directory. The pull also indicates which files have changed since the last push and if a conflict exists between a version in your local project and the scratch org.

STATE	FULL NAME	TYPE	PROJECT PATH
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls-meta.xml
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls
Changed	CoolClass	ApexClass	/classes/CoolClass.cls-meta.xml
Changed	CoolClass	ApexClass	/classes/CoolClass.cls

To pull source to the project if a conflict has been detected:

```
sfdx force:source:pull --forceoverwrite
```

SEE ALSO:

[Track Changes Between Your Project and Org](#)

Scratch Org Users

A scratch org includes one administrator user by default. The admin user is typically adequate for all your testing needs. But sometimes you need other users to test with different profiles and permission sets.

You can create a user by opening the scratch org in your browser and navigating to the Users page in Setup. You can also use the `force:user:create` CLI command to easily integrate the task into a continuous integration job.

Scratch Org User Limits, Defaults, and Considerations

- You can create a user only for a specific scratch org. If you try to create a user for a non-scratch org, the command fails. Also specify your Developer Hub, either explicitly or by setting it as your default, so that Salesforce can verify that the scratch org is active.
- Your scratch org edition determines the number of available user licenses. Your number of licenses determines the number of users you can create. For example, a Developer Edition org includes a maximum of two Salesforce user licenses. Therefore, in addition to the default administrator user, you can create one standard user.
- The new user's username must be unique across all Salesforce orgs and in the form of an email address. The username is active only within the bounds of the associated scratch org.
- You can't delete a user. The user is deactivated when you delete the scratch org with which the user is associated. Deactivating a user frees up the user license. But you can't reuse usernames, even if the associated user has been deactivated.
- The simplest way to create a user is to let the `force:user:create` command assign default or generated characteristics to the new user. If you want to customize your new user, create a definition file and specify it with the `--definitionfile (-f)` parameter. In the file, you can include all the User sObject fields and a set of Salesforce DX-specific options, described in [User Definition File for Customizing a Scratch Org User](#). You can also specify these options on the command line.
- If you do not customize your new user, the `force:user:create` command creates a user with the following default characteristics.
 - The username is the existing administrator's username prepended with a timestamp. For example, if the administrator username is `test-wvkpnfm5z113@example.com`, the new username is something like `1505759162830_test-wvkpnfm5z113@example.com`.
 - The user's profile is Standard User.
 - The values of the required fields of the User sObject are the corresponding values of the administrator user. For example, if the administrator's locale (specifically the `LocaleSidKey` field of User sObject) is `en_US`, the new user's locale is also `en_US`.

[Create a Scratch Org User](#)

Sometimes you need other users to test with different profiles and permission sets.

[User Definition File for Customizing a Scratch Org User](#)

To customize a new user, rather than use the default and generated values, create a definition file.

[Generate or Change a Password for a Scratch Org User](#)

By default, new scratch orgs contain one administrator user with no password. You can optionally set a password when you create a user. Use the CLI to generate or change a password for any scratch org user. After it's set, you can't unset a password, you can only change it.

SEE ALSO:

[User sObject API Reference](#)

Create a Scratch Org User

Sometimes you need other users to test with different profiles and permission sets.

Use the `force:user:create` command to create a user. Specify the `--setalias` parameter to assign a simple name to the user that you can reference in later CLI commands. When the command completes, it outputs the new username and user ID.

```
sfdx force:user:create --setalias qa-user
```

```
Successfully created user "test-b4agup43oxmu@example.com" with ID [0059A000000U0psQAC] for org 00D9A000000SXKUA2.
```

```
You can see more details about this user by running "sfdx force:user:display -u test-b4agup43oxmu@example.com".
```

Users are associated with a specific scratch org and Developer Hub. Specify the scratch org or Developer Hub username or alias at the command line if they aren't already set by default in your environment. If you try to create a user for a non-scratch org, the `force:user:create` command fails.

```
sfdx force:user:create --setalias qa-user --targetusername my-scratchorg  
--targetdevhubusername my-dev-hub
```

The `force:user:create` command uses default and generated values for the new user, such as the user's username, profile, and locale. You can customize the new user by creating a definition file and specifying it with the `--definitionfile` parameter.

```
sfdx force:user:create --setalias qa-user --definitionfile config/user-def.json
```

View the list of users associated with a scratch org with the `force:user:list` command. The (A) on the left identifies the administrator user that was created at the same time that the scratch org was created.

```
sfdx force:user:list
```

	ALIAS	USERNAME	PROFILE NAME	USER ID
(A)	admin-user	test-b4agup43oxmu@example.com	System Administrator	005xx000001SvBPAA0
	ci-user	wonder@example.com	Standard User	005xx000001SvBaAK

Display details about a user with the `force:user:display` command.

```
sfdx force:user:display --targetusername ci-user
```

```
==== User Description
```

KEY	VALUE
-----	-------

Access Token	<long-string>
--------------	---------------

Alias	ci-user
Id	005xx000001SvBaAAK
Instance Url	https://innovation-ability-4888-dev-ed.my.salesforce.com
Login Url	https://innovation-ability-4888-dev-ed.my.salesforce.com
Org Id	00D9A0000000SXKUA2
Profile Name	Standard User
Username	test-b4agup43oxmu@example.com

User Definition File for Customizing a Scratch Org User

To customize a new user, rather than use the default and generated values, create a definition file.

The user definition file uses JSON format and can include any Salesforce User sObject field and these Salesforce DX-specific options.

Salesforce DX Option	Description	Default If Not Specified
permsets	An array of permission sets assigned to the user. Separate multiple values with commas, and enclose in square brackets. You must have previously pushed the permission sets to the scratch org with <code>force:source:push</code> .	None
generatePassword	Boolean. Specifies whether to generate a random password for the user. If set to <code>true</code> , <code>force:user:create</code> displays the generated password after it completes. You can also view the password using <code>force:user:describe</code> .	False
profileName	Name of a profile to associate with the user. Similar to the ProfileId field of the User sObject except that you specify the name of the profile and not its ID. Convenient when you know only the name of the profile.	Standard User

The user definition file options are case-insensitive. However, we recommend that you use lower camel case for the Salesforce DX-specific options and upper camel case for the User sObject fields. This format is consistent with other Salesforce DX definition files.

This user definition file includes some User sObject fields and three Salesforce DX options (`profileName`, `permsets`, and `generatePassword`).

```
{
  "Username": "tester1@sfdx.org",
  "LastName": "Hobbs",
  "Email": "tester1@sfdx.org",
  "Alias": "tester1",
  "TimeZoneSidKey": "America/Denver",
  "LocaleSidKey": "en_US",
```

```

    "EmailEncodingKey": "UTF-8",
    "LanguageLocaleKey": "en_US",
    "profileName": "Standard Platform User",
    "permsets": ["Dreamhouse", "Cloudhouse"],
    "generatePassword": true
}

```

In the example, the username tester1@sfdx.org must be unique across the entire Salesforce ecosystem; otherwise, the `force:user:create` command fails. The alias in the Alias option is different from the alias you specify with the `--setalias` parameter of `force:user:create`. You use the Alias option alias only with the Salesforce UI. The `--setalias` alias is local to the computer from which you run the CLI, and you can use it with other CLI commands.

You indicate the path to the user definition file with the `--definitionfile` parameter of the `force:user:create` CLI command. You can name this file whatever you like and store it anywhere the CLI can access.

```
sfdx force:user:create --setalias qa-user --definitionfile config/user-def.json
```

You can override an option in the user definition file by specifying it as a name-value pair at the command line when you run `force:user:create`. This example overrides the username, list of permission sets, and whether to generate a password.

```
sfdx force:user:create --setalias qa-user --definitionfile config/user-def.json
permsets="Dreamy,Cloudy" Username=tester345@sfdx.org generatePassword=false
```

You can also add options at the command line that are not in the user definition file. This example adds the City option.

```
sfdx force:user:create --setalias qa-user --definitionfile config/user-def.json City=Oakland
```

SEE ALSO:

[User sObject API Reference](#)

Generate or Change a Password for a Scratch Org User

By default, new scratch orgs contain one administrator user with no password. You can optionally set a password when you create a user. Use the CLI to generate or change a password for any scratch org user. After it's set, you can't unset a password, you can only change it.

1. Generate a password for a scratch org user with this command:

```
sfdx force:user:password:generate --targetusername <username>
```

You can run this command for scratch org users only. The command outputs the generated password.

The target username must be an administrator user. The `--onbehalfof` parameter lets you assign passwords to multiple users at once, including admin users, or to users who don't have permissions to do it themselves. Specify multiple users by separating them with commas; enclose them in quotes if you include spaces. The command still requires an administrator user, which you specify with the `--targetusername` parameter. For example, let's say the administrator user has alias `admin-user` and you want to generate a password for users with aliases `ci-user` and `qa-user`:

```
sfdx force:user:password:generate --targetusername admin-user --onbehalfof
"ci-user, qa-user"
```

By default, the command generates a password that's 13 characters in length; the possible characters include all lower and upper case letters, numbers, and symbols. To change the password strength, use the `--length` and `--complexity` parameters. The `--complexity` parameter is a number from 0 through 5; the higher the value, the more possible characters are used, which

strengthens the password. The default value is 5. See the command-line help for a description of each value. This example shows how to generate a password that's 20 characters long:

```
sfdx force:user:password:generate --targetusername admin-user --length 20
```

2. View the generated password and other user details:

```
sfdx force:user:display --targetusername ci-user

==== User Description
KEY          VALUE
-----
Access Token <long-string>
Alias        ci-user
Id           005xx000001SvBaAAK
Instance Url https://innovation-ability-4888-dev-ed.my.salesforce.com
Login Url   https://innovation-ability-4888-dev-ed.my.salesforce.com
Org Id      00D9A0000000SXKUA2
Password     bAc00R&ob$
Profile Name Standard User
Username    test-b4agup43oxmu@example.com
```

3. Log in to the scratch org with the new password:

- From the `force:user:display` output, copy the value of Instance URL and paste it into your browser. In our example, the instance URL is <https://site-fun-3277.my.salesforce.com>.
- If you've already opened the scratch org with the `force:org:open` command, you're automatically logged in again. To try out the new password, log out and enter the username and password listed in the output of the `force:user:display` command.
- Click **Log In to Sandbox**.



Note: If you change a scratch org user's password using the Salesforce UI, the new password doesn't show up in the `force:user:display` output.

Manage Scratch Orgs from Dev Hub

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

In Dev Hub, ActiveScratchOrgs represent the scratch orgs that are currently in use. ScratchOrgInfos represent the requests that were used to create scratch orgs and provide historical context.

1. Log in to Dev Hub org as the System Administrator or as a user with the Salesforce DX permissions.
2. From the App Launcher, select **Active Scratch Org** 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 Org list view, choose **Delete** from the dropdown.

Deleting an active scratch org does not 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 Info** 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.

SEE ALSO:

[Add Salesforce DX Users](#)

Scratch Org Error Codes

If scratch org creation fails, the system generates an error code that can help you identify the cause.

Error Code	Description
C-9998	Not a valid scratch org. Contact Salesforce Customer Support for assistance.
C-9999	Generic fatal error. Contact Salesforce Customer Support for assistance.
S-1017	Namespace isn't registered. 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. See Salesforce Help: Link a Namespace to a Dev Hub Org .
S-2006	Invalid country.
SH-0001	Can't create scratch org from org shape. Contact the source org admin to add your Dev Hub org ID. From Setup, in the Quick Find box, enter <i>Org Shape</i> , and then select Org Shape .
SH-0002	Can't create scratch org. No org shape exists for the specified <code>sourceOrg</code> . Create an org shape and try again.
SH-0003	Can't create scratch org from org shape. The org shape version is outdated. Recreate the org shape and try again.
SH-9999	Can't validate org shape due to fatal error. Contact Salesforce Customer Support for assistance.

CHAPTER 8 Sandboxes

In this chapter ...

- [Authorize in to Your Production Org](#)
- [Create a Sandbox Definition File](#)
- [Create, Clone, or Delete a Sandbox](#)

Sandboxes are copies of your Salesforce org that you can use for development, testing, and training, without compromising the data and applications in your production org.

Salesforce offers sandboxes and a set of deployment tools, so you can:

- Isolate customization and development work from your production environment until you're ready to deploy changes.
- Test changes against copies of your production data and users.
- Provide a training environment.
- Coordinate individual changes into one deployment to production.

Traditionally, you or your Admin has created and managed your sandboxes through the Setup UI. But we realize that many developers want the ability to create and manage their developer and testing environments programmatically, and to automate their CI processes. Salesforce CLI enables you to do both.

USER PERMISSIONS

To view a sandbox:

- [View Setup and Configuration](#)

To create, refresh, activate, and delete a sandbox:

- [Manage Sandbox](#)

Where Do Sandboxes Fit in the Application Development Lifecycle?

The development model you use determines in which stages you use sandboxes. For more information on our development models and where sandboxes fit, see [Determine Which Application Lifecycle Model Is Right for You](#) (Trailhead).

Authorize in to Your Production Org

JWT and Web-based flows require a production org with sandbox licenses instead of a Dev Hub. However, it's okay if your production org is also a Dev Hub org.

The examples in [Authorize an Org Using the JWT-Based Flow](#) and [Authorize an Org Using the Web-Based Flow](#) are geared toward scratch orgs. Follow these tips to successfully authorize to your production org.

- Be sure to use `https://login.salesforce.com` for `sfdcLoginUrl` in `sfdx-project.json` file. Alternatively, you can use `auth:jwt:grant --instanceurl` to specify the URL directly on the command line. This value overrides the login URL you specified in the `sfdx-project.json` file.
- Specify the username for your production org when running the `auth:jwt:grant` command. No need to specify a Dev Hub or indicate a default Dev Hub.
- The JWT authorization flow requires that you create a connected app. When you create the connected app, log in to your production org, not a Dev Hub org.

Create a Sandbox Definition File

Before you can create a sandbox using Salesforce CLI, define the configuration for it in a sandbox definition file. The sandbox definition file is a blueprint for the sandbox. You can create different definition files for each sandbox type that you use in the development process.

Sandbox Configuration Values

Option	Required?	Description
<code>apexClassId</code>	No	A reference to the ID of an Apex class that runs after each copy of the sandbox. Allows you to perform business logic on the sandbox to prepare it for use.
<code>autoActivate</code>	No	If <code>true</code> , you can activate a sandbox refresh immediately.
<code>copyArchivedActivities</code>	No	Full sandboxes only. This field is visible if your organization has purchased an option to copy archived activities for sandbox. To obtain this option, contact Salesforce Customer Support.
<code>copyChatter</code>	No	If <code>true</code> , archived Chatter data is copied to the sandbox.
<code>description</code>	No	A description of the sandbox (1000 or fewer characters), which helps you distinguish it from other sandboxes.
<code>historyDays</code>	No	Full sandboxes only. Represents the number of days of object history to be copied in the sandbox. Valid values: <ul style="list-style-type: none"> • -1, which means all available days • 0 (default)

Option	Required?	Description
		<ul style="list-style-type: none"> • 10 • 20 • 30 • 60 • 90 • 120 • 150 • 180
licenseType	Yes (for sandbox creation)	Valid values are <code>Developer</code> , <code>Developer_Pro</code> , <code>Partial</code> , and <code>Full</code> .
sandboxName	Yes	A unique alphanumeric string (10 or fewer characters) to identify the sandbox. You can't reuse a name while a sandbox is in the process of being deleted.
sourceSandboxName	Yes (for sandbox cloning)	Name of the sandbox being cloned.
templateId	Yes (for Partial sandboxes)	<p>Optional for Full sandboxes. Not available for Developer and Developer Pro sandboxes.</p> <p>A reference to the sandbox template as identified by the 15-character ID beginning with <code>1ps</code> in the URL when viewing a sandbox template in a browser. A sandbox template lets you select which objects to copy in a sandbox.</p>



Note: You can indicate either `licenseType` or `sourceSandboxName` in your definition file, but not both.

Sample Sandbox Definition File

Although you can place the sandbox definition file anywhere, we recommend keeping it in your Salesforce DX project in the `config` directory. When naming the file, we suggest providing a descriptive name that ends in `sandbox-def.json`, for example, `developer-sandbox-def.json`.

Here's a sample definition file for creating a sandbox:

```
{
  "sandboxName": "dev1",
  "licenseType": "Developer"
}
```

Here's a sample definition file for cloning a sandbox:

```
{
  "sandboxName": "dev1clone",
```

```
        "sourceSandboxName": "dev1"  
    }
```

SEE ALSO:

[Tooling API: SandboxInfo](#)

Create, Clone, or Delete a Sandbox

Create a sandbox to use for development, testing, or training. Clone a sandbox to copy its data and metadata to another sandbox.

Before you create or clone a sandbox:

- Create a Salesforce DX project with a manifest file.
- Authorize to a production org with available sandbox licenses.
- Create the sandbox definition file.

Why We Recommend Using Aliases

When you create or clone a sandbox, the usernames generated in the sandbox are based on the usernames present in the production org or sandbox. The username looks like an email address, such as `username@company.com.dev1`. If the resulting username is not unique, we prepend some characters and digits to the username. The modified username looks something like

00x7Vqusername@company.com.dev1.

As you can imagine, remembering these usernames can be challenging, especially if you have several sandboxes you're managing. Aliasing is a powerful way to manage and track your orgs, and we consider it a best practice. So when you issue a command that requires the username, using an alias that you can remember can speed up things.

If you didn't set an alias when you created the sandbox, you can set one later.

```
sfdx alias:set MyDevSandbox=username@company.com.dev1
```

Create a Sandbox

Optional: Create a Sandbox Definition File

When you create a sandbox, Salesforce copies the metadata and optionally data from your production org to a sandbox org.

```
sfdx force:org:create --type sandbox --targetusername prodOrg --definitionfile  
config/dev-sandbox-def.json -a MyDevSandbox -s -w 30
```

The `-s` flag indicates that this sandbox is your default org for all CLI commands. If you're working with several orgs and you don't want this one to be the default, exclude this flag.

To directly define the required sandbox options, or to override the values defined in the sandbox definition file, specify key=value pairs on the command line.

```
sfdx force:org:create -t sandbox sandboxName=FullSbx licenseType=Full -u prodOrg -a  
MyFullSandbox -w 30
```

 **Tip:** Because the sandbox is processed in a queue, the sandbox creation process can take longer than the default wait time of 6 minutes. We recommend setting a larger value for `--wait`, for example, 30 minutes.

How long the creation process takes depends on the size and complexity of your production org. You see status messages posted to output:

```
Sandbox request dev1(0GXQ0000000CftJOWS) is Pending (0% completed). Sleeping 30 seconds.  
Will wait 30 minutes more before timing out.  
Sandbox request dev1(0GXQ0000000CftJOWS) is Processing (0% completed). Sleeping 30 seconds.  
Will wait 29 minutes 30 seconds more before timing out.
```

Once the wait period is over, you can run the `force:org:status` command to check the status of the sandbox creation process. If the sandbox is created within the wait time, the CLI automatically authenticates in to the sandbox. And the sandbox appears in the output of the `force:org:list` command. Team members can authenticate to the sandbox by running the `auth:web:login` command and providing their usernames and passwords.

```
sfdx auth:web:login -r https://test.salesforce.com
```

Clone a Sandbox

You can create a sandbox by cloning an existing sandbox rather than using your production org as your source. You can save time by customizing a sandbox with a set of data and metadata and then replicating it.

Sandbox cloning simplifies having multiple concurrent streams of work in your application life cycle. You can set up a sandbox for each type of work, such as development, testing, and staging. Your colleagues can easily clone individual sandboxes instead of sharing one sandbox and stepping on each other's toes.

```
sfdx force:org:clone -t sandbox -f config/dev-sandbox-def.json -u prodOrg -a MyDevSandbox  
-s -w 30
```

To override the configuration values defined in the sandbox definition file, specify key=value pairs on the command line.

```
sfdx force:org:clone -t sandbox SandboxName>NewClonedSandbox  
SourceSandboxName=ExistingSandbox -u prodOrg -a MyDevSandbox -w 30
```

 **Tip:** Because the sandbox is processed in a queue, the sandbox cloning process can take longer than the default wait time of 6 minutes. We recommend setting a larger value for `--wait`, for example, 30 minutes.

Once the wait period is over, you can run the `force:org:status` command to check the status of the sandbox cloning process. If the sandbox is cloned within the wait time, the CLI automatically authenticates in to the sandbox. And the sandbox appears in the output of the `force:org:list` command. Team members can authenticate to the sandbox by running the `auth:web:login` command and providing their usernames and passwords.

```
sfdx auth:web:login -r https://test.salesforce.com
```

Check the Sandbox Status

Creating or cloning a sandbox can take several minutes. Once the command times out, you can run the `force:org:status` command to report on creation or cloning status. When the sandbox is ready, this command authenticates to the sandbox.

If the `org:create` or `org:clone` command times out, the alias isn't set. However, you can set it using the `org:status` command:

```
sfdx force:org:status -n DevSbx1 -a MyDevSandbox -u prodOrg
```

Open a Sandbox

Once the sandbox is ready, you can open it by specifying its username or alias. However, you don't have to provide its password because the CLI manages the authentication details for you.

```
sfdx force:org:open -u MyDevSandbox
```

Delete a Sandbox

You can delete a sandbox using the CLI if it was authenticated when running `org:create`, `org:clone`, or `org:status`. Other sandboxes that you authenticated using `auth:web:login` or `auth:jwt:grant` also appear on the org list, but must be deleted using the sandbox detail page in your production org.

```
sfdx force:org:delete -u MyDevSandbox
```

Next:

- Retrieve metadata from your sandbox to your local DX project.
- Develop directly in your sandbox, then retrieve the changes to your local DX project.
- Deploy local changes to a sandbox.

SEE ALSO:

[Salesforce Help: Deploy Enhancements from Sandboxes](#)

[Salesforce Help: Create, Clone, or Refresh a Sandbox Using Setup UI](#)

[Authorize an Org Using the JWT Bearer Flow](#)

[Authorize an Org Using the Web Server Flow](#)

CHAPTER 9 Track Changes Between Your Project and Org

In this chapter ...

- See Changes Identified by Source Tracking
- Pull and Push Changes Identified by Source Tracking
- Resolve Conflicts Between Your Local Project and Org
- Get Change Information by Querying the SourceMember Object
- Best Practices
- Differences and Considerations

Run source tracking to see a list of components you create, update, or delete between your local project and a scratch org or sandbox.

In addition to listing the changes you make, source tracking makes it possible to:

- Automatically track changes to metadata components, saving you from tracking them manually.
- See changes pushed to a sandbox by other developers.
- Push or pull changed source.
- Identify and resolve conflicts between your local project and scratch org or sandbox before pushing or pulling source.

To see which metadata components support source tracking, check the Source Tracking column of the [Metadata Coverage Report](#).

Before working with source tracking, ensure that you complete these prerequisites.

1. [Install the Salesforce CLI](#).
2. [Enable Dev Hub](#).
3. [Enable Source Tracking for Sandboxes](#).
4. [Create a DX Project](#).
5. Create a [scratch org](#) or [sandbox](#), or refresh a sandbox that was created before enabling source tracking for sandboxes.

EDITIONS

You can enable source tracking in Developer and Developer Pro sandboxes only.

Source tracking in scratch orgs is available in: Professional, Enterprise, Performance, Unlimited, Database.com, and Developer Editions

USER PERMISSIONS

To track changes between your project and a sandbox:

- Manage Sandbox

To enable Source Tracking in sandboxes:

- Customize Application

How CLI Commands Support Source Tracking

While all the `force:source:*` commands support source-tracking, there are differences in how you use them.

 **Note:** In this section, the terms *push* and *deploy* are used interchangeably and have the same meaning: moving source from the local project to the target org. Similarly, *pull* and *retrieve* both mean moving source from the org to the local project.

The `force:source:push|pull` commands push or pull all changed source between your local project and the target org. The important word is *all*: you can't choose specific source files to push or pull. The commands work on all the changed files.

In contrast, the `force:source:deploy|retrieve` commands give you more granular control. By specifying the appropriate parameter, you can deploy or retrieve specific metadata components,

Track Changes Between Your Project and Org

package directories, or manifest files. To ensure that your source tracking files are updated, you must specify the `--tracksource` parameter. This example retrieves the `MyFabClass` Apex class:

```
sfdx force:source:retrieve --metadata ApexClass:MyFabClass  
--tracksource
```

In general, we recommend that you use `force:source:push|pull` to reflect all changes between your local project and org together, rather than specific changes one by one. Pushing or pulling all changes at once ensures that the tracking files accurately reflect your current state. For this reason, the examples in the source tracking topics use `force:source:push|pull` and not `force:source:deploy|retrieve`.

See Changes Identified by Source Tracking

To see changes between your local project and target org, navigate to the project directory for which you want to see changes.

1. In a terminal or command window, navigate to the project directory. In this example, the directory is named MyProject.

```
cd MyProject
```

2. Run the `force:source:status` command. Include the `-u` parameter to specify the username of the scratch org or sandbox that you want to compare with your local project. In this example, the username is DevSandbox.

```
sfdx force:source:status -u DevSandbox
```

The CLI displays the differences between the local project and the org. In this example, the local project adds an Apex class named `WidgetClass`. The org includes changes to a custom object named `Widget__c`, that haven't been pulled to the local project, but they conflict with `Widget__c` in the local project. In this example, someone working in the org deletes a listview called `All` on `Widget__c` and adds a permission set named `WidgetPermissions`. Pushing source to the org adds the `WidgetClass` `ApexClass`. Pulling source from the org changes `Widget__c`, deletes the `All` listview on `Widget__c`, and creates a permission set named `WidgetPermissions` in the local project.

==== Source Status			
STATE	FULL NAME	TYPE	PROJECT PATH
Local Add	WidgetClass	ApexClass	
	<code>./classes/WidgetClass.cls-meta.xml</code>		
Local Add	WidgetClass	ApexClass	<code>./classes/WidgetClass.cls</code>
Remote Changed (Conflict)	<code>Widget__c</code>	CustomObject	
	<code>./objects/Widget__c/Widget__c.object-meta.xml</code>		
Remote Deleted	<code>Widget__c.all</code>	ListView	
	<code>./objects/Widget__c/listViews/All.listView-meta.xml</code>		
Remote Add	<code>WidgetPermissions</code>	PermissionSet	

Source tracking returns a table of change information with four columns: STATE, FULL NAME, TYPE, and PROJECT PATH. Each row represents one change.

STATE details information about the change. It tells you where and what the change is, and whether there's a conflict. Here's what the different values of STATE mean.

- Local — The change is present in your local project and awaits a push to your org.
- Remote — The change is present in your org and awaits a pull to your local project.
- Add — A component is created.
- Changed — A component is edited. For example, a custom object is changed when fields are added or removed.
- Deleted — A component is removed.
- (Conflict) — Normally, Salesforce CLI automatically creates, deletes, or updates components when you push or pull source. In this case, the CLI encountered a change that it couldn't automatically resolve. Before pushing or pulling changes, resolve the conflicts.

FULL NAME is the API name of the component.

TYPE is the component's metadata type. It describes what the component is, such as an Apex class or a custom object.

PROJECT PATH is the location of the component in your local project. If it's blank, the component isn't present in your local project. When blank, it usually means that a component is present in the org but not in your local project.

If source tracking doesn't detect any changes, then the `force:source:status` command returns a statement saying `No results found`.

```
==== Source Status
No results found
```

After reviewing the differences between source your local project and the org, you're ready to push or pull source. With source tracking, only the changed source gets pushed or pulled.

Pull and Push Changes Identified by Source Tracking

When you create a Salesforce app, you typically use both low-code and pro-code techniques. An example of low-code is creating a custom object directly in an org using Setup. An example of pro-code is creating an Apex class in your local project using an IDE, such as VS Code. As you work, source tracking identifies changes so you can keep the remote metadata in the org in sync with the source in your local project.

The process is iterative. First you get a status of the remote and local changes. If conflicts exist, you resolve them. You now must ensure that these changes exist in both the org and your local project. So you pull the remote changes so that your local project, and then your version control system, contains everything and is the source of historical truth. You push your local changes, such as Apex code, to the org so you can validate and test it. And you keep iterating through this process until you finish developing the Salesforce app.

Let's look at some examples to see source tracking in action.

Say you run `force:source:status` and see local and remote changes.

==== Source Status			
STATE	FULL NAME	TYPE	PROJECT PATH
Local Add	WidgetClass	ApexClass	./classes/WidgetClass.cls-meta.xml
Local Add	WidgetClass	ApexClass	./classes/WidgetClass.cls
Remote Changed	Widget__c	CustomObject	
	./objects/Widget__c/Widget__c.object-meta.xml		
Remote Deleted	Widget__c.all	ListView	
	./objects/Widget__c/listViews/All.listView-meta.xml		
Remote Add	WidgetPermissions	PermissionSet	

As a best practice, pull changes and resolve conflicts before pushing your changes to the org. This practice helps other developers incorporate your changes and generally facilitates collaboration. For help with resolving conflicts, see [Resolve Conflicts Between Your Local Project and Org](#) on page 144.

```
sfdx force:source:pull -u DevSandbox
```

After pulling source, run `force:source:status` again. Now, source tracking only reports your local changes because pulling updated your local project to match what's in the org.

==== Source Status			
STATE	FULL NAME	TYPE	PROJECT PATH
Local Add	WidgetClass	ApexClass	./classes/WidgetClass.cls-meta.xml
Local Add	WidgetClass	ApexClass	./classes/WidgetClass.cls

Next, push your local changes. After pushing, other developers see your changes in the org as remote changes.

```
sfdx force:source:push -u DevSandbox
```

Run `force:source:status` again.

```
==== Source Status
No results found
```

The command reports no results, indicating that your local project and the org are synchronized.

Resolve Conflicts Between Your Local Project and Org

As a best practice, if conflicts exist for components in your local project or in the org, resolve them before moving forward. You can resolve the conflict manually, or overwrite one version of a component with another. Only overwrite changes if you're certain that the new version is the one you want to use.

Say you run `force:source:status` and see conflicting changes in your local project and in the org.

STATE	FULL NAME	TYPE	PROJECT PATH
Local Add (Conflict)	WidgetClass	ApexClass	./classes/WidgetClass.cls-meta.xml
Local Add (Conflict_	WidgetClass	ApexClass	./classes/WidgetClass.cls
Remote Changed (Conflict)	Widget__c	CustomObject	./objects/Widget__c/Widget__c.object-meta.xml

If you pull or push source, then the CLI reports these conflicts again and stops the operation from completing. To pull or push, resolve the conflicts, and then overwrite your local project or the org with the resolved file.

Overwrite Conflicting Changes

If you have no doubts that the local or remote version is correct, you can overwrite conflicting changes by including the `-f | --forceoverwrite` parameter when pulling or pushing changes.

To overwrite your local project changes with source from the org, include the `-f | --forceoverwrite` parameter when pulling changes.

```
sfdx force:source:pull -u DevSandbox -f
```

After pulling source, your local project has the same version of `WidgetClass` and `Widget__c` that existed in the org. If you had different versions of these same files in your local project, they're replaced with the versions that exist in your org.

To overwrite the org with source from your local project, include the `-f | --forceoverwrite` parameter when pushing changes.

```
sfdx force:source:push -u DevSandbox -f
```

After pushing source, your org has the same version of `WidgetClass` and `Widget__c` that exists on your local project. If your org contained different version of those same components, they're replaced with the versions from your local DX project.

Get Change Information by Querying the SourceMember Object

If you want to know more about the source-tracked changes in a developer org, query the `SourceMember` object to find out what changes were made and who made them.

For example, see a list of changed components and who last changed them by running a query like this from the CLI. In the example, DevSandbox is the username or alias of your sandbox.

```
sfdx force:data:soql:query -q "SELECT MemberName, MemberType, ChangedBy, RevisionCounter
FROM SourceMember" -t -u DevSandbox
```

The query returns a list of components. The CHANGEDBY column returns the user ID of the person who last changed the component.

MEMBERNAME	MEMBERTYPE	CHANGEDBY	REVISIONCOUNTER
Account.MyTextField__c	CustomField	0056g000000FDuZ	1
Contact.MyField__c	CustomField	0056g000000FDuZ	11
Admin	Profile	0051k000002KW4R	54
MyObj1__c-MyObj1 Layout	Layout	0051k000002KW4R	55
Account-Account Layout	Layout	0056g000000FDuZ	33
Contact-Contact Layout	Layout	0056g000000FDuZ	45
MyObj1__c	CustomObject	0051k000002KW4R	57
TVRemoteControl	ApexClass	0056g000000FDuZ	53

To get the username of the person who last changed the TVRemoteControl Apex class, for example, run a second query that maps the IDs from CHANGEDBY to usernames:

```
sfdx force:data:soql:query -q "SELECT Id,Username,Name FROM User WHERE Id =
'0056g000000FDuZ'" -u DevSandbox
```

And the result is similar to:

ID	USERNAME	NAME
0056g000000FDuZQAS	dev2@example.com.devsb1	Dev2

Best Practices

Get the most out of source tracking by following these best practices.

Review metadata change history with a version control system like Git

With a version control system, you can version your changes, track change history, and review metadata changes before promoting to other environments, like a sandbox.

Get source tracking files back into sync

If source tracking gets confused and starts reporting inaccuracies, you can use the `force:source:deploy|retrieve` commands to get back into sync. Which command you use depends on which source you most trust: use `force:source:deploy` if you trust your local source files and `force:source:retrieve` if you trust what's in your org. For either command, specify the `--forceoverwrite` parameter, which overwrites changed source in the org that conflicts with its local equivalent. Also specify `--tracksource`, which ensures that the source tracking files are correctly synchronized.

For example, let's say your project has two package directories that contain all your source files, and you trust your local project over what's in your org. Run this command, which deploys all local source to your org and synchronizes the source tracking files:

```
sfdx force:source:deploy -p force-app,force-app2 --tracksource --forceoverwrite
```

Differences and Considerations

As you get ready to work with source tracking, note these behavioral differences and considerations.

Performance Considerations of Source Tracking

Source tracking performs extra functions to determine changes to source tracked components, such as running more queries. So, some commands can take a little longer to run when working with medium-to-large sized projects. If you're working with small projects, you don't notice any slowdown.

Retrieve and Pull Changes to Profiles with Source Tracking

Retrieving and pulling profiles behaves a little differently with source tracking.

Performance Considerations of Source Tracking

Source tracking performs extra functions to determine changes to source tracked components, such as running more queries. So, some commands can take a little longer to run when working with medium-to-large sized projects. If you're working with small projects, you don't notice any slowdown.

A medium-sized project has 30 or more components or 50 or more tests. A project with 25 components and 51 tests is considered medium.

A large-sized project is 600 or more components or 150 or more tests. A project with 610 components and 140 tests is considered large.

If you experience long-running commands, break up your projects into smaller sets of components, and deploy the smaller sets.

Retrieve and Pull Changes to Profiles with Source Tracking

Retrieving and pulling profiles behaves a little differently with source tracking.

Without source tracking, retrieving profiles only returns some profile information. Retrieving profiles returns information about profiles that pertains to other items specified in the `package.xml` file.

For example, retrieving profiles with this `package.xml` file returns profile permissions for the `MyCustomField__c` custom field on the `Account` object.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
  <types>
    <members>Account.MyCustomField__c</members>
    <name>CustomField</name>
  </types>
  <types>
    <members>*</members>
    <name>Profile</name>
  </types>
  <version>50.0</version>
</Package>
```

With source tracking, retrieving profiles returns profile information pertaining to anything else specified in the `package.xml` file plus any components getting tracked by source tracking. That includes any entity for which a change exists between your local project and the org.

For example, say you create a custom field on the Opportunity object called OppCustomField__c in your local environment. Source tracking detects the change and reports it. Now you retrieve profiles using the same `package.xml` file as you did when source tracking was off.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
  <types>
    <members>Account.MyCustomField__c</members>
    <name>CustomField</name>
  </types>
  <types>
    <members>*</members>
    <name>Profile</name>
  </types>
  <version>50.0</version>
</Package>
```

Even though the `package.xml` file doesn't reference `OppCustomField__c`, because source tracking is tracking the new custom field, your retrieve returns profile permissions for both the `MyCustomField__c` custom field on the Account object and the `OppCustomField__c` on the Opportunity object.

For more information about retrieving profiles, see the [Profile metadata type](#) in the *Metadata API Developer Guide*.



Note: Although source pulls don't include `package.xml` files, both pull requests and retrieve requests return profile information pertaining to everything reported by source tracking.

CHAPTER 10 Development

In this chapter ...

- [Develop Against Any Org](#)
- [Assign a Permission Set](#)
- [Ways to Add Data to Your Org](#)
- [Create Lightning Apps and Aura Components](#)
- [Create Lightning Web Components](#)
- [Create an Apex Class](#)
- [Create an Apex Trigger](#)
- [Run Apex Tests](#)

After you import some test data, you've completed the process of setting up your project. Now, you're ready to start the development process.

Create Source Files from the CLI

To add source files from the CLI, make sure that you're working in an appropriate directory. For example, if your package directory is called `force-app`, create Apex classes in `force-app/main/default/classes`. You can organize your source as you want underneath each package directory except for documents, custom objects, and custom object translations.

As of API version 45.0, you can build Lightning components using two programming models: Lightning Web Components and Aura Components. To organize your components' source files, your Aura components must be in the `aura` directory. Your Lightning web components must be in the `lwc` directory.

Execute one of these commands.

- `force:apex:class:create`
- `force:apex:trigger:create`
- `force:cmtt:create`
- `force:cmtt:field:create`
- `force:cmtt:record:create`
- `force:lightning:app:create`
- `force:lightning:component:create`
- `force:lightning:event:create`
- `force:lightning:interface:create`
- `force:lightning:test:create`
- `force:staticresource:create`
- `force:visualforce:component:create`
- `force:visualforce:page:create`

Consider using these two helpful optional flags:

Option	Description
<code>-d, --outputdir</code>	The directory for saving the created files. If you don't indicate a directory, your source is added to the current folder. To add the source to an existing directory, indicate the absolute or relative path. If you don't indicate an absolute or a relative

Option	Description
	path and the directory doesn't exist, the CLI attempts to create it for you.
-t, --template	Template used for the file creation.

 **Tip:** If you want to know more information about a command, run it with the `--help` option. For example, `sfdx force:apex:class:create --help`.

Edit Source Files

Use your favorite code editor to edit Apex classes, Visualforce pages and components, Lightning web components, and Aura components in your project. You can also make edits in your default scratch org and then use `force:source:pull` to pull those changes down to your project. For Lightning pages (FlexiPage files) that are already in your scratch org, use the shortcut to open Lightning App Builder in a scratch org from your default browser. Lightning Pages are stored in the `flexipages` directory.

To edit a FlexiPage in your default browser—for example, to edit the `Property_Record_Page` source—execute this command from the `flexipages` directory.

```
sfdx force:source:open -f Property_Record_Page.flexipage-meta.xml
```

If you want to generate a URL that loads the `.flexipage-meta.xml` file in Lightning App Builder but doesn't launch your browser, use the `--urlonly` | `-r` flag.

```
sfdx force:source:open -f Property_Record_Page.flexipage-meta.xml -r
```

SEE ALSO:

[Salesforce CLI Command Reference](#)

Develop Against Any Org

Regardless of the development model you're using, you eventually test and validate your changes in a non-source-tracked org. For those of you who don't use scratch orgs, we provide a similar experience for developing and unit testing in other environments, such as sandboxes.

You can use Salesforce CLI to retrieve and deploy metadata to non-source-tracked orgs with the same ease of pushing and pulling source to and from scratch orgs. And best of all, no extra conversion steps are required! After you retrieve the metadata, you don't have to convert it to source format. When you're ready to deploy it back to the org, you don't have to convert it to metadata format. If you're new to Salesforce CLI, [Salesforce DX Project Structure and Source File Format](#) explains the difference between source format and metadata format.

Using `force:source:retrieve`, you can retrieve the metadata you need in source format to your local file system (DX project). When your changes are ready for testing or production, you can use `force:source:deploy` to deploy your local files directly to a non-source-tracked org.

So, how do these source commands differ from the scratch org commands, `source:push` and `source:pull`? Because the changes aren't tracked, you retrieve or deploy all the specified metadata instead of only what's changed. The source you retrieve or deploy overwrites what's you have locally or in your org, respectively.

Not sure what metadata types are supported or which metadata types support wild cards in `package.xml`? See [Metadata Types](#) in the *Metadata API Developer Guide*.

Before You Begin

Before you begin, don't forget to:

- Create a Salesforce DX project that includes a manifest (`package.xml`). Run `force:project:create -n MyProject --manifest`.
- Authorize your non-source-tracked org. If connecting to a sandbox, edit your `sfdx-project.json` file to set `sfdcLoginUrl` to `https://test.salesforce.com` before you authorize the org. Don't forget to [create aliases](#) for your non-source-tracked orgs.

Metadata Names That Require Encoding on the Command Line

When retrieving or deploying metadata using the `--metadata` option, commas in metadata names require encoding to work properly.

Don't: `sfdx force:source:deploy -m "Profile:Standard User,Layout:Page,Console"`

Do: `sfdx force:source:deploy -m "Profile:Standard User,Layout:Page%2CConsole"`

Retrieve Source from a Non-Source-Tracked Org

Use the `force:source:retrieve` command to retrieve source from orgs that don't have source tracking, such as a sandbox or your production org. If you already have the source code and metadata in a VCS, you might be able to skip this step. If you're starting anew, you retrieve the metadata associated with the feature, project, or customization you're working on.

 **Note:** The `source:retrieve` command works differently from `source:pull` for scratch orgs. This command doesn't notify you if there's a conflict. Instead, the source you retrieve overwrites the corresponding source files in your local project. To retrieve metadata that's in the metadata format, use `force:mdapi:retrieve`.

You can retrieve metadata in source format using one of these methods:

- Specify a `package.xml` that lists the components to retrieve.
- Specify a comma-separated list of metadata component names.
- Specify a comma-separated list of source file paths to retrieve. You can use the source path option when source exists locally, for example, after you've done an initial retrieve.
- Specify a comma-separated list of package names.

If the comma-separated list you're supplying contains spaces, enclose the entire comma-separated list in one set of double quotes.

To Retrieve:	Command Example
All metadata components listed in a manifest	<code>sfdx force:source:retrieve -x path/to/package.xml</code>
Source files in a directory	<code>sfdx force:source:retrieve -p path/to/source</code>
A specific Apex class and the objects whose source is in a directory	<code>sfdx force:source:retrieve -p "path/to/apex/classes/MyClass.cls,path/to/source/objects"</code>
Source files in a comma-separated list that contains spaces	<code>sfdx force:source:retrieve -p "path/to/objects/MyCustomObject/fields/MyField.field-meta.xml, path/to/apex/classes"</code>
All Apex classes	<code>sfdx force:source:retrieve -m ApexClass</code>
A specific Apex class	<code>sfdx force:source:retrieve -m "ApexClass:MyApexClass"</code>
A layout name that contains a comma (Layout:Page, Console)	<code>sfdx force:source:retrieve -m "Layout:Page%2C Console"</code>
All the metadata related to a specific package or packages	<code>sfdx force:source:retrieve -n DreamHouse</code>

You can specify only one scoping parameter when retrieving metadata: `--metadata`, `--sourcepath`, or `--manifest`. If you indicate `--packagenames`, you can include one additional scoping parameter.

```
sfdx force:source:retrieve -n DreamHouse, -x manifest/package.xml
```

Deploy Source to a Non-Source-Tracked Org

Use the `force:source:deploy` command to deploy source to orgs that don't have source tracking, such as a sandbox or production org.

 **Note:** The `source:deploy` command works differently from `source:push` for scratch orgs. The source you deploy overwrites the corresponding metadata in your org, similar to running `source:push` with the `--force` option. To deploy metadata that's in the metadata format, use `force:mdapi:deploy`.

You can deploy metadata in source format using these methods:

- Specify a `package.xml` that lists the components to deploy
- Specify a comma-separated list of metadata component names
- Specify a comma-separated list of source file paths to deploy

If the comma-separated list you're supplying contains spaces, enclose the entire comma-separated list in one set of double quotes.

To Deploy:	Command Example
All components listed in a manifest	<code>sfdx force:source:deploy -x path/to/package.xml</code>
Source files in a directory	<code>sfdx force:source:deploy -p path/to/source</code>
A specific Apex class and the objects whose source is in a directory	<code>sfdx force:source:deploy -p "path/to/apex/classes/MyClass.cls,path/to/source/objects"</code>
Source files in a comma-separated list that contains spaces	<code>sfdx force:source:deploy -p "path/to/objects/MyCustomObject/fields/MyField.field-meta.xml, path/to/apex/classes"</code>
All Apex classes	<code>sfdx force:source:deploy -m ApexClass</code>
A specific Apex class	<code>sfdx force:source:deploy -m "ApexClass:MyApexClass"</code>
All custom objects and Apex classes	<code>sfdx force:source:deploy -m "CustomObject,ApexClass"</code>
All Apex classes and a profile that has a space in its name	<code>sfdx force:source:deploy -m "ApexClass, Profile:Content Experience Profile"</code>
A recently validated set of components without running Apex tests (often referred to as a quick deploy)	<code>sfdx force:source:deploy -q VALIDATEDDEPLOYREQUESTID</code> You can run this option after you have run tests, passed code coverage requirements, and performed a check-only deployment using the <code>-c</code> <code>--checkonly</code> option.
Even if the deployment contains warnings	<code>sfdx force:source:deploy -g</code>
Regardless of whether the deployment contains errors (not recommended if deploying to a production org)	<code>sfdx force:source:deploy -o</code>

Delete Non-Tracked Source

Use the `force:source:delete` command to delete components from orgs that don't have source tracking, such as sandboxes.

-  **Note:** Run this command from within a Salesforce DX project. To remove deleted items from scratch orgs, which have change tracking, use `force:source:push`.

If the source exists locally in a DX project, you can delete metadata by specifying the path to the source or by listing individual metadata components. If the comma-separated list you're supplying contains spaces, enclose the entire comma-separated list in one set of double quotes.

To Delete:	Command Example
Source files in a directory	<code>sfdx force:source:delete -p path/to/source</code>
A specific component, such as a FlexiPage	<code>sfdx force:source:delete -m "FlexiPage:Broker_Record_Page"</code>

Do You Want to Retain the Generated Metadata?

Normally, when you run some CLI commands, a temporary directory with all the metadata is created then deleted upon successful completion of the command. However, retaining these files can be useful for several reasons. You can debug problems that occur during command execution. You can use the generated `package.xml` when running subsequent commands, or as a starting point for creating a manifest that includes all the metadata you care about.

To retain all the metadata in a specified directory path when you run these commands, set the `SFDX_MDAPI_TEMP_DIR` environment variable:

- `force:source:deploy`
- `force:source:retrieve`
- `force:source:delete`
- `force:source:push`
- `force:source:pull`
- `force:source:convert`
- `force:org:create` (if your scratch org definition contains scratch org settings, not org preferences)

Example:

```
SFDX_MDAPI_TEMP_DIR=/users/myName/myDXProject/metadata
```

Assign a Permission Set

After creating your scratch org and pushing the source, you must sometimes give your users access to your application, especially if your app contains custom objects.

1. If needed, create the permission set in the scratch org.

- a. Open the scratch org in your browser.

```
sfdx force:org:open -u <scratch org username/alias>
```

- b. From Setup, enter *Perm* in the Quick Find box, then select **Permission Sets**.
 - c. Click **New**.
 - d. Enter a descriptive label for the permission set, then click **Save**.
 - e. Under Apps, click **Assigned Apps > Edit**.
 - f. Under Available Apps, select your app, then click **Add** to move it to Enabled Apps.
 - g. Click **Save**.

2. Pull the permission set from the scratch org to your project.

```
sfdx force:source:pull -u <scratch org username/alias>
```

3. Assign the permission set to one or more users of the org that contains the app:

```
sfdx force:user:permset:assign --permsetname <permset_name> --targetusername <username/alias>
```

The target username must have permission to assign a permission set. Use the `--onbehalfof` parameter to assign a permission set to non-administrator users.

```
sfdx force:user:permset:assign --permsetname <permset_name> --targetusername <admin-user>  
--onbehalfof <non-admin-user>
```

You can also assign permission set licenses to users using the `force:user:permsetlicense:assign` command. It works similarly to the `force:user:permset:assign` command.

SEE ALSO:

[Salesforce Help: Permission Sets](#)

[Salesforce Help: Permission Set Licenses](#)

Ways to Add Data to Your Org

Orgs for development need a small set of stock data for testing.

Sometimes, the stock data doesn't meet your development needs. Apex tests generally create their own data. Therefore, if Apex tests are the only tests you're running in a scratch org, you can probably forget about data for the time being. However, other tests, such as UI, API, or user acceptance tests, do need baseline data. Make sure that you use consistent data sets when you run tests of each type.

The following sections describe the Salesforce CLI commands you can use to populate your orgs. The commands you use depend on your current stage of development.

You can also use the `force:data:soql:query` CLI command to run a SOQL query against a scratch org. While the command doesn't change the data in an org, it's useful for searching or counting the data. You can also use it with other data manipulation commands. See the [SOQL and SOSL Reference Guide](#) for general SOQL limits that also apply when you use these commands.

Considerations for Scratch Orgs

Scratch orgs come with the same set of data as the edition on which they are based. For example, Developer Edition orgs typically include 10–15 records for key standard objects, such as Account, Contact, and Lead. These records come in handy when you're testing something like a new trigger, workflow rule, Lightning web component, Aura component, or Visualforce page.

force:data:tree Commands

The SObject Tree Save API drives the `force:data:tree` commands for exporting and importing data. The commands use JSON files to describe objects and relationships. The export command requires a SOQL query to select the data in an org that it writes to the JSON files. Rather than loading all records of each type and establishing relationships, the import command loads parents and children already in the hierarchy.

 **Note:** These commands are intended for developers to test with small datasets. The query for export can return a maximum of 2000 records. The files for import can have a maximum of 200 records.

force:data:bulk Commands

Bulk API drives the `force:bulk` commands for exporting a basic data set from an org and storing that data in source control. You can then update or augment the data directly rather than in the org from where it came. The `force:data:bulk` commands use

CSV files to import data files into scratch orgs or to delete sets of data that you no longer want hanging around. Use dot notation to establish child-to-parent relationships.

force:data:record Commands

Everyone's process is unique, and you don't always need the same data as your teammates. When you want to create, modify, or delete individual records quickly, use the `force:data:record:create|delete|get|update` commands. No data files are needed.

Example: Export and Import Data Between Orgs

Let's say you've created the perfect set of data to test your application, and it currently resides in a scratch org. You finished coding a new feature in a second scratch org, pushed your source code, and assigned the needed permission sets. Now you want to populate the scratch org with your perfect set of data from the other org. How? Read on!

SEE ALSO:

- [SObject Tree Request Body \(REST API Developer Guide\)](#)
- [Create Multiple Records \(REST API Developer Guide\)](#)
- [Create Nested Records \(REST API Developer Guide\)](#)
- [Salesforce Object Query Language \(SOQL\)](#)
- [Sample CSV File \(Bulk API 2.0 and Bulk API Developer Guide\)](#)
- [Salesforce CLI Command Reference](#)

Example: Export and Import Data Between Orgs

Let's say you've created the perfect set of data to test your application, and it currently resides in a scratch org. You finished coding a new feature in a second scratch org, pushed your source code, and assigned the needed permission sets. Now you want to populate the scratch org with your perfect set of data from the other org. How? Read on!

This use case refers to the Broker and Properties custom objects of the [Salesforce DreamHouse LWC application example on GitHub](#). Let's say that the scratch org from which you want to export data has the alias `org-with-data`. Similarly, the scratch org into which you want to import data has the alias `org-needs-data`. For the `org-with-data` scratch org, it's assumed that you've already:

- Created the Broker and Properties custom objects by pushing the DreamHouse source.
- Assigned the permission set.
- Populated the objects with the data.

For the `org-needs-data` scratch org, however, it's assumed that you've created the Broker and Properties objects and assigned the permission set but not yet populated the objects with data.

See the [README](#) in the `dreamhouse-1wc` GitHub repo for instructions on these prerequisite tasks.

1. Export the data from the `org-with-data` scratch org.

Use the `force:data:soql:query` command to fine-tune the SELECT query so that it returns the exact set of data you want to export. This command outputs the results to your terminal or command window, but it doesn't generate any files.



Note: Because the SOQL query is long, the sample command is broken up with backslashes for easier reading. You can still copy and paste the command into your terminal window and run it.

```
sfdx force:data:soql:query -u org-with-data --query \  
    "SELECT Id, Name, Title_c, Phone_c, Mobile_Phone_c, \  
        Email_c, Picture_c, \  
        (SELECT Name, Address_c, City_c, State_c, Zip_c, \  
            Price_c, Beds_c, Baths_c, Picture_c, \  
            Thumbnail_c, Description_c \  
        FROM Properties_r) \  
    FROM Broker_c"
```

- When you're satisfied with the SELECT statement, use it to export the data into a set of JSON files.

```
sfdx force:data:tree:export -u org-with-data --query \  
    "SELECT Id, Name, Title_c, Phone_c, Mobile_Phone_c, \  
        Email_c, Picture_c, \  
        (SELECT Name, Address_c, City_c, State_c, Zip_c, \  
            Price_c, Beds_c, Baths_c, Picture_c, \  
            Thumbnail_c, Description_c \  
        FROM Properties_r) \  
    FROM Broker_c" \  
    --prefix export-demo --outputdir sfdx-out --plan
```

The export command writes the JSON files to the `sfdx-out` directory and prefixes each file name with the string `export-demo`. The files include a plan definition file, which refers to the other files that contain the data, one for each exported object (Broker and Properties).

- Import the data into the `org-needs-data` scratch org by specifying the plan definition file.

```
sfdx force:data:tree:import -u org-needs-data \  
    --plan sfdx-out/export-demo-Broker_c-Property_c-plan.json
```

Use the `--plan` parameter to specify the full path name of the plan execution file generated by the `force:data:tree:export` command. Plan execution file names always end in `-plan.json`.



Example: Looking for a more complicated example? The [easy-spaces-lwc](#) sample app has a [data plan](#) showing how to import Accounts, related Contacts, and a 3-level deep custom object chain.

SEE ALSO:

[GitHub: Dreamhouse LWC sample repo](#)

[Salesforce CLI Command Reference](#)

Create Lightning Apps and Aura Components

To create Lightning apps and Aura components from the CLI, you must have an `aura` directory in your Salesforce DX project.

- In `<app dir>/main/default`, create the `aura` directory.
- Change to the `aura` directory.

3. In the `aura` directory, create a Lightning app or an Aura component.

```
sfdx force:lightning:app:create -n myAuraapp
```

```
sfdx force:lightning:component:create --type aura -n myAuraComponent
```

SEE ALSO:

[Create Lightning Web Components](#)

Create Lightning Web Components

To create a Lightning web component from the CLI, you must have an `lwc` directory in your Salesforce DX project.

1. In `<app dir>/main/default`, create the `lwc` directory.
2. Change to the `lwc` directory.
3. In the `lwc` directory, create the Lightning web component.

```
sfdx force:lightning:component:create --type lwc -n myLightningWebComponent
```

SEE ALSO:

[Create Lightning Apps and Aura Components](#)

[*Lightning Web Components Dev Guide: Introducing Lightning Web Components*](#)

Create an Apex Class

You can create Apex classes from the CLI.

By default, the class is created in the directory from which you run the command. The standard DX project template assumes you'll create your Apex classes in the `<package directory>/force-app/main/default/classes` directory. To create classes in this directory, change to it before running the command.

```
sfdx force:apex:class:create -n myclass
```

If you're in a different directory, indicate the `-d` parameter to specify the absolute or relative path to the directory where you want to save your Apex class files. If you don't indicate an absolute or a relative path and the directory doesn't exist, the CLI attempts to create it for you.

```
sfdx force:apex:class:create -n myclass -d ../force-app/main/default/classes
```

The command generates two files:

- `myclass.cls-meta.xml`—metadata format
- `myclass.cls`—Apex source file

By default, the command creates a shell for an Apex class. However, you can select different templates depending on what you're creating.

Template	Description	More Information in Apex Developer Guide
DefaultApexClass (default)	Standard Apex class.	Classes
ApexException	Use Apex built-in exceptions or create custom exceptions. All exceptions have common methods.	Exception Class and Built-in Exceptions
ApexUnitTest	Use the <code>@isTest</code> annotation to define classes and methods that only contain code used for testing your application.	isTest Annotation
InboundEmailService	Use email services to process the contents, headers, and attachments of inbound email.	Apex Email Service

Create an Apex Trigger

Use Apex triggers to perform custom actions before or after a change to a Salesforce record, such as an insertion, update, or deletion. You can create Apex triggers from the CLI.

By default, the trigger is created in the directory from which you run the command. The standard DX project template assumes you'll create your Apex triggers in the `<package directory>/force-app/main/default/triggers` directory. To create triggers in this directory, change to it before running the command.

```
sfdx force:apex:trigger:create -n mytrigger
```

If you're in a different directory, indicate the `-d` parameter to specify the absolute or relative path to the directory where you want to save your Apex class files. If you don't indicate an absolute or a relative path and the directory doesn't exist, the CLI attempts to create it for you.

```
sfdx force:apex:trigger:create -n mytrigger -d ../force-app/main/default/triggers
```

Generate a skeleton trigger file by executing `force:apex:trigger:create`.

- Use the `-s` parameter to specify the sObject associated with this trigger, such as `Account`.
- Use the `-e` parameter to specify the triggering events, such as `before delete` or `after insert`.

```
sfdx force:apex:trigger:create -n mytrigger -s Account -e 'before insert,after insert' -d ../force-app/main/default/triggers
```

The command generates two files.

- `mytrigger.trigger-meta.xml`—metadata format
- `mytrigger.trigger`—Apex trigger source file

SEE ALSO:

[Triggers \(Apex Developer Guide\)](#)

[Apex Triggers \(Trailhead Module\)](#)

Run Apex Tests

When you're ready to test changes to your source code, you can run Apex tests in an org using Salesforce CLI on the command line, Salesforce Extensions for VS Code, or from within third-party continuous integration tools, such as Jenkins or CircleCI.

See the [Salesforce CLI Command Reference](#) for the full list of command options.

Minimum User Permissions and Settings Required

The user running Apex tests must have these user permissions in the org:

- View Setup and Configuration
- API Enabled

Also ensure that the Enable Streaming API setting is enabled in the org's user interface. The setting is enabled by default.

See [User Permissions](#) and [Configure User Interface Settings](#) for details.

Run All Apex Tests

This command runs all Apex tests in the scratch org asynchronously. Rather than display the actual test results, the command outputs the `force:apex:test:report` command that you then run to view the full results.

```
sfdx force:apex:test:run
```

Run Specific Apex Tests

Use `--testlevel` to run a subset of tests. See [Understanding Deploy](#) in the *Apex Developer Guide* for the list of possible test level values.

```
sfdx force:apex:test:run --testlevel RunLocalTests
```

Use the `--synchronous` parameter to run tests synchronously. The command waits to display the test results until all tests have completed.

```
sfdx force:apex:test:run --synchronous --classnames TestA
```

Use parameters to list the test classes or suites to run, specify the output format, view code coverage results, and more. For example, the following command runs the `TestA` and `TestB` test classes, provides results in Test Anything Protocol (TAP) format, and requests code coverage results.

```
sfdx force:apex:test:run --classnames "TestA,TestB" --resultformat tap --codecoverage
```

Use the `--tests` parameter to run specific test methods using the standard notation `Class.method`. If you're testing a managed package, use `namespace.Class.method`. If you specify a test class without a method, the command runs all methods in the class. This example shows how to run two methods in the `TestA` class and all methods in the `TestB` class.

```
sfdx force:apex:test:run --tests "TestA.excitingMethod,TestA.boringMethod,TestB"
```

Here's the same example but with a namespace.

```
sfdx force:apex:test:run --tests "ns.TestA.excitingMethod,ns.TestA.boringMethod,ns.TestB"
```

You can specify either `--tests` or `--classnames` with `force:apex:test:run` but not both.

View Test Results

Run the `force:apex:test:report` command to view the results. The results include the outcome of individual tests, how long each test ran, and the overall pass and fail rate.

```
sfdx force:apex:test:report --testrunid 7074C00000988ax
```

Debug Apex

If you use Salesforce Extensions for Visual Studio Code (VS Code) for your development tasks, you have a choice of Apex Debugger extensions. Whichever debugger you chose, you set breakpoints in your Apex classes and step through their execution to inspect your code in real time to find bugs. You can run Apex tests in VS Code or on the command line.

Generate and View Apex Debug Logs

Apex debug logs can record database operations, system processes, and errors that occur when executing a transaction or running unit tests in any authenticated org. Enable the Debug Log in Salesforce Extensions for VS Code, then view the logs with VS Code or Salesforce CLI.

SEE ALSO:

[Test Anything Protocol \(TAP\)](#)

[Salesforce CLI Command Reference](#)

Debug Apex

If you use Salesforce Extensions for Visual Studio Code (VS Code) for your development tasks, you have a choice of Apex Debugger extensions. Whichever debugger you chose, you set breakpoints in your Apex classes and step through their execution to inspect your code in real time to find bugs. You can run Apex tests in VS Code or on the command line.

Apex Replay Debugger

Apex Replay Debugger is available for use without any additional licenses. See [Apex Replay Debugger](#) to configure and use it.

Apex Interactive Debugger

You must have at least one available Apex Debugger session in your Dev Hub org. To purchase more sessions for an org, contact your System Admin to [open a case](#).

- Performance Edition and Unlimited Edition orgs include one Apex Debugger session.
- Apex Debuggers sessions aren't available in Trial and Developer Edition orgs.
- You can purchase Apex Debugger sessions for Enterprise Edition orgs.

Enable the Apex Debugger in your scratch orgs by adding the `DebugApex` feature to your scratch org definition file:

```
"features": "DebugApex"
```

See [Apex Interactive Debugger](#) to configure and use it.

ISV Debugger (Salesforce Extensions for VS Code Only)

ISV Customer Debugger is part of the Apex Interactive Debugger (`salesforcedx-vscode-apex-debugger`) extension, so you don't have to install anything other than the Salesforce Extension Pack and its prerequisites. You can debug only sandbox orgs.

See [ISV Customer Debugger](#) in Salesforce Extensions for VS Code for details.

SEE ALSO:

[Build Your Own Scratch Org Definition File](#)

[Apex Debugger for Visual Studio Code](#)

Generate and View Apex Debug Logs

Apex debug logs can record database operations, system processes, and errors that occur when executing a transaction or running unit tests in any authenticated org. Enable the Debug Log in Salesforce Extensions for VS Code, then view the logs with VS Code or Salesforce CLI.

1. In Salesforce Extensions for VS Code, prepare the org to generate logs and configure the debugger.
 - a. Log in to the org.
 - b. For Replay Debugger, run **SFDX: Turn on Apex Debug Log for Replay Debugger**.
 - c. Create a launch configuration file for [Replay Debugger](#) or [Interactive Debugger](#).
2. After you run tests, get a list of the debug logs.

```
sfdx force:apex:log:list
```

APPLICATION START TIME	DURATION (MS)	ID	LOCATION	SIZE (B)	LOG USER	OPERATION	REQUEST
Unknown	1143	07L9Axx	SystemLog	23900	User	ApexTestHandler	Api
2017-09-05x	Success						

3. View a debug log by passing its ID to the `force:apex:log:get` command.

```
sfdx force:apex:log:get --logid 07L9A000000aBYGUA2
```

```
38.0
APEX_CODE,FINEST;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;SYSTEM,DEBUG;VALIDATION,INFO;VISUALFORCE,INFO;WAVE,INFO;WORKFLOW,INFO
15:58:57.3
(3717091)|USER_INFO|[EXTERNAL]|0059A000000TwPM|test-ktjauhgzinnp@example.com|Pacific
Standard Time|GMT-07:00
15:58:57.3 (3888677)|EXECUTION_STARTED
15:58:57.3
(3924515)|CODE_UNIT_STARTED|[EXTERNAL]|01p9A000000FmMN|RejectDuplicateFavoriteTest.acceptNonDuplicate()
15:58:57.3 (5372873)|HEAP_ALLOCATE|[72]|Bytes:3
...
```

SEE ALSO:

[Debug Log \(Apex Developer Guide\)](#)

CHAPTER 11 Build and Release Your App

In this chapter ...

- [Build and Release Your App with Metadata API](#)

When you finish writing your code, the next step is to deploy it. We offer different deployment options based on your role and needs as a customer, system integrator, or independent software vendor (ISV) partner.

To learn about the benefits of the different development models, review these Trailhead modules:

- [Determine Which Application Lifecycle Management Model Is Right for You](#)
- [Application Lifecycle and Development Models](#)
- [Package Development Model](#)
- [Quick Start: Unlocked Packages](#)
- [Unlocked Packages for Customers](#)

Based on your adoption readiness, review this table for your recommended options:

Release Options To Deliver Apps

Customers and Non-ISV Partners

- [Unlocked package](#)

An unlocked package is for customers who want to organize metadata into a package and deploy the metadata (via packages) to different orgs.



Note: An unlocked package offers a super-set of the capabilities of an unmanaged package. Therefore, unmanaged packages aren't included in this list.

For more information, see [Unlocked Packages](#).

- [Change sets, or org development via Salesforce CLI](#)

ISV Partners

- [Second-Generation \(2GP\) managed package](#)

If you are an ISV that develops apps and lists them on AppExchange, Salesforce recommends managed packages.

Second-generation managed packaging (2GP) ushers in a new way for AppExchange partners to develop, distribute, and manage their apps and metadata. You can use 2GP to organize your source, build small modular packages, integrate with your version control system, and better utilize your custom Apex code. You can execute all packaging operations via Salesforce CLI, or automate them using scripts.

For more information on 2GP managed packages, see [Second-Generation Managed Packages](#)

- [First-Generation \(1GP\) managed package](#)

Similar to 2GP, 1GP managed packages are used by ISVs to distribute their business apps to customers via AppExchange.

If you are familiar with first-generation managed packages and want to learn more about how 1GP differs from 2GP, see [Comparison of First- and Second-Generation Managed Packages](#).

For more information on 1GP managed packages, see [Create a First-Generation Managed Package using Salesforce DX](#).

Not Ready for Package Development?

If you or your team isn't ready for package development, you can continue to use change sets, or try to the org development model, where you deploy changes using Salesforce CLI. For more information, see [Build and Release Your App with Metadata API](#).

Build and Release Your App with Metadata API

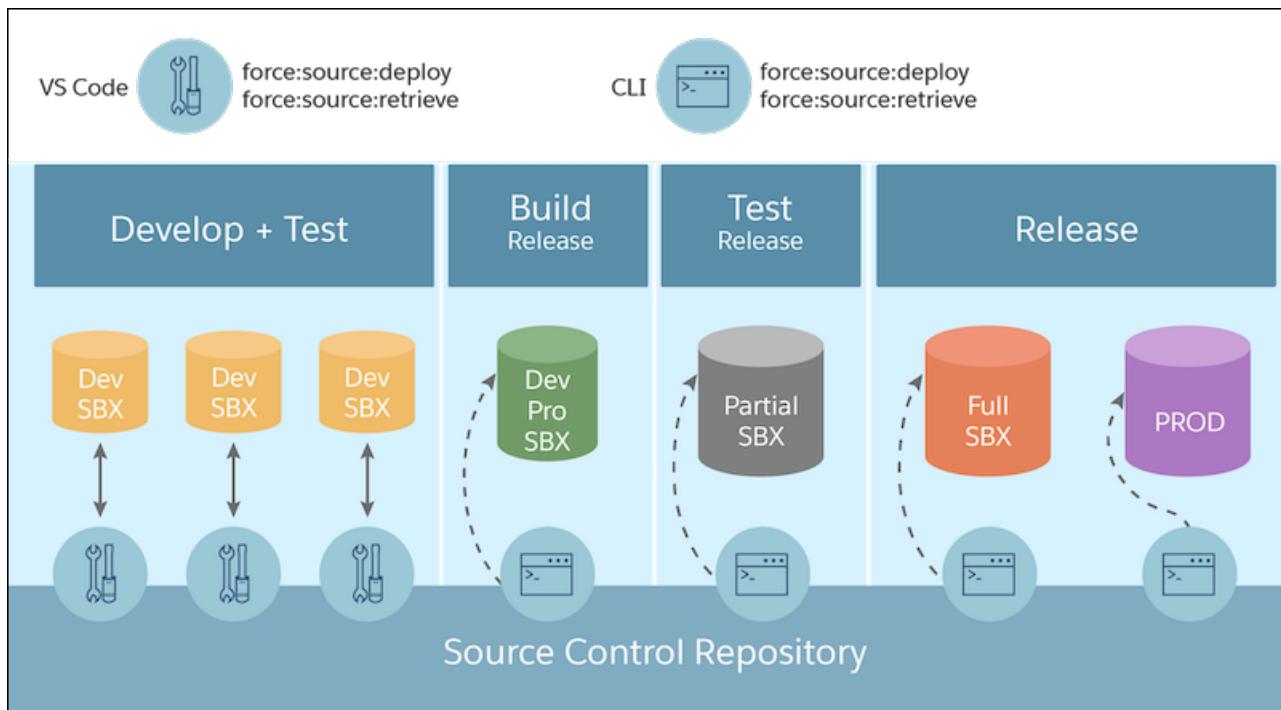
Develop and test your app in your sandboxes. Use Salesforce CLI or Salesforce Extensions for VS Code to retrieve and deploy your source. This development work flow is called the org development model.

Develop and Test in a Sandbox Using the Org Development Model

Similar to change sets, the release artifact is a set of changed metadata to update in the production org. You can use develop, test, and deploy your changes using the `source` commands. If you want to know more about this development model, see the [Org Development Model](#) module in Trailhead.

Development and Release Environments

- Develop and test:** Each team member has their own Developer sandbox to create their assigned customization. Developer sandboxes contain no production data.
- Build release:** Team members each migrate their customizations from their respective developer sandboxes to a shared Developer Pro sandbox for integration. Developer Pro sandboxes don't contain production data, but you can seed them with testing data.
- Test release:** For user-acceptance testing, the team uses a Partial sandbox to create a complete replica of production.
- Release:** After the release is in production, the team can use the Full sandbox to train users without the risk of altering production data. A Full sandbox includes a copy of production data.



What Tools Do I Need?

Tool	Description
Salesforce DX project	The Salesforce DX project contains the metadata and source files that comprise your changes. A DX project has a specific project structure and source format. In addition to source files, the project contains a configuration file, <code>sfdx-project.json</code> . This file contains project information and enables you to leverage Salesforce DX tools for many of your development tasks.
Deployment artifact	After testing the changes, you create the deployment artifact, a <code>.zip</code> file that contains changed files to deploy. Deploy the release artifact to the full (staging) sandbox first, and then finally to production. You can think of the deployment artifact as the inbound change set. The changes don't take effect until they are deployed.
Source control system	All changes are merged and stored in a source control system, which contains the Salesforce DX project.
Salesforce CLI	You can use Salesforce CLI for every phase of the org development life cycle. It improves productivity by providing a single interface for all your development, testing, and automation use cases.
Salesforce Extensions for VS Code	Salesforce Extensions for VS Code is built on top of Salesforce CLI and Visual Studio Code. Together, they are an integrated development environment for custom development on Lightning Platform. You can run Salesforce CLI commands directly from the command palette or terminal.
Change management mechanisms	It's still important to capture your changes externally using formal change-tracking tools, such as a change list, a deployment run list, and other project management tools.

Considerations for Deploying Apex Code

To deploy Apex to production, unit tests of your Apex code must meet coverage requirements. Code coverage indicates how many executable lines of code in your classes and triggers are covered by your test methods. Write test methods to test your triggers and classes, and then run those tests to generate code coverage information.

If you don't specify a test level when initiating a deployment, the default test execution behavior depends on the contents of your deployment package.

- If your deployment package contains Apex classes or triggers, when you deploy to production, all tests are executed, except tests that originate from a managed package.
- If your package doesn't contain Apex code, no tests are run by default.

You can run tests for a deployment of non-Apex components. You can override the default test execution behavior by setting the test level in your deployment options. Test levels are enforced regardless of the types of components present in your deployment package.

We recommend that you run all local tests in your development environment, such as a sandbox, before deploying to production. Running tests in your development environment reduces the number of tests required in a production deployment.

[Develop and Test Changes Locally](#)

Develop changes in source format, deploying to and retrieving from your Developer sandbox.

[Build and Test the Release Artifact](#)

After your team has finished its development tasks, transition to the build release phase to integrate your changes in a Developer Pro sandbox. Then build the release artifact.

[Test the Release Artifact in a Staging Environment](#)

Stage the changes and run regression tests in a Full sandbox.

[Release Your App to Production](#)

Now that all your tests have passed in the Full sandbox, you're ready to deploy to production.

[Cancel a Metadata Deployment](#)

You can cancel a metadata deployment from the CLI and specify a wait time for the command to complete.

SEE ALSO:

[Metadata API Developer Guide](#)

[Salesforce CLI Command Reference](#)

Develop and Test Changes Locally

Develop changes in source format, deploying to and retrieving from your Developer sandbox.

These steps provide the high-level work flow.

1. Create the source control repository.
2. Create a DX project.
3. Add the DX project files to your source control repository.
4. Authorize the Developer sandbox.
5. Perform development tasks in your developer sandbox.
6. Retrieve the changes from the developer sandbox. If you have a few changes, you can indicate a comma-separated list of metadata component names. If you have many changes, you can use a manifest (`package.xml`).

```
sfdx force:source:retrieve --manifest path/to/package.xml
```

7. Commit the changes to the source control repository.

Next: Deploy all changes the team has made to the Developer Pro sandbox, then test those changes.

SEE ALSO:

[Metadata API Developer Guide](#)

[Salesforce CLI Command Reference](#)

Build and Test the Release Artifact

After your team has finished its development tasks, transition to the build release phase to integrate your changes in a Developer Pro sandbox. Then build the release artifact.

Here are the high-level steps in the work flow to create the release artifact.

1. Pull the changes from the repo so your local project contains all the changes your team has made.
2. Authorize the Developer Pro sandbox.
3. Run the deploy command that mimics what you'll deploy to production, for example:

```
sfdx force:source:deploy --manifest manifest/package.xml --targetusername dev-pro-sandbox
 \
--testlevel RunSpecifiedTests --runtests TestMyCode
```

4. Open the sandbox.
5. Perform testing.
6. If the testing passes, continue to the test release phase where you deploy the release artifact to the partial sandbox. Then perform user-acceptance testing.

After the testing passes, move to the release phase and perform regression tests in the Full sandbox.

Test the Release Artifact in a Staging Environment

Stage the changes and run regression tests in a Full sandbox.

After you have made all your changes based on the integration testing, the next step is to stage the changes in a Full sandbox. The process of deploying changes to the Full sandbox is similar to the process you used to deploy changes to your Developer Pro sandbox. This phase includes regression testing and mimics how you release the changes to production.

These steps provide the high-level work flow.

1. Authorize the Full sandbox.
2. (Optional) If you made any changes based on your testing in the Developer Pro sandbox, create a release artifact (.zip). If not, use the existing release artifact.
3. To validate the deployment without saving the components in the target org, run all local (regression) tests. A validation enables you to verify the results of tests that would be executed during a deployment, but doesn't commit any changes.

```
sfdx force:source:deploy --checkonly --manifest manifest/package.xml --targetusername
full-sandbox --testlevel RunLocalTests
```

4. Test the actual production deployment steps in the staging sandbox. Set up the same quick deploy that you plan to execute against the production org.

```
sfdx force:source:deploy --checkonly --manifest manifest/package.xml --targetusername
full-sandbox --testlevel RunSpecifiedTests TestLanguageCourseTrigger
```

This command returns a job ID that you reference in the quick deploy.

5. Next, test the quick deploy using the job ID returned in the previous step.

```
sfdx force:source:deploy --targetusername full-sandbox --validateddeployrequestid jobID
```

After you run the quick deploy, you have 10 days to perform the deployment to production.

Release Your App to Production

Now that all your tests have passed in the Full sandbox, you're ready to deploy to production.

1. In your deployment run list, complete any pre-deployment tasks.
2. Authorize your production org.
3. Set up the quick deploy.

```
sfdx force:source:deploy --checkonly --sourcepath force-app --targetusername prod-org  
--testlevel RunLocalTests
```

This command returns a job ID that you reference in the quick deploy.

4. After the tests are run, verify that all the Apex tests have passed. Be sure that the tests cover at least 75% of the code being deployed.
5. Run the quick deploy:

```
sfdx force:source:deploy --targetusername prod-org --validateddeployrequestid jobID
```

6. Open the production org, then perform any post-deployment tasks listed in the deployment run list.

SEE ALSO:

[Metadata API Developer Guide](#)
[Salesforce CLI Command Reference](#)

Cancel a Metadata Deployment

You can cancel a metadata deployment from the CLI and specify a wait time for the command to complete.

To cancel your most recent deployment, run `force:source:deploy:cancel`. You can cancel earlier deployments by adding the `-i` (JOBID) parameter to specify the deployment that you want to cancel.

```
$ sfdx force:source:deploy:cancel -i <jobid>
```

The default wait time for the cancel command to complete and display its results in the terminal window is 33 minutes. If the command isn't completed by the end of the wait period, the CLI returns control of the terminal window to you. You can adjust the wait time as needed by specifying the number of minutes in the `-w` (WAIT) parameter, as shown in the following example:

```
$ sfdx force:source:deploy:cancel -w 20
```

Curious about the status of a canceled deployment? Run a deployment report.

```
$ sfdx force:source:deploy:report
```

SEE ALSO:

[Metadata API Developer Guide](#)
[Salesforce CLI Command Reference](#)

CHAPTER 12 Create a First-Generation Managed Package using Salesforce DX

In this chapter ...

- [Build and Release Your App with Managed Packages](#)
- [View Information About a Package](#)

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

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

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

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

Build and Release Your App with Managed Packages

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

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.

```
sfdx force:org:open --targetusername me@my.org --path lightning/setup/Package/home
```

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

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

Packaging Checklist

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

Deploy the Package Metadata to the Packaging Org

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

Create a Beta Version of Your App

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

Install the Package in a Target Org

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

Create a Managed Package Version of Your App

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

SEE ALSO:

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

https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_ws_retrieve_man_pack.htm

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.

```
sfdx force:org:create -s -f 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.

```
sfdx force:source:convert --outputdir mdapi_output_dir --packagename 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 `--packagename` 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.

```
sfdx auth:web:login --setalias MyPackagingOrgAlias
```

You can also authenticate with an OAuth client ID: `sfdx auth:web:login --clientid oauth_client_id`

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

```
sfdx force:mdapi:deploy --deploydir mdapi_output_dir --targetusername me@example.com
```

The `--targetusername` 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 `--zipfile` 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 `force:mdapi:deploy`, 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 `force:mdapi:deploy` again.

```
sfdx force:mdapi:deploy:report
```

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](#)

https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_exclude_source.htm

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.

```
sfdx auth:web:login --targetusername me@example.com
```

2. Create the beta version of the package.

```
sfdx force:package1:version:create --packageid 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 `--installationkey` 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 `--installationkey`, 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 (--managedreleased 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 *ISVforce 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.

```
sfdx force:org:create -s -f config/project-scratch-def.json
```

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

METADATAPACKAGEVERSIONID	METADATAPACKAGEID	NAME	VERSION	RELEASESTATE	BUILDDNUMBER
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.

```
sfdx force: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 `--targetusername` parameter. If the package is protected by an installation key, supply the key with the `--installationkey` parameter.

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

SEE ALSO:

[ISVforce Guide](#)

[Salesforce CLI Command Reference](#)

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.

```
sfdx auth:web:login --instanceurl https://test.salesforce.com --setdefaultusername org_alias
```

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

```
sfdx force:package1:version:list --packageid 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. The *ISVforce Guide* has information on editable components.

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

```
sfdx force:package1:version:create --packageid 033xx00000007oi --name "Spring 17" --description "Spring 17 Release" --version 3.2 --managedreleased
```

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 `--installationkey`. 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 `force:package1:version:list`.

SEE ALSO:

[Salesforce CLI Command Reference](#)

[ISVforce Guide](#)

[Link a Namespace to a 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.

- From the project directory, run this command, supplying a package version ID.

`force:package1:version:display -i 04txx000000069yAAA`

The output is similar to this example.

METADATAPACKAGEVERSIONID	METADATAPACKAGEID	NAME	VERSION	RELEASESTATE	BUILDDNUMBER
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.

- From the project directory, run the `list` command.

`force: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	BUILDDNUMBER
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.

CHAPTER 13 Second-Generation Managed Packages

In this chapter ...

- [What's a Second-Generation Managed Package?](#)
- [Before You Create Second-Generation Managed Packages](#)
- [Workflow for Second-Generation Managed Packages](#)
- [Components Available in Managed Packages](#)
- [Develop Second-Generation Managed Packages](#)
- [Install and Uninstall Second-Generation Managed Packages](#)
- [Prepare to Distribute Your Second-Generation Managed Package](#)
- [Push a Package Upgrade for Second-Generation Managed Packages](#)
- [Advanced Features for Second-Generation Managed Packages](#)
- [Best Practices for Second-Generation Managed Packages](#)
- [Manage Licenses for Managed Packages](#)
- [Manage Features in Second-Generation Managed Packages](#)

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

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

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

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

Second-Generation Managed Packages

- Gaps Between First-Generation and Second-Generation Managed Packaging

What's a Second-Generation Managed Package?

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

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

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

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

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

Why Switch to Second-Generation Managed Packaging?

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

Comparison of First- and Second-Generation Managed Packages

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

Why Switch to Second-Generation Managed Packaging?

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

Source-Driven Development

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

Minimal Interaction with Salesforce Orgs

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

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

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

API- and CLI-first Model

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

Flexible Versioning

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

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

One Namespace Shared Across Multiple Packages

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

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

Declarative Dependencies

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

Simplified Patch Versioning

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

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

Avoid Having to Migrate Customers in the Future

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

Comparison of First- and Second-Generation Managed Packages

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

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

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

Before You Create Second-Generation Managed Packages

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

Did you?

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

Developers who work with managed 2GP packages need the correct permission set in the Dev Hub org. Developers need either the System Administrator profile or the Create and Update Second-Generation Packages permission. For more information, see [Add Salesforce DX Users](#).

The maximum number of managed 2GP package versions that you can create from a Dev Hub per day is the same as your daily scratch org allocation. To request a limit increase, contact Salesforce Partner Support.

Scratch orgs and packages count separately, so creating a second-generation managed package doesn't count against your daily scratch org limit. To view your scratch org limits, use the CLI:

```
sfdx force:limits:api:display -u <Dev Hub username or alias>
```

For more information on scratch org limits, see [Scratch Orgs](#).

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

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

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

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

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

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

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

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

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

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

[Which Package Dependencies Work with Second-Generation Managed Packages?](#)

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

Know Your Orgs for Second-Generation Managed Packages

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

Choose Your Dev Hub Org

Use the Dev Hub org for these purposes.

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

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

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

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

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

Namespace Org

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

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

Other Orgs

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

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

SEE ALSO:

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

Namespaces for Second-Generation Managed Packages

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

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

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

- You can develop more than one managed 2GP package with the same namespace but you can associate each managed 2GP package with only a single namespace.
- If you work with more than one namespace, we recommend that you set up one project for each namespace.

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

Create and Register Your Namespace for Second-Generation Managed Packages

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

To create a namespace:

1. Sign up for a new Developer Edition org.
2. In Setup, enter *Package Manager* in the Quick Find box, and select **Package Manager**.
3. In Developer Settings, click **Edit**, and under Change Developer Settings, click **Continue**.
4. In Namespace Prefix enter a namespace, and select **Check Availability**.
5. For **Package to be managed**, select **None**, then click **Review My Selections**.
6. Review your selections, and then click **Save**.

To register a namespace:

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

Key Concepts in Second-Generation Managed Packaging

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

What's the difference between...	Details
An app, a package, and metadata?	An app is a set of features that you're developing for your customers. Metadata is the technical representation of Salesforce features like custom objects, Apex classes, and Lightning pages. An app is composed of a set of metadata. A package is the container for your app's Salesforce metadata. Packages are used to distribute the app that you build. When a package is installed in an org, the app's metadata is deployed to the org.
A package and package version?	Your app, and thus your package, evolves over time. Whenever you change, add, or remove the metadata in your package, you create a new package version. Each package version is an immutable artifact, a static snapshot of your metadata at a specific point in time. So while your package evolves continuously, you take snapshots of it when it's in a stable state in the form of a package version. Technically speaking, when we say "Install a package," we really mean install a specific package version.

What's the difference between...	Details
A package install and package upgrade?	<p>A package install refers to the first time a version of the package is installed in an org. When a package is installed, the metadata associated with the package is deployed into the org.</p> <p>A package upgrade refers to the installation of new package version in an org that already has a previous version of the package installed. During a package upgrade, metadata changes are deployed. An upgrade can include deploying new metadata, modifying existing metadata, or deleting or deprecating metadata. At any given point in time, an org can only ever have one version of a package installed in that org.</p>

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

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

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

- Manageability Rules
- Package Ancestry
- Package Upgrades

Manageability Rules

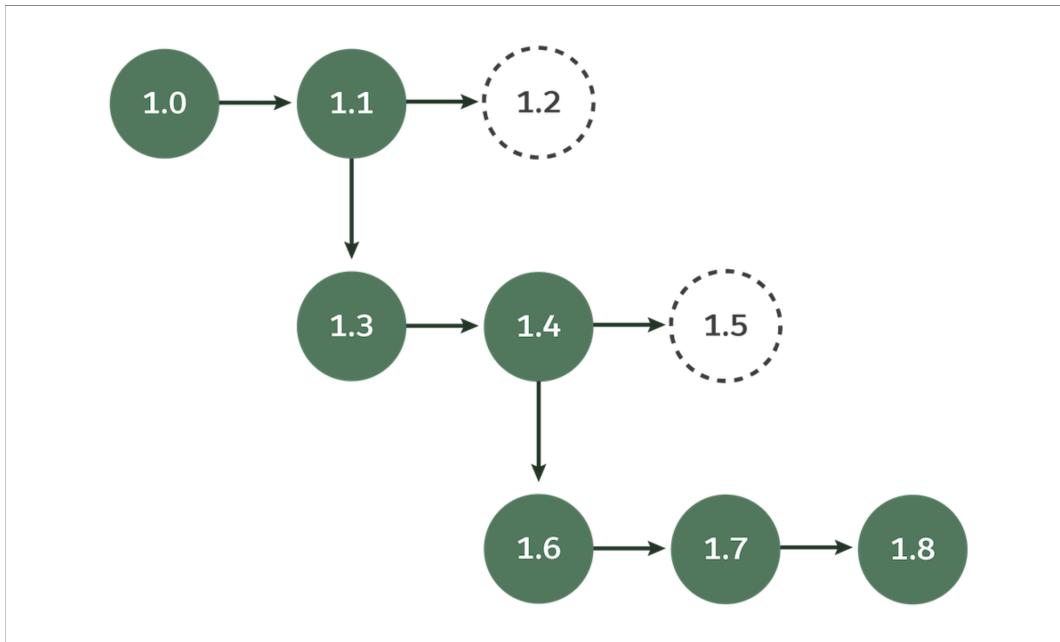
Each metadata component that you include in a managed 2GP package has certain rules that determine its behavior in a subscriber org. Manageability rules determine whether you, or the subscriber, can edit or remove components after the package version is installed in a subscriber's org.

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

Package Ancestry

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

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



How Manageability Rules and Ancestry Impact Package Upgrades

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

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

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

Which Package Dependencies Work with Second-Generation Managed Packages?

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

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

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

Can a managed 2GP package depend on a managed 1GP package?	Yes
Can a managed 2GP package depend on another managed 2GP package?	Yes
Can a managed 1GP package depend on a managed 2GP package?	No. This dependency isn't supported, and we block the installation of managed 2GP packages in managed 1GP packaging orgs. We can override this behavior on an individual basis. To share your scenario and request an override, log a case with Salesforce Partner Support . We're investigating how to support this dependency scenario more broadly.
Can a managed 1GP package depend on another managed 1GP package?	Yes
Can a managed 2GP package depend on an unlocked package?	Not recommended. However, you can build an unlocked package that depends on a managed 1GP or managed 2GP package.

Workflow for Second-Generation Managed Packages

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

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

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

1. Create a DX project.

```
sfdx force:project:create --outputdir expense-manager-workspace --projectname expenser-app
```

2. Authorize the Dev Hub org, and create a scratch org.

```
sfdx auth:web:login --setdefaultdevhubusername
```

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

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

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

```
sfdx force:org:create --definitionfile config/project-scratch-def.json --targetusername MyScratchOrg1
```

4. Verify that all package components are in the project directory where you want to create a package. If you're trying out the exact steps and commands in this workflow, you must add at least one piece of metadata before you continue to the next step.
5. In the `sfdx-project.json` file, specify a namespace using the `namespace` attribute. For example: `"namespace": "exp-mgr"`
If you specified a namespace when you created a Salesforce DX project in step one, you can skip this step. Before adding a namespace, make sure that you've linked the [namespace](#) to your Dev Hub org.
6. From the Salesforce DX project directory, create the package.

```
sfdx force:package:create --name "Expense Manager" --path force-app --packagetype Managed
```

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

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

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

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

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

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

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

```
sfdx force:package:version:create --package "Expense Manager" --codecoverage
--installationkey test1234 --wait 10
```

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

```
sfdx force:package:install --package "Expense Manager@0.1.0-1" --targetusername MyTestOrg1
--installationkey test1234 --wait 10 --publishwait 10
```

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

```
sfdx force:org:open --targetusername MyTestOrg1
```

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

Components Available in Managed Packages

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

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. 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. 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. In most cases, removing components from a package version marks the component as deprecated, and doesn't hard delete the component from the subscriber org. These deprecated components can be deleted by the admin of the subscriber org. For details about which managed 2GP components are deprecated, see Remove Metadata Components from Second-Generation Managed Packages . To request access to this feature, log a support case in the Salesforce Partner Community . 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. 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. This lets the subscriber 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 Managed Packages

[Account Relationship Share Rule](#)

[Action](#)

[Action Link Group Template](#)

[Action Plan Template](#)

[Activation Platform](#)

[AI Application](#)

[AI Application Config](#)

[AIUsecaseDefinition](#)

[Analytics](#)

[Apex Class](#)

[Apex Sharing Reason](#)

[Apex Sharing Recalculation](#)

[Apex Trigger](#)

[Application](#)

[Application Subtype Definition](#)

[Article Type](#)

AssessmentQuestion
AssessmentQuestionSet
Batch Calc Job Definition
Batch Process Job Definition
Benefit Action
Branding Set
Briefcase Definition
Building Energy Intensity Record Type Configuration
Business Process Group
Business Process Type Definition
Call Center
Care Benefit Verify Settings
Care Limit Type
Care Request Configuration
Care System Field Mapping
Chatter Extension
Community Template Definition
Community Theme Definition
Compact Layout
Connected App
Contract Type
Conversation Vendor Info
CORS Allowlist
CSP Trusted Site
Custom Application
Custom Button or Link
Custom Console Components
Custom Field on Standard or Custom Object
Custom Field on Custom Metadata Type
Custom Help Menu
Custom Index
Custom Label
Custom Metadata Records
Custom Metadata Types
Custom Notification Type
Custom Object Translation
Custom Object

Custom Permission
Custom Setting
Custom Tab
Dashboard
Data Classification on Custom Fields
Data Connector Ingest API
Data Connector S3
Data Stream Definition
Data Source
Data Source Object
Decision Matrix Definition
Decision Matrix Definition Version
Decision Table
Decision Table Dataset Link
Discovery AI Model
Discovery Goal
Discovery Story
Document
Document Generation Setting
Eclair GeoData
Email Template (Classic)
Email Template (Lightning)
Embedded Service Config
Embedded Service Menu Settings
Entitlement Process
Entitlement Template
ESignature Config
ESignature Envelope Config
Experience Builder Template
Experience Builder Theme
Explainability Action Definition
Explainability Action Version
Explainability Message Template
Expression Set Definition
Expression Set Definition Version
Expression Set Object Alias
Embedded Service Deployment

External Actions
External Credential
External Data Connector
External Data Source
External Services
Field Set
Field Source Target Relationship
Flow
Flow Category
Flow Definition
Flow Test
Fuel Type
Fuel Type Sustainability Unit of Measure
Folder
Gateway Provider Payment Method Type
Global Picklist
Home Page Component
Home Page Layout
Identity Verification Proc Def
Inbound Network Connection
Letterhead
Lightning Application
Lightning Bolt
Lightning Component
Lightning Event
Lightning Interface
Lightning Message Channel
Lightning Page
List View
Live Chat Sensitive Data Rule
Loyalty Program Setup
Marketing App Extension
Market Segment Definition
Milestone Type
MktCalculatedInsightsObjectDef
MktDataTranObject
ML Prediction Definition

ML Recommendation Definition
Named Credential
Recommendation Strategy
Sustainability UOM
Sustainability UOM Conversion
Object Source Target Map
OcrSampleDocument
OcrTemplate
Outbound Network Connection
Page Layout
Path Assistant
Payment Gateway Provider
Permission Set
Permission Set Groups
Platform Cache
Platform Event Channel
Platform Event Channel Member
Platform Event Subscriber Configuration
Process
Prompts (In-App Guidance)
Quick Action
Record Action Deployment
Record Type
RedirectWhitelistUrl
Referenced Dashboard
Remote Site Setting
Report
Report Type
Reporting Snapshot
Salesforce IoT
Slack App
Service Catalog Category
Service Catalog Item Definition
Service Catalog Fulfillment Flow
Stationary Asset Environmental Source Record Type Configuration
Static Resource
Streaming App Data Connector

[Timeline Object Definition](#)[Timesheet Template](#)[Translation](#)

Add translations to your managed packages.

[UI Object Relation Config](#)[Validation Rule](#)[Vehicle Asset Emissions Source Record Type Configuration](#)[View Definition](#)[Virtual Visit Config](#)[Visualforce Component](#)[Visualforce Page](#)[Wave Application](#)[Wave Component](#)[Wave Dataflow](#)[Wave Dashboard](#)[Wave Dataset](#)[Wave Lens](#)[Wave Recipe](#)[Wave Template Bundle](#)[Wave Xmd](#)[Web Store Template](#)[Workflow Email Alert](#)[Workflow Field Update](#)[Workflow Outbound Message](#)[Workflow Rule](#)[Workflow Task](#)

Account Relationship Share Rule

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- None

Both Package Developer and Subscriber Can Edit

- Action layout
- Predefined values for action fields

Neither Package Developer or Subscriber Can Edit

- All attributes except action layout and predefined values for action fields

Action Link Group Template

Can This Component Be Updated or Removed After Package Version Promotion?

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

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: ActionLinkGroupTemplate

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

Documentation

Salesforce Help: [Action Link Templates](#)

Action Plan Template

Can This Component Be Updated or Removed After Package Version Promotion?

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](#)

- 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](#)

Activation Platform

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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 supported:

- 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](#)

AI Application

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Type

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

- Status
- ExternalId
- MExternalId

[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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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](#)

- 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 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.

Can This Component Be Updated or Removed After Package Version Promotion?

Component Is Updated During Package Upgrade	Yes (Analytics Dataflow only). All other analytics components can't be updated.
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (Analytic snapshot only). Supported in managed 2GP packages only. All other analytics components can't be removed.
Component Has IP Protection	No

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](#).

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

More Information

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

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

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

Apex Class

Can This Component Be Updated or Removed After Package Version Promotion?

Component Is Updated During Package Upgrade	Yes
Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	Yes (if not set to <code>global</code> access). Supported in both 1GP and 2GP packages.
Component Has IP Protection	Yes

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

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 upload 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

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

ISVforce Guide: [About API and Dynamic Apex Access in Packages](#)

ISVforce Guide: [Using Apex in Group and Professional Editions](#)

Apex Sharing Reason

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Sharing Recalculation

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

- Apex Class

Neither Package Developer or Subscriber Can Edit

- None

Apex Trigger

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- API Version
- Code

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- Name

More Information

Feature Name

Metadata Name: ApexTrigger

Component Type in 1GP Package Manager UI: Apex Trigger

Documentation

Apex Developer Guide: [Triggers](#)

Application

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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)

Application Subtype Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#)

- Label
- Developer Name
- Description
- Application Usage Type

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

- None

More Information

Feature Name

Metadata Name: ApplicationSubtypeDefinition

Article Type

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- Description
- Label
- Plural Label
- Starts with a Vowel Sound

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

- Available for Customer Portal
- Channel Displays
- Default Sharing Model
- Development Status
- Enable Divisions
- Grant Access Using Hierarchy

- Search Layouts
- Neither Package Developer or Subscriber Can Edit
- Name

AssessmentQuestion

Can This Component Be Updated or Removed After Package Version Promotion?

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](#)

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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](#)

Batch Calc Job Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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
-----------------------------	-----------------------

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Branding Set

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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: BldgEnergyIntensityCnfg

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: Map Building Energy Intensity Record Type Configurations*
- *Salesforce Help: Benchmark Building Energy Intensity Data*

Business Process Group

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#)

- Label
- Developer Name
- Description
- Application Usage Type

Neither Package Developer or Subscriber Can Edit

- None

More Information

Feature Name

Metadata Name: BusinessProcessTypeDefinition

Call Center

Can This Component Be Updated or Removed After Package Version Promotion?

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	No

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

More Information

Feature Name

Metadata Name: CallCenter

Component Type in 1GP Package Manager UI: Call Center

Documentation

Metadata API Developer Guide: [CallCenter](#)

Care Benefit Verify Settings

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- External ID Field
- IsActive

- 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](#)

Chatter Extension

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Community Template Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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](#)

Connected App

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Salesforce DX 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.
- You can add an existing connected app (one created before Summer '13) to a managed package. You can also combine new and existing connected apps in the same managed package.

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

SEE ALSO:

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

Contract Type

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Metadata API Developer Guide: [ContractType](#)

Conversation Vendor Info

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

CORS Allowlist

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

More Information

Feature Name

Metadata Name: CustomApplication

Documentation

Metadata API Developer Guide: [CustomApplication](#)

Custom Button or Link

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

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.

Can This Component Be Updated or Removed After Package Version Promotion?

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

Can This Component Be Updated or Removed After Package Version Promotion?

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 2P 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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Custom Help Menu

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Custom Index

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

More Information

Feature Name

Metadata Name: CustomIndex

Component Type in 1GP Package Manager UI: Custom Index

Considerations When Packaging

Subscribers can remove custom index using the Metadata API only.

Documentation

Best Practices for Deployments with Large Data Volumes: [Indexes](#)

Custom Label

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Component Type in 1GP Package Manager UI: [Delete entry if component isn't packageable in 1GP]

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 ISVforce guide.

Documentation

Metadata API Developer Guide: [CustomLabels](#)

Custom Metadata Records

Can This Component Be Updated or Removed After Package Version Promotion?

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	Yes

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

More Information

This component can be marked as protected. For more details, see [Protected Components](#) in the ISVforce guide.

Custom Metadata Types

Can This Component Be Updated or Removed After Package Version Promotion?

Component Is Updated During Package Upgrade	Yes
---	-----

Second-Generation Managed Packages

Custom Notification Type

Subscriber Can Delete Component From Org	No
Package Developer Can Remove Component From Package	No
Component Has IP Protection	Yes

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

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 ISVforce guide.

Custom Notification Type

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 create up to 500 custom notification types. An 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 Translation

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Object

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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. Upload 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 */Sforce guide*.

Documentation

Metadata API Developer Guide: [CustomObject](#)

Custom Permission

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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 *ISVforce guide*.

Documentation

Metadata API Developer Guide: [CustomPermission](#)

Custom Setting

Can This Component Be Updated or Removed After Package Version Promotion?

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	Yes

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
- Label

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- Object Name
- Setting Type
- Visibility

More Information

Considerations When Packaging

This component can be marked as protected. For more details, see [Protected Components](#) in the ISVforce guide.

If a custom setting 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.

Custom Tab

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Data Classification on Custom Fields

Can This Component Be Updated or Removed After Package Version Promotion?

Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes
Package Developer Can Remove Component From Package	Yes. Supported for 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](#).

Data Connector Ingest API

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Connector](#)

Data Connector S3

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

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 Stream Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Source

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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: [Data Source](#)

Data Source Object

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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: [Data Source](#)

Decision Matrix Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#)

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#)

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Discovery AI Model

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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)

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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)

Can This Component Be Updated or Removed After Package Version Promotion?

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.
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](#)

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Entitlement Process

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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: EntitlementProcess

Documentation

Metadata API Developer Guide: [EntitlementProcess](#)

Salesforce Help: [Entitlement Processes](#)

Salesforce Help: [Set Up Entitlement Processes](#)

Entitlement Template

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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)

Documentation

Metadata API Developer Guide: [ESignatureConfig](#)

ESignature Envelope Config

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Experience Builder Template

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

Experience Builder Theme

Can This Component Be Updated or Removed After Package Version Promotion?

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](#).

Explainability Action Definition

Can This Component Be Updated or Removed After Package Version Promotion?

Component Is Updated During Package Upgrade	No
Subscriber Can Delete Component From Org	Yes

Package Developer Can Remove Component From Package	Yes
---	-----

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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- Label
- Active
- Description
- Explainability Action Definition

Neither Package Developer or Subscriber Can Edit

- None

More Information

Feature Name

Metadata Name: ExplainabilityActionVersion

Explainability Message Template

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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. Ensure that all the versions in the expression set are inactive before removing the components from the package.

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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

Component Is Updated During Package Upgrade	Yes. Only if the component is in inactive state.
Subscriber Can Delete Component From Org	Yes. Only if the component is in inactive state.
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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Embedded Service Deployment

Can This Component Be Updated or Removed After Package Version Promotion?

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	No

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

More Information

The Salesforce site object isn't packageable. Make sure that the destination org has a site with the same developer name as the site in the source org where the package is created.

External Actions

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- ActionName

- ApiName
- Description
- ActionSchema
- ActionSelector
- ActionParams
- Version

Both Package Developer and Subscriber Can Edit

- IsActive

Neither Package Developer or Subscriber Can Edit

- None

More Information

Feature Name

Metadata Name: MarketingAppExtAction

Component Type in 1GP Package Manager UI: Marketing App Extension

Use Case

Partners and ISVs can provide integrations with third-parties so Account Engagement customers can automate actions or tasks in external applications.

Considerations When Packaging

This component is included when the parent component MarketingAppExtension is added to a package. The related component marketingAppExtActivity isn't supported for packaging.

License Requirements

This feature is available in Plus, Advanced, or Premium editions of Account Engagement. To work with marketing app extensions and related components, users must be a Salesforce Admin or have the [required permissions to access Marketing Setup](#).

Usage Limits

The number of active extensions, activities, and actions the end user can have at one time depends on their edition of Account Engagement.

- Plus—10 active extensions, with 10 active activities and 10 active actions per active extension
- Advanced—20 active extensions, with 20 active activities and 20 active actions per active extension
- Premium—30 active extensions, with 30 active activities and 30 active actions per active extension

For more on limits, see [Considerations for Working with Marketing App Extensions](#).

Post Install Steps

To receive data, the action and its related extension must be activated for automations and the extension must have a business unit assignment.

Relationship to Other Components

This component is a child of the MarketingAppExtension component.

Documentation

This component is part of Account Engagement's extensibility feature set.

- *Salesforce Help:* [Automate Data Sharing with Third-Party Apps](#)
- *Developer Guide:* [Work with Extensibility Features](#)

External Credential

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- Label
- AuthProtocolVariant

Both Package Developer and Subscriber Can Edit

- AuthHeader
- AuthProvider
- AuthProviderUrlQueryParameter
- AuthParameter
- Description
- NamedPrincipal
- PerUserPrincipal
- Principal
- SequenceNumber

Neither Package Developer or Subscriber Can Edit

- AuthenticationProtocol
- Name

More Information

Feature Name

Metadata Name: ExternalCredential

Considerations When Packaging

After installing an external credential from a managed or unmanaged package, you must 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.

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.

Documentation

Salesforce Help: [Named Credentials](#)

External Data Connector

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Services

Can This Component Be Updated or Removed After Package Version Promotion?

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](#).

More Information

Feature Name

Metadata Name: ExternalServiceRegistration

Component Type in 1GP Package Manager UI: ExternalServiceRegistration

Considerations When Packaging

Package developers must add named credential components to the External Services registration package. A subscriber can also create a named credential in Salesforce. However, the subscriber must use the same name as the named credential specified in the External Services registration that references it.

Create named credentials manually or with Apex. Be sure to add the named credential to a package so that subscriber orgs can install it. When a subscriber org installs a named credential, it can use the Apex callouts generated by the External Services registration process.

Documentation

Salesforce Help: [External Services](#)

Field Set

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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	Yes, except a flow that is a template or overridable.

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

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.
- 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.
- 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.

Second-Generation Managed Packages	Flow Category
------------------------------------	---------------

- When you translate a flow from a managed package, the flow's Master Definition Name doesn't appear on the Translate page or the Override page. To update the translation for the Master Definition Name, edit the flow label and then update the translation from the Translate page.
- If any of the following elements are used in a flow, packageable components that they reference aren't included in the package automatically. To deploy the package successfully, manually add those referenced components to the package.
 - Post to Chatter
 - Send Email
 - Submit for Approval
- If a flow references a Lightning component that depends on a CSP Trusted Site, the trusted site isn't included in the package or change set automatically.

Usage Limits

[Salesforce Help: General Flow Limits](#)

Relationship to Other Components

The associated Flow Definition component is required for managed 1GP packages.

Documentation

[Metadata API Developer Guide: Flow](#)

[Salesforce Help: Packaging Considerations for Flows](#)

[Salesforce Help: Considerations for Deploying Flows with Packages](#)

Flow Category

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Fuel Type

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Folder

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Gateway Provider Payment Method Type

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Global Picklist

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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.

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Only Package Developer Can Edit

- Connection Type

- Developer Name
- Description
- Master Label
- Region

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.

Documentation

Salesforce Help: [Establish an Inbound Connection with AWS](#)

Letterhead

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- None

Both Package Developer and Subscriber Can Edit

- All attributes except Letterhead Name

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

- Letterhead Name

More Information

Feature Name

Metadata Name: Letterhead

Documentation

Metadata API Developer Guide: [Letterhead](#)

Lightning Application

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Lightning Bolt

Can This Component Be Updated or Removed After Package Version Promotion?

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](#).

More Information

Feature Name

Metadata Name: LightningBolt

Component Type in 1GP Package Manager UI: Lightning Bolt

Documentation

Metadata API Developer Guide: [LightningBolt](#)

Lightning Component

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Lightning Web Component

Metadata Name: LightningComponentBundle

Component Type in 1GP Package Manager UI: Lightning Web Component Bundle

Aura Component

Metadata Name: AuraDefinitionBundle

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

Documentation

[Lightning Web Components Developer Guide](#)

[Lightning Aura Components Developer Guide](#)

Lightning Event

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Lightning Interface

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Lightning Message Channel

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

More Information

Feature Name

Metadata Name: LightningMessageChannel

Component Type in 1GP Package Manager UI: Lightning Message Channel

Documentation

Metadata API Developer Guide: [Lightning Message Channel](#)

Lightning Web Components Developer Guide: [Create a Message Channel](#)

Lightning Page

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- Lightning page

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

More Information

Feature Name

Metadata Name: FlexiPage

Documentation

Metadata API Developer Guide: [Flexipage](#)

List View

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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: LiveChatSensitiveDataRule

Component Type in 1GP Package Manager UI: Sensitive Data Rules

Documentation

Metadata API Developer Guide: [LiveChatSensitiveDataRule](#)

Loyalty Program Setup

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- Loyalty Program Process records

Both Package Developer and Subscriber Can Edit

- Label
- Description
- Status

Neither Package Developer or Subscriber Can Edit

- API Name
- URL

More Information

Feature Name

Metadata Name: LoyaltyProgramSetup

Component Type in 1GP Package Manager UI: Loyalty Program Setup

Use Case

Promotion setup allows loyalty program managers to create loyalty program processes.

License Requirements

Loyalty Management permission set license

Documentation

[Salesforce Help: Create Processes with Promotion Setup](#)

Marketing App Extension

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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*

Market Segment Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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

Milestone Type

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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: MilestoneType

Documentation

Metadata API Developer Guide: [MilestoneType](#)

Salesforce Help: [Milestone Recurrence Types](#)

MktCalculatedInsightsObjectDef

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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: MktCalInsightObjectDef

Component Type in 1GP Package Manager UI: MktCalInsightObjectDef.

Use Case

Defines CDP calculated insight for easy creation on subscriber organizations.

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 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: [MktCalInsightObjectDef](#)

MktDataTranObject

Can This Component Be Updated or Removed After Package Version Promotion?

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](#)

- 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](#)

ML Prediction Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

[Only Package Developer Can Edit](#)

- ApplicationId
- Description
- Type
- PredictionField

- PushbackField
- PredictionStrategy
- NegativeExpression
- PositiveExpression

Both Package Developer and Subscriber Can Edit

- Status
- ExternalId
- Priority
- MIEternalId

Neither Package Developer or Subscriber Can Edit

- Name

More Information

Feature Name

Metadata Name: MLPredictionDefinition

Considerations When Packaging

MLPredictionDefinition is always associated with an AIApplication. The MLPredictionDefinition must never directly be included in a package. To create a package with MLPredictionDefinition, select the associated AIApplication (Type = PredictionBuilder). Packaging automatically analyzes the relationships and includes the associated MLPredictionDefinitions (and MLDataDefinitions).

Documentation

Salesforce Help: [Einstein Prediction Builder](#)

ML Recommendation Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- ApplicationId
- Description
- InteractionDateField

- NegativeExpression
- PositiveExpression

Both Package Developer and Subscriber Can Edit

- Status
- ExternalId
- MExternalId

Neither Package Developer or Subscriber Can Edit

- Name

More Information

Feature Name

Metadata Name: MLRecommendationDefinition

Considerations When Packaging

MLRecommendationDefinition is always associated with an AIApplication. The MLRecommendationDefinition must never directly be included in a package. To create a package with MLRecommendationDefinition, select the associated AIApplication (Type = RecommendationBuilder). Packaging automatically analyzes the relationships and includes the associated MLRecommendationDefinitions (and MLDataDefinitions).

Documentation

Salesforce Help: [Einstein Recommendation Builder](#)

Named Credential

Can This Component Be Updated or Removed After Package Version Promotion?

EDITIONS

Available in: All Editions

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- Endpoint (deprecated)
- Label
- NamedCredentialType

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

- AuthProvider (deprecated)
- AuthTokenEndpointUrl (deprecated)
- Authentication
- AwsAccessKey, AwsAccessSecret, AwsRegion, and AwsService (all deprecated)
- CalloutOptions
- - AllowMergeFieldsInBody
 - AllowMergeFieldsInHeader
 - GenerateAuthorizationHeader
- Certificate (deprecated)
- HttpHeader
- JwtAudience, JwtFormulaSubject, JwtIssuer, JwtSigningCertificateId, JwtTextSubject, and JwtValidityPeriodSeconds (all deprecated)
- OauthRefreshToken, OauthScope, and OAuthToken (all deprecated)
- OutboundNetworkConnectionId (deprecated)
- Password (deprecated)
- PrincipalType (deprecated)
- Protocol (deprecated)
- Url
- Username (deprecated)

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

- Name

More Information

Feature Name

Metadata Name: NamedCredential

Considerations When Packaging

Named credentials aren't automatically added to packages. If you package an external data source or Apex code that specifies a named credential as a callout endpoint, add the named credential to the package. Alternatively, make sure that the subscriber org has a valid named credential with the same name.

- For managed packages, the subscriber must add the package's namespace to a named credential's list of allowed namespaces to enable callouts. This action isn't necessary if the named credential is installed as part of the same package.
- If you have multiple orgs, you can create a named credential with the same name but with a different endpoint URL in each org. You can then package and deploy—on all the orgs—one callout definition that references the shared name of those named credentials. For example, the named credential in each org can have a different endpoint URL to accommodate differences in development and production environments. If an Apex callout specifies the shared name of those named credentials, the Apex class that defines the callout can be packaged and deployed on all those orgs without programmatically checking the environment.

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.

Legacy Named Credentials

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](#)

Recommendation Strategy

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Sustainability UOM

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)*

Object Source Target Map

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

[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

AWSExtract1000LimitAddOn-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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

[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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

-  **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

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.

Documentation

Salesforce Help: [Establish an Outbound Connection with AWS](#)

Page Layout

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Platform Event Channel

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

More Information

Feature Name

Metadata Name: PlatformEventChannel

Component Type in 1GP Package Manager UI: Platform Event Channel

Documentation

Metadata API Developer Guide: [PlatformEventChannel](#)

SEE ALSO:

[Change Data Capture Developer Guide: Compose Streams of Change Data Capture Notifications with Custom Channels](#)

Platform Event Channel Member

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

More Information

Feature Name

Metadata Name: PlatformEventChannelMember

Component Type in 1GP Package Manager UI: Platform Event Channel Member

Considerations When Packaging

- As of Winter '22, installing a managed package that contains Change Data Capture entity selections no longer causes an installation error. Before Winter '22, installing a managed package that contained Change Data Capture entity selections that were over the default allocation caused package installation errors.
- To package Change Data Capture entity selections, create a custom channel through the PlatformEventChannel metadata type. Then add entity selections to the custom channel through the PlatformEventChannelMember metadata type.

Documentation

Metadata API Developer Guide: [PlatformEventChannelMember](#)

SEE ALSO:

[Change Data Capture Developer Guide: Compose Streams of Change Data Capture Notifications with Custom Channels](#)

Platform Event Subscriber Configuration

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Editable Properties After Package Promotion or Installation

Only Package Developer Can Edit

- batchSize
- platformEventConsumer
- user

Both Package Developer and Subscriber Can Edit

- None

Neither Package Developer or Subscriber Can Edit

- None

More Information

Feature Name

Metadata Name: PlatformEventSubscriberConfig

Component Type in 1GP Package Manager UI: Platform Event Subscriber Configuration

Use Case

Override the default running user and batch size of a platform event Apex trigger.

Relationship to Other Components

PlatformEventSubscriberConfig is tied to an Apex trigger.

Documentation

Platform Events Developer Guide: [Configure the User and Batch Size for Your Platform Event Trigger](#)

Process

See [Flow](#)

Prompts (In-App Guidance)

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

More Information

Feature Name

Metadata Name: Prompt

Component Type in 1GP Package Manager UI: Prompt

Documentation

Metadata API Developer Guide: [Prompt](#)

Quick Action

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- Field Overrides

Both Package Developer and Subscriber Can Edit

- All attributes except Field Overrides

Neither Package Developer or Subscriber Can Edit

- None

More Information

Feature Name

Metadata Name: QuickAction

Component Type in 1GP Package Manager UI: Quick Action

Documentation

Salesforce Help: Quick Actions

Record Action Deployment

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 Type

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 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 Redirects to External URLs](#)

Metadata API Developer Guide: [SecuritySettings](#)

Referenced Dashboard

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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

Remote Site Setting

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Developer Guide: [RemoteSiteSettings](#)

Report

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 upload 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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](#)

Reporting Snapshot

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Reporting Snapshot Unique Name

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

- Reporting Snapshot Unique Name

More Information

Developers of managed packages must consider the implications of introducing reporting snapshots 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 reporting snapshot referencing the report isn't installed, even though the Package Installation page indicates that it will be. Also, if the subscriber has modified the report, the report can return results impacting the information displayed by the reporting snapshot. As a best practice, the developer releases the reporting snapshot and the related reports in the same version.

Because the subscriber selects the running user, some reporting snapshot field mappings could become invalid if the running user doesn't have access to source or target fields.

Salesforce IoT

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

Slack App

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#)

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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
-----------------------------	----

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

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 Item Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 SvcCatalogFulfillmentFlowItem records.

Documentation

Salesforce Help: [Catalog Item Fulfillment Flows](#)

Stationary Asset Environmental Source Record Type Configuration

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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: Map Stationary Asset Environmental Source Record Types](#)
- [Salesforce Help: Create a Stationary Asset Environmental Source Record](#)

Static Resource

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

- ApplIdentifier
- DataConnectorType
- StreamingAppDataConnectorType

More Information

Feature Name

Metadata Name: StreamingAppDataConnector

Use Case

StreamingAppDataConnector is spidered in through data streams (DataStreamDefinition). You need this component if you want to package a web or mobile data stream.

License Requirements

You need Customer 360 Audiences Corporate (cdpPsl) licenses on both the managed 1GP packaging org, and the subscriber org.

Post Install Steps

A user must create a data stream via ui-api or using the Customer Data Platform.

Relationship to Other Components

StreamingAppDataConnector component is tightly related to the ExternalDataConnector component and requires a DataStreamDefinition to pull both in.

Documentation

Customer Data Platform Developer Guide: [Capture Web Interactions](#)

Customer Data Platform Developer Guide: [Integrate your Mobile Applications](#)

Timeline Object Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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](#).

More Information

Feature Name

Metadata Name: TimesheetTemplate

Translation

Add translations to your managed packages.

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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.

Override Translations in Second-Generation Managed Packages and Unlocked Packages

You can override metadata translations for custom objects in namespaced unlocked packages and second-generation managed packages. For example, override the label on a custom field or workflow task.

Override Translations in Second-Generation Managed Packages and Unlocked Packages

You can override metadata translations for custom objects in namespaced unlocked packages and second-generation managed packages. For example, override the label on a custom field or workflow task.



Note: Overriding translations in second-generation managed packages and unlocked packages has limitations:

- You can't override translations for standard objects in packages.
- You can't override translations for global picklist value sets.
- You can't override data translations.

If you installed a managed package that includes translations, those translated values appear to users regardless of whether the language is active on the Translation Language Settings Setup page. Before you can override those translations, you must select languages for your org and enable Translation Workbench.

1. From Setup, in the Quick Find box, enter *Override*, and then select **Override**.

2. Select the **Package** that you're overriding.

3. Select the **Language** that you're entering your overrides in.



Note: The Language list shows the languages that meet these criteria:

- The language is in the package that's associated with this namespace.
- There is at least one translation for the language, or it's the package default language.

4. Select a **Setup Component**. See [Metadata Available for Translation](#) for a list of translatable components.

5. Depending on the setup component, select the next options.

The aspect is a part of the setup component that you can translate. For example:

- Workflow tasks have an object (for example, Account or Contact) and aspect (Subject or Comment).
- Custom Report Types have a custom report type entity (Custom Report Type, Custom Report Type Column, or Custom Report Type Layout Section) and aspect (field label or description).

Editions

Available in: Salesforce Classic ([not available in all orgs](#)) and Lightning Experience

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

User Permissions

To override metadata translations:

- View Setup and Configuration
- AND
- Customize Application

- Flows have a flow type (Flow and Autolaunched Flow), a flow name, and a flow component (Definition, Version, Screen Info, Screen Field, and Choice). Flow components can have a flow version, screen, or aspect.

For global value sets and picklist values, you can translate inactive values by selecting **Show Inactive Values**.

- To enter new values, double-click in the translation column. You can press TAB to advance to the next editable field or SHIFT-TAB to go to the previous editable field.

 **Note:** The Out of Date column indicates the possibility that the term needs translation because the primary label has been updated. When editing a button or link label, you see the Button or Link Name column, which is used to refer to the component when using SOAP API.

- Click **Save**.

UI Object Relation Config

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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](#)

Validation Rule

Can This Component Be Updated or Removed After Package Version Promotion?

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

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](#).

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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: Map Vehicle Asset Emissions Source Record Type Configurations](#)
- [Salesforce Help: Create a Vehicle Asset Emissions Source Record](#)

View Definition

Can This Component Be Updated or Removed After Package Version Promotion?

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

- 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 Application

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 source 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 source 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 source 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 source 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 source 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 source 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 source 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 should be installed in source 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 Xmd

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

- 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 source 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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

More Information

Feature Name

Metadata Name: WebStoreTemplate

Documentation

Metadata API Developer Guide: [WebStoreTemplate](#)

Workflow Email Alert

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 ISVforce guide.

Workflow Field Update

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 ISVforce guide.

Workflow Outbound Message

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 ISVforce guide.

Workflow Rule

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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

Can This Component Be Updated or Removed After Package Version Promotion?

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

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

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 ISVforce guide.

Develop Second-Generation Managed Packages

Ready to get started? Create your first second-generation managed package, and then update and create new versions of your package.

[Create and Update a Second-Generation Managed Package](#)

A package is a top-level container that holds important details about the app or package: the package name, description, and associated namespace. When you're ready to test or share your package, use the `force:package:create` Salesforce CLI command to create a package.

[View Package Details for a Second-Generation Managed Package](#)

View the details of previously created second-generation managed packages from the command line.

[Create and Update Versions of a Second-Generation Managed Package](#)

A package version is a fixed snapshot of the package contents and related metadata. The package version is an installable, immutable artifact that lets you manage changes and track what's different each time you release or deploy a specific set of changes.

[View Details about a Second-Generation Managed Package Version](#)

Retrieve details about second-generation managed package versions that are in progress, or have already been created.

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

The project configuration file is a blueprint for your project. The settings in the file create an outline of your managed 2GP package and determine the package attributes and package contents.

[Get Ready to Promote and Release a Second-Generation Managed Package Version](#)

By now it's likely that you've already created many different versions of your managed 2GP package and tested them. When you have a package version that you're ready to distribute, promoting the package version is the next step.

[Specify a Package Ancestor in the Project File for a Second-Generation Managed Package](#)

When you create a second-generation managed package version you specify a package ancestor in your `sfdx-project.json` file. We require that the package ancestor you specify is the highest promoted package version number for that package. You can either update the ancestor version number each time you create a package version, or you can use a keyword.

Create and Update a Second-Generation Managed Package

A package is a top-level container that holds important details about the app or package: the package name, description, and associated namespace. When you're ready to test or share your package, use the `force:package:create` Salesforce CLI command to create a package.

To create a package, change to the project directory in the CLI. The package name you enter becomes the package alias, and is automatically added to the project file. You can choose to designate an active Dev Hub org user to receive email notifications for Apex gacks, and install, upgrade, or uninstall failures associated with your packages. For definitions of each parameter shown here, see `force:package:create` in the Salesforce CLI Reference Guide.

```
sfdx force:package:create --name "Expenser App" --packagetype Managed \
--path "expenser-main" --targetdevhubusername my-hub --errornotificationusername \
me@devhub.org
```

The package details you supply when you create a package are automatically added to your `sfdx-project.json` [project configuration file](#).

Update the Package

To update the name or description of an existing package, use this command.

```
sfdx force:package:update --package "Expense App" --name "Expense Manager App" \
--description "The Winter '20 release is packed with an exciting set of features." \
--errornotificationusername me2@devhub.org
```



Note: You can't change the package namespace or package type after you create the package.

View Package Details for a Second-Generation Managed Package

View the details of previously created second-generation managed packages from the command line.

To display a list of all packages in the Dev Hub org, use this command.

```
sfdx force:package:list --targetdevhubusername my-hub
```

You can view the namespace, package name, ID, and other details in the output.

Namespace	Prefix	Name	Id	Alias	Description	Type
db_exp_manager		Expenser App	0HoB00000004CzRKAU	Expenser App		Managed
db_exp_manager		Expenser Logic	0HoB00000004CzMKAU	Expenser Logic		Managed
db_exp_manager		Expenser Schema	0HoB00000004CzHKAU	Expenser Schema		Managed

Include optional parameters to filter the list results based on the modification date, creation date, and to order by specific fields or package IDs. To limit the details, use `--concise`.

To show expanded details, use `--verbose`. The verbose parameter displays these additional details.

- Created By
- Error Notification Username
- Subscriber Package ID

Create and Update Versions of a Second-Generation Managed Package

A package version is a fixed snapshot of the package contents and related metadata. The package version is an installable, immutable artifact that lets you manage changes and track what's different each time you release or deploy a specific set of changes.

Before you create a package version, first verify package details, such as the package name, dependencies, and major, minor, and patch version numbers, in the `sfdx-project.json` file. Verify that the metadata you want to change or add in the new package version is in the package's main directory.

How Many Managed 2GP Package Versions Can I Create Per Day?

Run this command to see how many package versions you can create per day and how many you have remaining.

```
sfdx force:limits:api:display
```

Look for the `Package2VersionCreates` entry.

NAME	REMAINING	MAXIMUM
Package2VersionCreates	23	50

Create a Managed 2GP Package Version

Create the package version with this command. Specify the package alias or ID (0Ho). You can also include a scratch definition file that contains a list of features and setting that the metadata of the package version depends on.

```
sfdx force:package:version:create --package "Expenser App" --installationkey "HIF83ks8ks7C"
 \
--definitionfile config/project-scratch-def.json --wait 10
```

 **Note:** When creating a package version, specify a `--wait` time to run the command in non-asynchronous mode. If the package version is created within that time, the `sfdx-project.json` file is automatically updated with the package version information. If not, you must manually edit the project file.

Update a Managed 2GP Package Version

You can update most properties of a package version from the command line. For example, you can change the package version name or description. One important exception is that you can't change the release status.

In this example, we're adding the `tag` parameter and specifying the git commit ID associated with this package version.

```
sfdx force:package:version:update --package "Expenser App@1.3.0-5" --tag "git commit id
08dcfsdf"
```

After the update is complete, you'll see output that looks like

```
Successfully updated the package version. 04tB0000000KPhnIAG
```

View Details about a Second-Generation Managed Package Version

Retrieve details about second-generation managed package versions that are in progress, or have already been created.

View Status and Progress Details for a Managed 2GP Package Version

Depending on the package size and other variables, creating a package version can be a long-running process. You can easily view the status and monitor progress using this report command.

```
sfdx force:package:version:create:report --packagecreateresponseid 08cxx00000000YDAAY
```

The output shows details about the request.

==== Package Version Create Request	
NAME	VALUE
Version Create Request Id	08cB00000004CBxIAM
Status	InProgress
Package Id	0HoB00000004C9hKAE

Package Version Id	05iB0000000CaaNIAS
Subscriber Package Version Id	04tB0000000NOimIAG
Tag	git commit id 08dcfsdf
Branch	
CreatedDate	2018-05-08 09:48
Installation URL	https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB0000000NOimIAG

You can find the request ID (08c) in the initial output of `force:package:version:create`.

If you have more than one pending request to create package versions, you can view a list of all requests with this command.

```
sfdx force:package:version:create:list --createdlastdays 0
```

Details for each request display as shown here (IDs and labels truncated).

== Package Version Create Requests [3]								
ID	STATUS	PACKAGE2 ID	PKG2 VERSION ID	SUB PKG2 VER ID	TAG	BRANCH	CREATED DATE	
08c...	Error	0Ho...						
08c...	Success	0Ho...	05i...	04t...			2022-06-22 12:07	
08c...	Success	0Ho...	05i...	04t...			2022-06-23 14:55	

Retrieve List of all Package Versions Associated with a Dev Hub Org

To display a list of all package versions in the Dev Hub org, use this command.

```
sfdx force:package:version:list --targetdevhubusername my-hub
```

You can view the namespace, version name, and other details in the output.

Package Name Installation Key	Namespace Released	Version	Sub Pkg Ver Id	Alias
Expenser Schema false	db_exp_manager true	0.1.0.1	04tB0000000719qIAA	Expenser Schema@0.1.0-1
Expenser Schema false	db_exp_manager true	0.2.0.1	04tB000000071AjIAI	Expenser Schema@0.2.0-1
Expenser Schema false	db_exp_manager false	0.3.0.1	04tB000000071AtIAI	Expenser Schema@0.3.0-1
Expenser Schema false	db_exp_manager true	0.3.0.2	04tB000000071AyIAI	Expenser Schema@0.3.0-2
Expenser Schema false	db_exp_manager false	0.3.1.1	04tB0000000KGU6IAO	Expenser Schema@0.3.1-1
Expenser Schema false	db_exp_manager true	0.3.1.2	04tB0000000KGUBIA4	Expenser Schema@0.3.1-2
Expenser Schema false	db_exp_manager true	0.3.2.1	04tB0000000KGUQIA4	Expenser Schema@0.3.2-1
Expenser Logic false	db_exp_manager true	0.1.0.1	04tB0000000719vIAA	Expenser Logic@0.1.0-1
Expenser App false	db_exp_manager true	0.1.0.1	04tB000000071AOIAI	Expenser App@0.1.0-1

To view details about a specific package, include `--package` parameter when you run `sfdx force:package:version:list`.

To show expanded details, use `--verbose`. The verbose parameter displays these additional details.

- Ancestor
- Ancestor Version
- Branch
- Build Duration in Seconds
- Code Coverage
- Code Coverage Met
- Created By
- Created Date
- Description
- Installation URL
- Language
- Managed Metadata Removed
- Package ID
- Package Version ID
- Release Version
- Tag
- Validation Skipped
- WasTransferred

Project Configuration File for a Second-Generation Managed Package

The project configuration file is a blueprint for your project. The settings in the file create an outline of your managed 2GP package and determine the package attributes and package contents.

Here are some of the parameters you can specify in the project configuration file. For additional parameters, see [Advanced Project Configuration Parameters for Second-Generation Managed Packages](#).

Name	Details
ancestorId	<p>Required? It depends on whether you've already promoted a package version of this package. If yes, you must specify either the ancestorId or ancestorVersion. If no, this parameter isn't required.</p> <p>Default if Not Specified: None</p> <p>None. The ID of the immediate parent in the package ancestry tree of the package version you're creating. The <code>ancestorId</code> requires the 04t of the package version, or an alias to the package version. When specifying ancestors, you can use either <code>ancestorId</code> or <code>ancestorVersion</code>.</p> <p>Example:</p> <pre>"ancestorId": "Expenser Logic@0.1.0-1"</pre> <p>For more information, see Specify a Package Ancestor in the Project File for a Second-Generation Managed Package.</p>

Name	Details
ancestorVersion	<p>Required? It depends on whether you've already promoted a package version of this package. If yes, you must specify either the ancestorId or ancestorVersion. If no, this parameter isn't required.</p> <p>Default if Not Specified: None</p> <p>The version number of the immediate parent in the package ancestry tree of the package version you are creating.</p> <p>Specify the ancestor version using the format of major.minor.patch.build. When specifying ancestors, you can use either ancestorId or ancestorVersion.</p> <p>Example:</p> <pre>"ancestorVersion": "0.1.0.1"</pre> <p>For more information, see Specify a Package Ancestor in the Project File for a Second-Generation Managed Package.</p>
default	<p>Required? Yes, if you've specified more than one package directory</p> <p>Default if Not Specified: true</p> <p>Indicates the default package directory. When metadata is retrieved from a development org (scratch org or source-tracked sandbox) using <code>force:source:pull</code>, it's placed in the default package directory.</p> <p>There can be only one package directory in which the default is set to true.</p>
definitionFile	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>A reference to an external <code>.json</code> file used to specify the features and org settings required for the metadata of your package, such as the scratch org definition.</p> <p>Example:</p> <pre>"definitionFile": "config/project-scratch-def.json",</pre>
namespace	<p>Required? Yes</p> <p>Default if Not Specified: None</p> <p>A 1–15 character alphanumeric identifier that distinguishes your package and its contents from packages of other developers.</p>
package	<p>Required? Yes</p> <p>Default if Not Specified: None</p> <p>The package name specified in the project json file.</p>

Name	Details
packageAliases	<p>Required? No</p> <p>Default if Not Specified: Salesforce CLI updates this file with the aliases when you create a package or package version. You can also manually update this section for existing packages or package versions. You can use the alias instead of the cryptic package ID when running CLI <code>force:package</code> commands.</p>
path	<p>Required? Yes</p> <p>Default if Not Specified: None.</p> <p>Specify the location that contains the package metadata in the <code>--path</code> attribute of <code>force:package:create</code> Salesforce CLI command.</p>
seedMetadata	<p>Required? No</p> <p>Default if Not Specified: None.</p> <p>Specify the path to your seedMetadata directory.</p> <p>Seed metadata is available to standard value sets only. If your package depends on standard value sets, you can specify a seed metadata directory that contains the value sets.</p> <p>Example:</p> <pre>"packageDirectories": [{ "seedMetadata": { "path": "my-unpackaged-seed-directory" } },]</pre>
versionDescription	<p>Required? No</p> <p>Default if Not Specified: None</p>
versionName	<p>Required? No</p> <p>Default if Not Specified: If not specified, the CLI uses <code>versionNumber</code> as the version name.</p>
versionNumber	<p>Required? Yes</p> <p>Default if Not Specified: None</p> <p>Version numbers are formatted as major.minor.patch.build. For example, 1.2.1.8.</p> <p>To automatically increment the build number to the next available build for the package, use the keyword NEXT (1.2.1.NEXT).</p>

When you specify a parameter using Salesforce CLI, it overrides the value listed in the project definition file.

The Salesforce DX project definition file is a JSON file located in the root directory of your project. Use the `force:project:create` CLI command to generate a project file that you can build upon. Here's how the parameters in `packageDirectories` appear.

```
{
  "namespace": "exp-mgr",
  "sfdcLoginUrl": "https://login.salesforce.com",
  "sourceApiVersion": "47.0",
  "packageDirectories": [
    {
      "path": "util",
      "default": true,
      "package": "Expense Manager - Util",
      "versionName": "Winter '20",
      "versionDescription": "Welcome to Winter 2020 Release of Expense Manager Util
Package",
      "versionNumber": "4.7.0.NEXT",
      "definitionFile": "config/scratch-org-def.json"
    },
    {
      "path": "exp-core",
      "default": false,
      "package": "Expense Manager",
      "versionName": "v 3.2",
      "versionDescription": "Winter 2020 Release",
      "versionNumber": "3.2.0.NEXT",
      "ancestorVersion": "3.0.0.7",
      "definitionFile": "config/scratch-org-def.json",
      "dependencies": [
        {
          "package": "Expense Manager - Util",
          "versionNumber": "4.7.0.LATEST"
        },
        {
          "package": "External Apex Library - 1.0.0.4"
        }
      ]
    }
  ],
  "packageAliases": {
    "Expense Manager - Util": "0HoB00000004CFpKAM",
    "External Apex Library@1.0.0.4": "04tB0000000IB1EIAW",
    "Expense Manager": "0HoB00000004CFuKAM"
  }
}
```

What If I Don't Want My Salesforce DX Project Automatically Updated?

In some circumstances, you don't want to have automatic updates to the `sfdx-project.json` file. When you require more control, use these environment variables to suppress automatic updates to the project file.

For This Command	Set This Environment Variable to True
force:package:create	SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_CREATE
force:package:version:create	SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_VERSION_CREATE

Get Ready to Promote and Release a Second-Generation Managed Package Version

By now it's likely that you've already created many different versions of your managed 2GP package and tested them. When you have a package version that you're ready to distribute, promoting the package version is the next step.

Each package version you create is a beta version, unless you promote it to the managed-released state. Beta versions can be installed in only scratch orgs and sandboxes. After you install a beta version into an org, you can't later upgrade that installed beta version. Keep this in mind when you select which org to install and test your beta package version. If you use this sandbox as part of your release pipeline, then using a disposable scratch org is a better option to test your beta package.

A beta package version must pass a 75% code coverage requirement before it can be promoted. To learn more, see [Code Coverage for Second-Generation Managed Packages](#).

To promote a package version to the released state, run the `force:package:version:promote` Salesforce CLI command. For step-by-step instructions on promoting a package version, see [Release a Second Generation Managed Package](#).

After a package version is promoted, you can install it in either a production org or development orgs, and can be distributed to your customers.

For every minor package version, you can promote only one beta version. For example, if you create several beta versions of package version 2.3, only one of those versions can be promoted. After promoting package version 2.3, start your new development using version number 2.4.

After a package version is promoted to the released state, you can't reverse the promotion. If you discover you don't want to distribute a version you promoted, you can't reverse that version back to the beta state. To ensure that that version isn't inadvertently shared and installed in a customer org, we recommend you use the `force:package:version:update` Salesforce CLI command and set the installation key to something cryptic and difficult to guess.

Specify a Package Ancestor in the Project File for a Second-Generation Managed Package

When you create a second-generation managed package version you specify a package ancestor in your `sfdx-project.json` file. We require that the package ancestor you specify is the highest promoted package version number for that package. You can either update the ancestor version number each time you create a package version, or you can use a keyword.

Here are three different ways to set the package ancestor.

Use the HIGHEST Keyword (Recommended)

Use the keyword HIGHEST with either the `ancestorId` or `ancestorVersion` attribute in the `sfdx-project.json` file. This keyword automatically sets the ancestor to the highest promoted package version number.

```
"packageDirectories": [
{
  "path": "util",
```

```
"package": "Expense Manager - Util",
"versionNumber": "4.7.0.NEXT",
"ancestorVersion": "HIGHEST"
},
```

This keyword makes it easy to set your package ancestor to use linear versioning, until you have a reason to break from linear versioning.

Use the Ancestor Version Attribute

Set the `ancestorVersion` attribute in the `sfdx-project.json` file to the package version's major.minor.patch number. This approach requires you to update the ancestor version number every time the major, minor, or patch value changes.

```
"packageDirectories": [
{
"path": "util",
"package": "Expense Manager - Util",
"versionNumber": "4.7.0.NEXT",
"ancestorVersion": "4.6.0"
},
```

Use the Ancestor ID Attribute

Set the `ancestorId` attribute in the `sfdx-project.json` file to either the 04t ID or the package version's alias. This approach requires you to update the ancestor version number every time you create a package version.

```
"packageDirectories": [
{
"path": "util",
"package": "Expense Manager - Util",
"versionNumber": "4.7.0.NEXT",
"ancestorId": "04tB0000000cWwnIAE"
},
```

```
"packageDirectories": [
{
"path": "util",
"package": "Expense Manager - Util",
"versionNumber": "4.7.0.NEXT",
"ancestorId": "expense-manager@4.6.0.1"
},
```



Note: Only package versions that have been promoted to managed-released state, can be listed as an ancestor.

Override Linear Package Ancestry Behavior

To break from linear package versioning, specify a package ancestor that isn't the highest promoted package version and use the Salesforce CLI parameter `--skipancestorcheck` when you create a package version.

```
sfdx force:package:version:create --package "Expenser App" --skipancestorcheck
```

The CLI parameter indicates that you're intentionally choosing to specify a package version that isn't the highest promoted package version.

You can choose to not specify a package ancestor by using the keyword, `NONE`, with either the `ancestorId` or `ancestorVersion` attribute in the `sfdx-project.json` file.

```
"packageDirectories": [
{
  "path": "util",
  "package": "Expense Manager - Util",
  "versionNumber": "4.7.0.NEXT",
  "ancestorVersion": "NONE"
},
```

Because package ancestors determine package upgrade paths, existing customers can't upgrade to a package version that is created without a specified ancestor. Use `NONE` if you don't plan to promote the package version you're creating.

If you've already promoted a previous package version, and you set the ancestor to `NONE` on a new package version associated with the same package, include `--skipancestorcheck` when you create that package version. When you create your first package version, you can also set the ancestor to `NONE` and specify `--skipancestorcheck`.

What to Remember about Package Ancestry

- Package ancestry determines whether existing packages can be upgraded to newer package versions. If you're breaking from linear versioning, or plan to abandon a package version that is installed in customer orgs, consider how your existing customers will be impacted, and whether an upgrade path is available to them.
- If you abandon a package version, delete the version using the Salesforce CLI command `force:package:version:delete`. If you aren't able to delete the package version, then update the package version's installation key so the abandoned package version can't be inadvertently installed. Use `force:package:version:update` to update the installation key.

Install and Uninstall Second-Generation Managed Packages

Use a disposable scratch org to test your second-generation managed packages (managed 2GP). You can install or uninstall a managed 2GP package using a Salesforce CLI command, or from the Setup page. Because you can't upgrade a beta package version, be sure you don't install it in a sandbox that you use in your release pipeline, such as UAT or staging.

[Use the CLI to Install a Second-Generation Managed Package](#)

If you're working with the Salesforce CLI, you can use the `force:package:install` command to install packages in a scratch org or target subscriber org.

[Use a URL to Install a Second-Generation Managed Package](#)

Install a second-generation managed package from a browser.

[Upgrade a Second-Generation Managed Package Version](#)

Are you introducing metadata changes to an existing second-generation managed package? You can use the CLI to upgrade one package version to another.

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

You can uninstall a second-generation managed package from an org using Salesforce CLI or from the Setup UI. When you uninstall second-generation managed packages, all components in the package are deleted from the org.

Use the CLI to Install a Second-Generation Managed Package

If you're working with the Salesforce CLI, you can use the `force:package:install` command to install packages in a scratch org or target subscriber org.

Before you install a second-generation managed package (managed 2GP) in a scratch org, run this command to list all the packages and locate the ID or package alias.

```
sfdx force:package:version:list
```

Identify the version you want to install. Enter this command, supplying the package alias or package ID (starts with 04t).

```
sfdx force:package:install --package "Expense Manager@1.2.0-12" --targetusername jdoe@example.com
```

If you've already set the scratch org with a default username, enter just the package version ID.

```
sfdx force:package:install --package "Expense Manager@1.2.0-12"
```

 **Note:** If you've defined an alias (with the `-a` parameter), you can specify the alias instead of the username for `--targetusername`.

The CLI displays status messages regarding the installation.

```
Waiting for the subscriber package version install request to get processed. Status = InProgress Successfully installed the subscriber package version: 04txx0000000FIuAAM.
```

Control Managed 2GP Package Installation Timeouts

When you issue a `force:package:install` command, it takes a few minutes for a package version to become available in the target org and for installation to complete. To allow sufficient time for a successful install, use these parameters that represent mutually exclusive timers.

- `--publishwait` defines the maximum number of minutes that the command waits for the package version to be available in the target org. The default is 0. If the package is not available in the target org in this time frame, the install is terminated.
Setting `--publishwait` is useful when you create a new package version and then immediately try to install it to target orgs.

 **Note:** If `--publishwait` is set to 0, the package installation immediately fails, unless the package version is already available in the target org.

- `--wait` defines the maximum number of minutes that the command waits for the installation to complete after the package is available. The default is 0. When the `--wait` interval ends, the install command completes, but the installation continues until it either fails or succeeds. You can poll the status of the installation using `sfdx force:package:install:report`.

 **Note:** The `--wait` timer takes effect after the time specified by `--publishwait` has elapsed. If the `--publishwait` interval times out before the package is available in the target org, the `--wait` interval never starts.

For example, consider a package called Expense Manager that takes five minutes to become available on the target org, and 11 minutes to install. The following command has `publishwait` set to three minutes and `wait` set to 10 minutes. Because Expense Manager requires more time than the set `publishwait` interval, the installation is aborted at the end of the three minute `publishwait` interval.

```
sfdx force:package:install --package "Expense Manager@1.2.0-12" --publishwait 3 --wait 10
```

The following command has `publishwait` set to six minutes and `wait` set to 10 minutes. If not already available, Expense Manager takes five minutes to become available on the target org. The clock then starts ticking for the 10 minute `wait` time. At the end of 10

minutes, the command completes because the `wait` time interval has elapsed, although the installation is not yet complete. At this point, `package:install:report` indicates that the installation is in progress. After one more minute, the installation completes and `package:install:report` indicates a successful installation.

```
sfdx force:package:install --package "Expense Manager@1.2.0-12" --publishwait 6 --wait 10
```

Use a URL to Install a Second-Generation Managed Package

Install a second-generation managed package from a browser.

If you create packages from the CLI, you can derive an installation URL for the package by adding the subscriber package ID to your Dev Hub URL. You can use this URL to test different deployment or installation scenarios.

For example, if the package version has the subscriber package ID, 04tB00000009oZ3JBl, add the ID as the value of `apvId`.

`https://MyDomainName.lightning.force.com/packagingSetupUI/ipLanding.app?apvId=04tB00000009oZ3JBl`

Anyone with the URL and a valid login to a Salesforce org can install the package.

To install the package:

1. In a browser, enter the installation URL.
2. Enter your username and password for the Salesforce org in which you want to install the package, and then click **Login**.
3. If the package is protected by an installation key, enter the installation key.
4. For a default installation, click **Install**.

A message describes the progress. You receive a confirmation message when the installation is complete.

Upgrade a Second-Generation Managed Package Version

Are you introducing metadata changes to an existing second-generation managed package? You can use the CLI to upgrade one package version to another.

When you perform a package upgrade, here's what to expect for metadata changes.

- Metadata introduced in the new version is installed as part of the upgrade.
- Metadata modified in the new version is updated as part of the upgrade.
- Metadata removed in the new version is either deprecated or deleted as part of the upgrade.

To upgrade a package version, the new version must be a direct descendent of the package version installed in your org. See [Specify a Package Ancestor in the Project File for a Second-Generation Managed Package](#) for more information.

Uninstall a Second-Generation Managed Package

You can uninstall a second-generation managed package from an org using Salesforce CLI or from the Setup UI. When you uninstall second-generation managed packages, all components in the package are deleted from the org.

To use the CLI to uninstall a package from the target org, authorize the Dev Hub org and run this command.

```
sfdx force:package:uninstall --package "Expense Manager@2.3.0-5"
```

You can also uninstall a package from the web browser. Open the Salesforce org where you installed the package.

```
sfdx force:org:open -u me@my.org
```

Then uninstall the package.

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

Considerations on Uninstalling Packages

- If you're uninstalling a package that includes a custom object, all components on that custom object are also deleted. Deleted items include custom fields, validation rules, custom buttons, and links, 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.

SEE ALSO:

[Salesforce CLI Command Reference](#)

Prepare to Distribute Your Second-Generation Managed Package

Before you release a version of your second-generation managed package, ensure that you understand the code coverage requirements, release logistics, and how to publish your app on AppExchange.

[Code Coverage for Second-Generation Managed Packages](#)

Before you can release and distribute a second-generation managed package version on AppExchange, the Apex code must meet a minimum 75% code coverage requirement. And every Apex Trigger in a package needs test coverage.

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

To ensure the security of the metadata in your second-generation managed package, you must specify an installation key when creating a package version. Package creators provide the key to authorized subscribers so they can install the package. Package installers provide the key during installation, whether installing the package from the CLI or from a browser. An installation key is the first step during installation. The key ensures that no package information, such as the name or components, is disclosed until the correct installation key is supplied.

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

Each new second-generation managed package version is marked as beta when created. As you develop your package, you may create several package versions before you create a version that is ready to be released and distributed. Only released package versions can be listed on AppExchange and installed in customer orgs.

[Share Release Notes and Post-Install Instructions for Second-Generation Managed Packages](#)

Share details with your subscribers about what's new and changed in a released second-generation managed package.

[Publishing Your App on AppExchange](#)

If you've published a first-generation managed package, you'll notice the process for publishing a second-generation managed package (managed 2GP) is different. After you link your Dev Hub org to the AppExchange publishing console, all your released managed 2GP package versions are visible in the publishing console.

Code Coverage for Second-Generation Managed Packages

Before you can release and distribute a second-generation managed package version on AppExchange, the Apex code must meet a minimum 75% code coverage requirement. And every Apex Trigger in a package needs test coverage.

To compute code coverage using Salesforce CLI, use the `--codecoverage` parameter when you run the `force:package:version:create` command.

Package version creation often takes longer to complete when code coverage is being computed, so consider when to include the code coverage parameter. You can create beta package versions without computing code coverage, but these beta versions can't be promoted.

If you try to promote a beta package version to managed-released and the version was created without specifying code coverage, or the code coverage in the package version is less than 75%, the package promotion fails. Code coverage is calculated during package version validation. If you skip validation using the `--skipvalidation` parameter, code coverage isn't calculated for that package version.

View code coverage information for a package version using `force:package:version:list` with the `--verbose` parameter, or `force:package:version:report` command in Salesforce CLI.

Package Installation Key for Second-Generation Managed Packages

To ensure the security of the metadata in your second-generation managed package, you must specify an installation key when creating a package version. Package creators provide the key to authorized subscribers so they can install the package. Package installers provide the key during installation, whether installing the package from the CLI or from a browser. An installation key is the first step during installation. The key ensures that no package information, such as the name or components, is disclosed until the correct installation key is supplied.

To set the installation key, add the `--installationkey` parameter to the command when you create the package version. This command creates a package and protects it with the installation key.

```
sfdx force:package:version:create --package "Expense Manager" --installationkey  
"JSB7s8vXU93fI"
```

Supply the installation key when you install the package version in the target org.

```
sfdx force:package:install --package "Expense Manager" --installationkey "JSB7s8vXU93fI"
```

Change the Installation Key for an Existing Package Version

You can change the installation key for an existing package version with the `force:package:version:update` command.

```
sfdx force:package:version:update --package "Expense Manager@1.2.0-4" --installationkey "HIF83kS8kS7C"
```

Create a Package Version Without an Installation Key

If you don't require security measures to protect your package metadata, you can create a package version without an installation key.

```
sfdx force:package:version:create --package "Expense Manager" --installationkeybypass
```

Check Whether a Package Version Requires an Installation Key

To determine whether a package version requires an installation key, use either the `force:package:version:list` or `force:package:version:report` CLI command.

Release a Second-Generation Managed Package

Each new second-generation managed package version is marked as beta when created. As you develop your package, you may create several package versions before you create a version that is ready to be released and distributed. Only released package versions can be listed on AppExchange and installed in customer orgs.

Before you promote the package version, ensure that the user permission, **Promote a package version to released**, is enabled in the Dev Hub org associated with the package. Consider creating a permission set with this user permission, and then assign the permission set to the appropriate user profiles.

When you're ready to release, use `force:package:version:promote`.

```
sfdx force:package:version:promote --package "Expense Manager@1.3.0-7"
```

If the command is successful, a confirmation message appears.

```
Successfully promoted the package version, ID: 04tb0000000719qIAA to released.
```

After the update succeeds, view the package details.

```
sfdx force:package:version:report --package "Expense Manager@1.3.0-7"
```

Confirm that the value of the Released property is `true`.

==== Package Version	
NAME	VALUE
Name	ver 1.0
Alias	Expense Manager-1.0.0.5
Package Version Id	05ib0000000CaahIAC
Package Id	0HoB0000000CabmKAC
Subscriber Package Version Id	04tb0000000NPbBIW

Version	1.0.0.5
Description	update version
Branch	
Tag	git commit id 08dcfsdf
Released	true
Created Date	2018-05-08 09:48
Installation URL	https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB0000000NPbBIAW

You can promote and release only one time for each package version number, and you can't undo this change.

Share Release Notes and Post-Install Instructions for Second-Generation Managed Packages

Share details with your subscribers about what's new and changed in a released second-generation managed package.

You can specify a release notes URL to display on the package detail page in the subscriber's org. And you can share instructions about using your package by specifying a post install URL. The release notes and post install URLs display on the Installed Packages page in Setup, after a successful package installation. For subscribers who install packages using an installation URL, the package installer page displays a link to release notes. And subscribers are redirected to your post install URL following a successful package installation or upgrade.

Specify the `postInstallUrl` and `releaseNotesUrl` attributes in the `packageDirectories` section for the package.

```
"packageDirectories": [
  {
    "path": "expenser-schema",
    "default": true,
    "package": "Expense Schema",
    "versionName": "ver 0.3.2",
    "versionNumber": "0.3.2.NEXT",
    "postInstallScript": "PostInstallScript",
    "uninstallScript": "UninstallScript",
    "postInstallUrl": "https://expenser.com/post-install-instructions.html",
    "releaseNotesUrl": "https://expenser.com/winter-2020-release-notes.html"
  },
  ],
  {
    "namespace": "db_exp_manager",
    "sfdcLoginUrl": "https://login.salesforce.com",
    "sourceApiVersion": "47.0",
    "packageAliases": {
      "Expenser Schema": "0HoB00000004CzHKAU",
      "Expenser Schema@0.1.0-1": "04tB0000000719qIAA"
    }
  }
]
```

You can also use the `--postinstallurl` and the `--releasenotesurl` Salesforce CLI parameters with the `force:package:version:create` command. The CLI parameters override the URLs specified in the `sfdx-project.json` file.

Publishing Your App on AppExchange

If you've published a first-generation managed package, you'll notice the process for publishing a second-generation managed package (managed 2GP) is different. After you link your Dev Hub org to the AppExchange publishing console, all your released managed 2GP package versions are visible in the publishing console.

To list an app on AppExchange, it must pass the AppExchange security review. For more information, see [Pass the AppExchange Security Review](#).

Link Dev Hub to the AppExchange Publishing Console

- Log in to the [Salesforce Partner Community](#).
- Select the **Publishing** tab, and then select **Organizations**.
- Click **Connect Org**, and enter the login credentials for your Dev Hub org.

Register Your Managed 2GP Package

- From Packages tab, locate the package version you want to register, and click **Register Package** in the Licenses column. Registering a package links the package to your [license management app](#).
- Enter the login credentials for the Dev Hub org associated with the package, in the modal window.
- Set the default license behavior for the package, including trial length, and number of seats included with the license, and click **Save**.

Packages that share a namespace can be associated the same License Management Org (LMO), or you can associate the packages with different LMOs.

Push a Package Upgrade for Second-Generation Managed Packages

Push upgrades enable you to upgrade second-generation managed packages installed in subscriber orgs, without asking customers to install the upgrade themselves. You can choose which orgs receive a push upgrade, what version the package is upgraded to, and when you want the upgrade to occur. Push upgrades are helpful if you need to push a change for a hot bug fix.

Use SOAP API to initiate the push upgrade, track the status of each job, and review error messages if any push upgrades fail. Here are the objects that help with push upgrades.

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

To Do This:	Use This Object:
Retrieve details about your package version.	MetadataPackageVersion SOAP API
Retrieve information about the subscriber org, such as the org ID and the package version currently installed.	PackageSubscriber SOAP API
Schedule a push upgrade, or check the status of the push upgrade.	PackagePushRequest SOAP API
Specify the org to receive the push upgrade. Create an individual package push job for every org receiving the push upgrade.	PackagePushJob SOAP API
Review any error messages associated with a push upgrade request.	PackagePushError SOAP API

Push Upgrade Considerations for Second-Generation Managed Packages

- You can push upgrades to only packages that have passed AppExchange security review.
- The same manageability rules for package version upgrades are applicable to push upgrades.
- When a push upgrade is installed, the Apex in package is complied.
- Push upgrades can be used even if the package version requires a password.

Push Upgrade Considerations for Unlocked Packages

- You can include new and changed features, or remove features during a push upgrade.
- When a push upgrade is installed, the Apex in the package is complied.
- You can use push upgrades even if the package version requires a password.

[Schedule a Push Upgrade Using SOAP API for First- and Second-Generation Managed Packages](#)

[Assign Access to New and Changed Features in First- and Second-Generation Managed Packages](#)

[Sample Post Install Script for a Push Upgrade for First- and Second-Generation Managed Packages](#)

Schedule a Push Upgrade Using SOAP API for First- and Second-Generation Managed Packages

For code samples and more detailed steps, see SOAP API object documentation linked in each step.

1. Authenticate to your [Dev Hub org](#).
2. Query [MetadataPackage](#) to verify package details.
3. Query [MetadataPackageVersion](#) to verify the package version to use for the push upgrade.
4. Query [PackageSubscriber](#) to retrieve details about subscriber orgs such as the org ID and installed package version. To retrieve information about more than 2,000 subscribers, use SOAP API [queryMore \(\)](#) call.
5. Create a [PackagePushRequest](#) object. Specify the `PackageVersionId` and `ScheduledStartTime` (optional). If you omit the `ScheduledStartTime`, the push begins when you set the `PackagePushRequest`'s status to `Pending`.
6. Create a [PackagePushJob](#) for each subscriber and associate it with the `PackagePushRequest` you created in the previous step.
7. Schedule the push upgrade by changing the status of the `PackagePushRequest` to `Pending`.

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	<p>Notify admins about the changes the upgrade introduces, and ask them to assign permissions to all users of the package.</p> <p>This approach allows admins to choose when to make the new features available.</p>

If the push upgrade includes:	We recommend you:
Enhancements to existing features	<p>Include a post-install script in the package that assigns permissions to the new components or new fields automatically.</p> <p>This approach ensures that current users of the package can continue using features without interruption.</p> <p> Note: Post-install scripts aren't available to unlocked packages.</p>

Sample Post Install Script for a Push Upgrade for First- and Second-Generation Managed Packages

 **Note:** Post-install scripts can be used with first and second-generation managed packages only.

Automate the assignment of new components to existing users of a package. For more information on writing a post-install Apex script, see Run Apex on Package Install/Upgrade.

In this sample script, the package upgrade contains new Visualforce pages and a new permission set that grants access to those pages. The script performs the following actions.

- Gets the Id of the Visualforce pages in the old version of the package
- Gets the permission sets that have access to those pages
- Gets the list of profiles associated with these permission sets
- Gets the list of users who have those profiles assigned
- Assigns the permission set in the new package to those users

```
global class PostInstallClass implements InstallHandler {
    global void onInstall(InstallContext context) {

        //Get the Id of the Visualforce pages
        List<ApexPage> pagesList = [SELECT Id FROM ApexPage WHERE NamespacePrefix =
            'TestPackage' AND Name = 'vfpagel'];

        //Get the permission sets that have access to those pages
        List<SetupEntityAccess> setupEntityAccessList = [SELECT Id,
            ParentId, SetupEntityId, SetupEntityType FROM SetupEntityAccess
            WHERE SetupEntityId IN :pagesList];
        Set<ID> PermissionSetList = new Set<ID> ();

        for (SetupEntityAccess sea : setupEntityAccessList) {
            PermissionSetList.add(sea.ParentId);
        }
        List<PermissionSet> PermissionSetWithProfileIdList =
            [SELECT id, Name, IsOwnedByProfile, Profile.Name,
            ProfileId FROM PermissionSet WHERE IsOwnedByProfile = true
            AND Id IN :PermissionSetList];

        //Get the list of profiles associated with those permission sets
        Set<ID> ProfileList = new Set<ID> ();
        for (PermissionSet per : PermissionSetWithProfileIdList) {
```

```
    ProfileList.add(per.ProfileId);
}

//Get the list of users who have those profiles assigned
List<User> UserList = [SELECT id FROM User where ProfileId IN :ProfileList];

//Assign the permission set in the new package to those users
List<PermissionSet> PermissionSetToAssignList = [SELECT id, Name
    FROM PermissionSet WHERE Name='TestPermSet' AND
    NamespacePrefix = 'TestPackage'];
PermissionSet PermissionSetToAssign = PermissionSetToAssignList[0];
List<PermissionSetAssignment> PermissionSetAssignmentList = new
List<PermissionSetAssignment>();
for (User us : UserList) {
    PermissionSetAssignment psa = new PermissionSetAssignment();
    psa.PermissionSetId = PermissionSetToAssign.id;
    psa.AssigneeId = us.id;
    PermissionSetAssignmentList.add(psa);
}
insert PermissionSetAssignmentList;
}

// Test for the post install class
@isTest
private class PostInstallClassTest {
    @isTest
    public static void test() {
        PostInstallClass myClass = new PostInstallClass();
        Test.testInstall(myClass, null);
    }
}
```

Advanced Features for Second-Generation Managed Packages

After you're comfortable with creating second-generation managed packages, learn about these advanced features to customize your package development processes.

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

Second-generation managed packaging (managed 2GP) offers a flexible linear package versioning model by letting you break your linear versioning and abandon a package version you no longer want to build upon. We call these versioning decisions *package ancestry*.

[Patch Versions for Second-Generation Managed Packages](#)

Patch versions of a second-generation managed package are a way to fix small issues with your package without introducing major feature changes. Customers who are using an older version of your package can install a patch and not be forced to upgrade to a new major package version.

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

To avoid monolithic package development practices, you plan to develop smaller, modular packages that group similar functionality and components. You can then define the dependencies between these packages. A package dependency is when metadata contained in one package depends on metadata contained in another package. For example, defining dependencies allow you to extend the functionality of a base package with components and metadata located in a separate package.

[Advanced Project Configuration Parameters for Second-Generation Managed Packages](#)

As your managed 2GP package development becomes more complex, consider including these optional parameters in your `sfdx-project.json` file.

[Sample Script for Installing Second-Generation Managed Packages with Dependencies](#)

Use this sample script as a basis to create your own script to install second-generation managed packages with dependencies. This script contains a query that finds dependent packages and installs them in the correct dependency order.

[Skip Validation to Quickly Iterate Second-Generation Package Development](#)

Iterate second-generation managed package development more efficiently by skipping validation of dependencies, package ancestors, and metadata during package version creation. Skipping validation reduces the time it takes to create a new package version, but you can promote only validated package versions to the released state.

[Second-Generation Managed Packaging Keywords](#)

A keyword is a variable that you can use to specify a package version number.

[Target a Specific Release for Your Second-Generation Managed Packages During Salesforce Release Transitions](#)

During major Salesforce release transitions, you can specify `preview` or `previous` when creating a package version. Specifying the release version for a package allows you to test upcoming features, run regression tests, and support customers regardless of which Salesforce release their org is on. Previously, you could only create package versions that matched the Salesforce release your Dev Hub org was on.

[Use Branches in Second-Generation Managed Packaging](#)

Development teams who use branches in their source control system (SCS), often build package versions based on the metadata in a particular branch of code.

[Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages](#)

For scenarios where you require metadata that isn't part of your second-generation managed package, but is necessary for Apex test runs, you can specify the path containing unpackaged metadata in the `sfdx-project.json` file. The unpackaged metadata isn't included in the package and isn't installed in subscriber orgs.

[Package IDs and Aliases for Second-Generation Managed Packages](#)

During the package lifecycle, packages and package versions are identified by an ID or package alias. When you create a second-generation managed package or package version, Salesforce CLI creates a package alias based on the package name, and stores that name in the `packageAliases` section of the `sfdx-project.json` file. When you run CLI commands or write scripts to automate packaging workflows, it's often easier to reference the package alias, instead of the package ID or package version ID.

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

Namespaces impact the combination of package types you can install in an org.

[Customize Second-Generation Managed Package Installs and Uninstalls Using Scripts](#)

Customize a second-generation managed package (managed 2GP) install or upgrade by specifying an Apex post install script to run automatically after a subscriber installs or upgrades a managed 2GP package. You can also specify an Apex uninstall script to run automatically when a subscriber uninstalls a managed 2GP package.

[Behavior of Specific Metadata in Second-Generation Managed Packages](#)

Learn how profiles and namespace visibility are handled for second-generation managed packages.

[Remove Metadata Components from Second-Generation Managed Packages](#)

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

Delete a Second-Generation Managed Package or Package Version

Use the `force:package:version:delete` and `force:package:delete` commands to delete packages and package versions that you no longer need.

Frequently Used Packaging Operations for Second-Generation Managed Packages

Transfer a Second-Generation Managed Package to a Different Dev Hub

You can transfer the ownership of a second-generation managed package from one Dev Hub org to another. These transfers can occur either internally between two Dev Hub orgs your company owns, or you can transfer a package externally to another Salesforce Partner or ISV. This change provides a way to sell a second-generation managed package to a different company.

Package Ancestors for Second-Generation Managed Packages

Second-generation managed packaging (managed 2GP) offers a flexible linear package versioning model by letting you break your linear versioning and abandon a package version you no longer want to build upon. We call these versioning decisions *package ancestry*.

When package versioning is linear, the package version number (formatted as major.minor.patch.build) always increments to an increasing number. For example, looking at just the major and minor version numbers, linear versioning looks something like 1.0 1.1 1.2 2.0. The next package version created in this linear versioning example must be higher than 2.0.



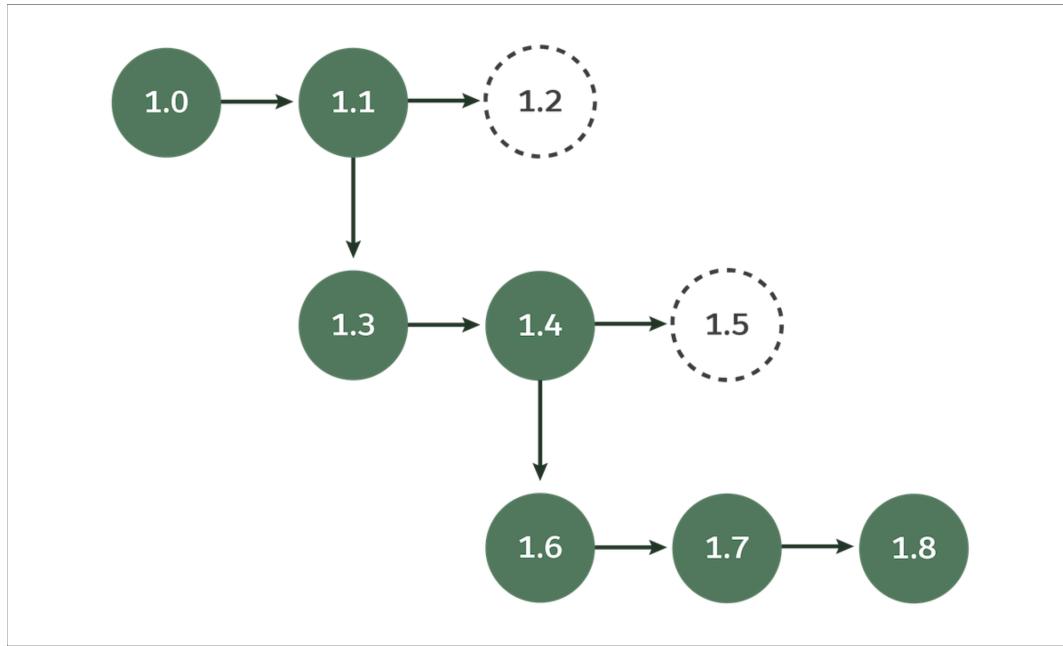
Note: For simplicity, we aren't discussing patch versioning here. See [Create a Patch Version](#) for information on how patch versioning works.

How Managed 2GP Package Versioning Affects Package Upgrades

Before we dig into package ancestry and how managed 2GP lets you break your linear versioning, let's clarify how package versioning impacts package upgrades. Let's use our previous example of a package version history that looks like this, 1.0 1.1 1.2 2.0. A customer could install version 1.0 and upgrade through each of the subsequent package versions, or they could skip versions and upgrade from say 1.0 to 2.0. As long as they upgrade from a lower package version number to a higher package version number, the package upgrade succeeds.

But what if during your development process you create a package version that you don't want to build upon? Managed 2GP lets you break free from linear versioning and select a different package version to build upon.

Say your team creates version 1.0, then 1.1, then 1.2 and oops! 1.2 made a mess of 1.1. Not a problem. When you create a package version, you specify which package version is the ancestor. So you abandon 1.2, and make 1.1 the ancestor of 1.3. And this process can be repeated. For example, the illustration shows how to abandon 1.5, and build 1.6 off 1.4.



This more complex and tree-like versioning has a secondary benefit, because it makes it possible for two or more development teams to do parallel package development.

With Great Power Comes Great Responsibility

The flexibility to break from linear versioning is powerful, but remember that if abandoned versions like 1.2 and 1.5 are installed in customer orgs, those customers no longer have an upgrade path. Packages can only upgrade along the ancestry line. For example, you can upgrade from version 1.1 to 1.7, but not from version 1.5 to 1.7.

 **Note:** You can specify only package versions that have been promoted to managed-released state as an ancestor.

Understanding Package Upgrades with Ancestry

Review how package ancestry impacts which package version upgrades are allowed.

[View Package Ancestry](#)

Use Salesforce CLI commands to quickly confirm your package's ancestor, or to create a visualization of the package ancestry tree.

Understanding Package Upgrades with Ancestry

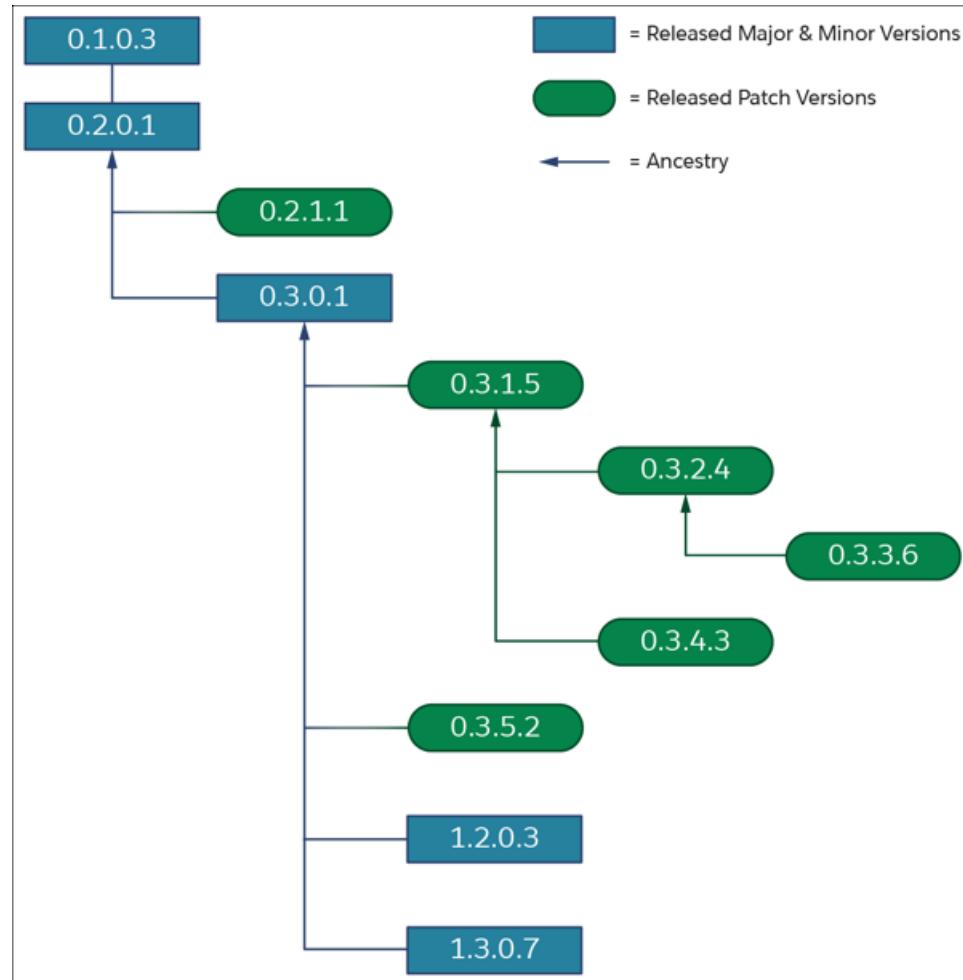
Review how package ancestry impacts which package version upgrades are allowed.

Refer to this table and the package ancestry tree to understand whether your subscribers can upgrade between these 2GP package versions.

Upgrade From	Upgrade To	Will This Package Upgrade Succeed?
0.2.0.1	0.3.4.3	Yes
0.1.0.3	0.3.5.2	Yes
0.3.3.6	0.3.5.2	Yes. Ancestry isn't enforced for patch version upgrades that occur between package

Upgrade From	Upgrade To	Will This Package Upgrade Succeed?
		versions that share the same major and minor package version numbers. In this example, both versions begin with 0.3.
0.2.0.1	1.2.0.3	Yes
0.3.0.1	0.1.0.3	No. Downgrading an installed package isn't allowed.
0.3.4.3	0.3.3.6	No. Downgrading an installed package isn't allowed.
1.2.0.3	1.3.0.7	No. To upgrade to 1.3.0.7, package version 1.2.0.3 must be the specified ancestor, or the specified ancestor must be a descendent of 1.2.0.3.

Example Package Ancestry Tree



View Package Ancestry

Use Salesforce CLI commands to quickly confirm your package's ancestor, or to create a visualization of the package ancestry tree.

View Package Ancestor Details in Salesforce CLI

For package versions created in Spring '20 or later, use the `package:version:report` or `package:version:list` command to view the name and version number of the package ancestor.

Output from `package:version:report` command.

==== Package Version	
Name	Value
Name	ver 0.2
Subscriber Package Version Id	04txx00000040T0AAM
Package Id	0Hoxx0000004G2yCAE
Version	0.2.0.1
Description	
Branch	
Tag	
Released	false
Validation Skipped	false
Ancestor	04txx0000004MnmAAE
Ancestor Version	0.1.0.2
Code Coverage	17%
Code Coverage Met	false

Output from `package:version:list` command.

Validation Skipped	Ancestor	Ancestor Version	Branch
false			
false	04txx0000004M10AAU	0.7.0.1	

Visualize Package Ancestry

Use the `displayancestry` CLI command to create visualizations of your package or package version's ancestry tree. You can view the visualization in Salesforce CLI or use the `.dotcode` parameter to generate output that can be used in graph visualization software.

Use `sfdx force:package:version:displayancestry` to quickly visualize your package ancestry and understand the possible package upgrade paths.

```
$ sfdx force:package:version:displayancestry --package 0Hoxx0000004V00CAU
  0.1.0.1
    0.2.0.1
      0.2.0.2
        0.3.0.1
        0.3.0.2
        0.3.0.3
          0.4.0.1
            1.0.0.1
              1.1.0.1
              1.0.0.2
```

```
$ sfdx force:package:version:displayancestry --package 04txx0000004k94AAA  
0.2.0.2 -> 0.1.0.1 (root)  
|  
+-- 0.2.0.2  
    +-- 0.3.0.1  
    +-- 0.3.0.2  
    +-- 0.3.0.3  
        +-- 0.4.0.1  
    +-- 1.0.0.1  
        +-- 1.1.0.1  
            +-- 1.0.0.2
```

To generate dotcode output, specify `sfdx force:package:version:displayancestry --dotcode`.

Patch Versions for Second-Generation Managed Packages

Patch versions of a second-generation managed package are a way to fix small issues with your package without introducing major feature changes. Customers who are using an older version of your package can install a patch and not be forced to upgrade to a new major package version.

Package versions follow a major.minor.patch.build number format. Any package version number that contains a non-zero patch number is a patch version. For example, 1.1.2.5.

Patch versions are intended for minor changes. You can't:

- Add package components.
- Delete existing package components.
- Change the API and dynamic Apex access controls.
- Deprecate any Apex code.
- Add new Apex class relationships, such as extends.
- Add Apex access modifiers, such as virtual or global.
- Add features, settings, package dependencies, or web services.
- Change a component from protected to global.
- Change the visibility of CustomSettings or CustomMetadataType from protected to public.

When creating a patch version, you must specify the package ancestor. Keep in mind that the major and minor version number of the patch and the package ancestor must match. And the specified package ancestor must be managed-released.

You can specify another patch version as the package ancestor. See [Specify a Package Ancestor in the Project File for a Second-Generation Managed Package](#) for more information on how to specify a package ancestor.

When you create a patch version, the patch automatically inherits the features and settings defined in the package ancestor's external definition file. To create a patch, follow the same steps as you do when you create a package version, and increment the patch number.

 **Note:** To enable patch versioning, log a case in the [Salesforce Partner Community](#) and request patch versioning be enabled in the org where you created the namespace for this package.

Create Dependencies Between Second-Generation Managed Packages

To avoid monolithic package development practices, you plan to develop smaller, modular packages that group similar functionality and components. You can then define the dependencies between these packages. A package dependency is when metadata contained in one package depends on metadata contained in another package. For example, defining dependencies allow you to extend the functionality of a base package with components and metadata located in a separate package.

How to Specify a Managed 2GP Package Dependency

 **Note:** To understand which combination of managed 2GP and managed 1GP package dependencies are supported, see [Understand Managed 2GP Package Dependencies](#).

To specify dependencies between managed packages associated with the same Dev Hub, use either the package version alias or a combination of the package name and the version number.

Example 1:

```
"dependencies": [
  {
    "package": "MyPackageName@0.1.0.1"
  }
]
```

Example 2:

```
"dependencies": [
  {
    "package": "MyPackageName",
    "versionNumber": "0.1.0.LATEST"
  }
]
```

To specify a dependency on a managed package that isn't associated with your Dev Hub:

```
"dependencies": [
  {
    "package": "OtherOrgPackage@1.2.0"
  }
]
```

 **Note:** You can use the LATEST keyword for the version number to set the dependency.

To denote dependencies with package IDs instead of package aliases, use:

- The `0Ho` ID if you specify the package ID along with the version number
- The `04t` ID if you specify only the package version ID

Specifying Multiple Package Dependencies

If your package has more than one dependency, provide a comma-separated list of packages in the order of installation.

For example, if your package depends on the package Expense Manager - Util, which in turn depends on the package External Apex Library, the package dependencies are:

```
"dependencies": [
  {
    "package": "External Apex Library - 1.0.0.4"
  },
  {
    "package": "Expense Manager - Util",
    "versionNumber": "4.7.0.LATEST"
  }
]
```

```
    }  
]
```

Which Types of Dependencies Are Supported?

Circular Dependencies

Circular dependencies among packages aren't supported.

A circular dependency occurs when pkgC depends on pkgB, pkgB depends on pkgA, and pkgA depends on pkgC.

Multi-level Dependencies

Multi-level package dependencies are supported.

A multi-level dependency occurs when pkgC depends on pkgB, and pkgB depends on pkgA.

List multi-level dependencies in the `sfdx-project.json` file in package installation order. In this example, pkgA must be installed first, followed by pkgB, and then pkgC.

```
{  
  "packageDirectories": [  
    {  
      "path": "pkgA-wsp",  
      "default": true,  
      "package": "pkgA",  
      "versionName": "ver 0.9",  
      "versionNumber": "0.9.0.NEXT",  
      "ancestorVersion": "0.7.0.1"  
    },  
    {  
      "path": "pkgB-wsp",  
      "default": false,  
      "package": "pkgB",  
      "versionName": "ver 0.3",  
      "versionNumber": "0.3.0.NEXT",  
      "dependencies": [  
        {  
          "package": "pkgA@0.9.0.LATEST"  
        }  
      ]  
    },  
    {  
      "path": "pkgC-wsp",  
      "default": false,  
      "package": "pkgC",  
      "versionName": "ver 0.1",  
      "versionNumber": "0.1.0.NEXT",  
      "dependencies": [  
        {  
          "package": "pkgA@0.9.0.LATEST"  
        },  
        {  
          "package": "pkgB@0.3.0.LATEST"  
        }  
      ]  
    }  
  ]
```

```

        }
    ],
}

```

The specified package version number also impacts the installation of package dependencies. Before pkgB can be installed, pkgA version 0.9 or higher must first be installed. If this condition isn't met, the installation of pkgB fails.

Advanced Project Configuration Parameters for Second-Generation Managed Packages

As your managed 2GP package development becomes more complex, consider including these optional parameters in your `sfdx-project.json` file.

Name	Details
apexTestAccess	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>Assign permission sets and permission set licenses to the user in context when your Apex tests run at package version creation.</p> <pre> "apexTestAccess": { "permissionSets": ["Permission_Set_1", "Permission_Set_2"], "permissionSetLicenses": ["SalesConsoleUser"] } </pre> <p>See Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages</p>
branch	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>If your package has an associated branch, but your package dependency is associated with a different branch, use this format.</p> <pre> "dependencies": [{ "package": "pkgB", "versionNumber": "1.3.0.LATEST", "branch": "featureC" }] </pre>

Name	Details
	<p>If your package has an associated branch, but your package dependency doesn't have an associated branch, use this format.</p> <pre>"dependencies": [{ "package": "pkgB", "versionNumber": "1.3.0.LATEST", "branch": "" }]</pre> <p>See Use Branches in Second-Generation Managed Packaging</p>
dependencies	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>Specify the dependencies on other packages.</p> <p>To specify dependencies for managed packages within the same Dev Hub, use either the package version alias or a combination of the package name and the version number.</p> <pre>"dependencies": [{ "package": "MyPackageName@0.1.0.1" }]</pre> <pre>"dependencies": [{ "package": "MyPackageName", "versionNumber": "0.1.0.LATEST" }]</pre> <p>To specify dependencies for managed packages outside of the Dev Hub use:</p> <pre>"dependencies": [{ "package": "OtherOrgPackage@1.2.0" }]</pre> <p> Note: You can use the LATEST keyword for the version number to set the dependency.</p> <p>To denote dependencies with package IDs instead of package aliases, use:</p> <ul style="list-style-type: none"> • The 0Ho ID if you specify the package ID along with the version number • The 04t ID if you specify only the package version ID

Name	Details
	<p>If the package has more than one dependency, provide a comma-separated list of packages in the order of installation. For example, if a package depends on the package Expense Manager - Util, which in turn depends on the package External Apex Library, the package dependencies are:</p> <pre>"dependencies": [{ "package" : "External Apex Library - 1.0.0.4" }, { "package": "Expense Manager - Util", "versionNumber": "4.7.0.LATEST" }]</pre>
postInstallScript	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>An Apex script that runs automatically in the subscriber org after the managed package is installed or upgraded.</p>
postInstallURL	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>A URL to post-install instructions for subscribers.</p>
releaseNotesUrl	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>A URL to release notes.</p>
unpackagedMetadata	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>See Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages.</p>
uninstallScript	<p>Required? No</p> <p>Default if Not Specified: None</p> <p>An Apex script to run automatically in the subscriber org before the managed package is uninstalled.</p>

Sample Script for Installing Second-Generation Managed Packages with Dependencies

Use this sample script as a basis to create your own script to install second-generation managed packages with dependencies. This script contains a query that finds dependent packages and installs them in the correct dependency order.

Sample Script

 **Note:** Be sure to replace the package version ID and scratch org user name with your own specific details.

```
#!/bin/bash

# The execution of this script stops if a command or pipeline has an error.

# For example, failure to install a dependent package will cause the script
# to stop execution.

set -e

# Specify a package version id (starts with 04t)

# If you know the package alias but not the id, use force:package:version:list to find it.

PACKAGE=04tB0000000NmnHIAS

# Specify the user name of the subscriber org.

USER_NAME=test-bvdfz3m9tqdf@example.com

# Specify the timeout in minutes for package installation.

WAIT_TIME=15

echo "Retrieving dependencies for package Id: \"$PACKAGE"

# Execute soql query to retrieve package dependencies in json format.

RESULT_JSON=`sfdx force:data:soql:query -u $USER_NAME -t -q "SELECT Dependencies FROM
SubscriberPackageVersion WHERE Id='\$PACKAGE'" --json`

# Parse the json string using python to test whether the result json contains a list of
ids or not.

DEPENDENCIES=`echo $RESULT_JSON | python -c 'import sys, json; print
```

```
json.load(sys.stdin) ["result"] ["records"] [0] ["Dependencies"] ``

# If the parsed dependencies is None, the package has no dependencies. Otherwise, parse
# the result into a list of ids.

# Then loop through the ids to install each of the dependent packages.

if [[ "$DEPENDENCIES" != 'None' ]]; then

    DEPENDENCIES=`echo $RESULT_JSON | python -c '``

import sys, json

ids = json.load(sys.stdin) ["result"] ["records"] [0] ["Dependencies"] ["ids"]

dependencies = []

for id in ids:

    dependencies.append(id["subscriberPackageVersionId"])

print " ".join(dependencies)

```

echo "The package you are installing depends on these packages (in correct dependency
order): "$DEPENDENCIES

for id in $DEPENDENCIES

do

 echo "Installing dependent package: "$id

 sfdx force:package:install --package $id -u $USER_NAME -w $WAIT_TIME --publishwait
10

done

else

 echo "The package has no dependencies"

fi

After processing the dependencies, proceed to install the specified package.

echo "Installing package: "$PACKAGE
```

```
sfdx force:package:install --package $PACKAGE -u $USER_NAME -w $WAIT_TIME --publishwait
10

exit 0;
```

## Skip Validation to Quickly Iterate Second-Generation Managed Package Development

Iterate second-generation managed package development more efficiently by skipping validation of dependencies, package ancestors, and metadata during package version creation. Skipping validation reduces the time it takes to create a new package version, but you can promote only validated package versions to the released state.

```
sfdx force:package:version:create --skipvalidation
```

In Tooling API, use the [SkipValidation](#) field on the [Package2VersionCreateRequest](#) object.

 **Note:** You can't specify both skip validation and code coverage, because code coverage is calculated during validation.

## Second-Generation Managed Packaging Keywords

A keyword is a variable that you can use to specify a package version number.

You can use keywords to automatically increment the value of the package build numbers, ancestor version numbers, set the package dependency to the latest version, or the latest released and promoted version.

| Use the Keyword                                                                                                                                                      | Example                                                                                                    |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| LATEST to specify the latest version of the package dependency when you create a package version.                                                                    | <pre>"dependencies": [   {     "package": "MyPackageName",     "versionNumber": "0.1.0.LATEST"   } ]</pre> |
| NEXT to increment the build number to the next available for the package version.                                                                                    | <pre>"versionNumber": "1.2.0.NEXT"</pre>                                                                   |
| RELEASED to specify the latest promoted and released version of the package dependency when you create a package version.                                            | <pre>"dependencies": [   {     "package": "pkgB",     "versionNumber": "2.1.0.RELEASED"   } ]</pre>        |
| HIGHEST to automatically set the package ancestor to the highest promoted and released package version number.<br><br>Use only with ancestor version or ancestor ID. | <pre>"packageDirectories": [   {     "path": "util",     "ancestor": "HIGHEST"   } ]</pre>                 |

| Use the Keyword                                                                                                                                                                                              | Example                                                                                                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                              | <pre>"package": "Expense Manager - Util", "versionNumber": "4.7.0.NEXT", "ancestorVersion": HIGHEST },</pre>                                        |
| <p>NONE in the ancestor version or ancestor ID field.</p> <p>Ancestry defines package upgrade paths. If the package ancestor is set to NONE, an existing customer can't upgrade to that package version.</p> | <pre>"packageDirectories": [ { "path": "util", "package": "Expense Manager - Util", "versionNumber": "4.7.0.NEXT", "ancestorVersion": NONE },</pre> |

## Target a Specific Release for Your Second-Generation Managed Packages During Salesforce Release Transitions

During major Salesforce release transitions, you can specify `preview` or `previous` when creating a package version. Specifying the release version for a package allows you to test upcoming features, run regression tests, and support customers regardless of which Salesforce release their org is on. Previously, you could only create package versions that matched the Salesforce release your Dev Hub org was on.

To create a package version based on a preview or previous Salesforce release version, create a scratch org definition file that includes either:

```
{
 "release": "previous"
}
```

or

```
{
 "release": "preview"
}
```

In the `sfdx-project.json` file, set the `sourceApiVersion` to correspond with the release version of the package version you're creating. If you are targeting a previous release, any `sourceApiVersion` value below the current release is accepted.

Then when you create your package version, specify the scratch org definition file.

```
sfdx force:package:version:create --package pkgA --definitionfile
config/project-scratch-def.json
```

Preview start date is when sandbox instances are upgraded. Preview end date is when all instances are on the GA release.

| Release Version | Preview Start Date | Preview End Date  |
|-----------------|--------------------|-------------------|
| Spring '23      | January 8, 2023    | February 11, 2023 |
| Summer '23      | May 7, 2023        | June 10, 2023     |
| Winter '24      | September 10, 2023 | October 14, 2023  |

## Use Branches in Second-Generation Managed Packaging

Development teams who use branches in their source control system (SCS), often build package versions based on the metadata in a particular branch of code.

To identify which branch in your SCS a package version is based on, tag your package version with a branch name using `--branch` attribute in this Salesforce CLI command.

```
sfdx force:package:version:create --branch featureA
```

You can specify any alphanumeric value up to 240 characters as the branch name.

You can also specify the branch name in the package directories section of the `sfdx-project.json` file.

```
"packageDirectories": [
 {
 "path": "util",
 "default": true,
 "package": "pkgA",
 "versionName": "Spring '21",
 "versionNumber": "4.7.0.NEXT",
 "branch": "featureA"
 }
]
```

When you specify a branch, the package alias for that package version is automatically appended with the branch name. You can view the package alias in the `sfdx.project.json` file.

```
"packageAliases": {
 "pkgA@1.0.0-4-featureA": "04tB0000000IB1EIAW"}
```

Keep in mind that version numbers increment within each branch, and not across branches. For example, you could have two or more beta package versions with the version number 1.3.0.1.

| Branch Name   | Package Version Alias |
|---------------|-----------------------|
| featureA      | pkgA@1.3.0-1-featureA |
| featureB      | pkgA@1.3.0-1-featureB |
| Not specified | pkgA@1.3.0-1          |

Although more than one beta package version can have the same version number, there can be only one promoted and released package version for a given major.minor.patch package version.

## Package Dependencies and Branches

By default, your package can have dependencies on other packages in the same branch. For package dependencies based on packages in other branches, explicitly set the branch attribute in the `sfdx-project.json` file.

| To specify a package dependency | Use this format                                         |
|---------------------------------|---------------------------------------------------------|
| Using the branch attribute      | <pre>"dependencies": [   {     "package": "pkgB",</pre> |

| To specify a package dependency                                                           | Use this format                                                                                    |
|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
|                                                                                           | <pre>"versionNumber": "1.3.0.LATEST", "branch": "featureC" }]</pre>                                |
| Using the most recent promoted and released version of package                            | <pre>"dependencies": [ { "package": "pkgB", "versionNumber": "2.1.0.RELEASED" }]</pre>             |
| If your package has an associated branch, but the dependent package doesn't have a branch | <pre>"dependencies": [ { "package": "pkgB", "versionNumber": "1.3.0.LATEST", "branch": "" }]</pre> |
| Using the package alias                                                                   | <pre>"dependencies": [ { "package": "pkgB@2.1.0-1-featureC" }]</pre>                               |

## Specify Unpackaged Metadata or Apex Access for Package Version Creation Tests for Second-Generation Managed Packages

For scenarios where you require metadata that isn't part of your second-generation managed package, but is necessary for Apex test runs, you can specify the path containing unpackaged metadata in the `sfdx-project.json` file. The unpackaged metadata isn't included in the package and isn't installed in subscriber orgs.

### Specify Unpackaged Metadata for Package Version Creation Tests

Specify the path to the unpackaged metadata in your `sfdx-project.json` file.

In this example, metadata in the `my-unpackaged-directory` is available for test runs during the package version creation of the `TV_unl` package.

```
"packageDirectories": [
{
 "path": "force-app",
 "package": "TV_unl",
 "versionName": "ver 0.1",
 "versionNumber": "0.1.0.NEXT",
 "default": true,
 "unpackagedMetadata": {
 "path": "my-unpackaged-directory"
 }
}]
```

```
 },
]
```

The `unpackagedMetadata` attribute is intended for metadata that isn't part of your package. You can't include the same metadata in both an unpackaged directory and a packaged directory.

## Manage Apex Access for Package Version Creation Tests

Sometimes the Apex tests that you write require a user to have certain permission sets or permission set licenses. Use the `apexTestAccess` setting to assign permission sets and permission set licenses to the user in whose context your Apex tests get run at package version creation.

```
"packageDirectories": [
 {
 "path": "force-app",
 "package": "TV_unl",
 "versionName": "ver 0.1",
 "versionNumber": "0.1.0.NEXT",
 "default": true,
 "unpackagedMetadata": {
 "path": "my-unpackaged-directory"
 },
 "apexTestAccess": {
 "permissionSets": [
 "Permission_Set_1",
 "Permission_Set_2"
],
 "permissionSetLicenses": [
 "SalesConsoleUser"
]
 }
 },
]
```

 **Note:** To assign user licenses, use the [rusAs Method](#). User licenses can't be assigned in the `sfdx-project.json` file.

## Package IDs and Aliases for Second-Generation Managed Packages

During the package lifecycle, packages and package versions are identified by an ID or package alias. When you create a second-generation managed package or package version, Salesforce CLI creates a package alias based on the package name, and stores that name in the `packageAliases` section of the `sfdx-project.json` file. When you run CLI commands or write scripts to automate packaging workflows, it's often easier to reference the package alias, instead of the package ID or package version ID.

Package aliases are stored in the `sfdx-project.json` file as name-value pairs, in which the name is the alias and the value is the ID. You can modify package aliases for existing packages and package versions in the project file.

At the command line, you also see IDs for things like package members (a component in a package) and requests (like a `package:version:create` request).

 **Note:** As a shortcut, the documentation sometimes refers to an ID by its three-character prefix. For example, a package version ID always starts with 04t.

Here are the most commonly used IDs.

| ID Example         | Short ID Name                 | Description                                                                                                                                                                                                   |
|--------------------|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 033J0000dAb27uxVRE | Subscriber Package ID         | Use this ID when contacting Salesforce for packaging or security review support. To locate this ID for your package, run <code>force:package:list --verbose</code> against the Dev Hub that owns the package. |
| 04t6A0000004eytQAA | Subscriber Package Version ID | Use this ID to install a package version. Returned by <code>force:package:version:create</code> .                                                                                                             |
| 0Hoxx00000000CqCAI | Package ID                    | Use this ID on the command line to create a package version. Or enter it into the <code>sfdx-project.json</code> file and use the directory name. Generated by <code>force:package:create</code> .            |
| 08cxx00000000BEAAY | Version Creation Request ID   | Use this ID to view the status and monitor progress for a specific request to create a package version such as <code>force:package:version:create:report</code>                                               |

## Avoid Namespace Collisions in Second-Generation Managed Packages

Namespaces impact the combination of package types you can install in an org.

To understand how namespaces affect the types of packages you can install in a namespaced or no-namespace org, review this table.

| Installation Org            | No-namespace Unlocked Package                                                                                                              | Namespaced Unlocked Package                                                                       | Second-generation Managed Package (2GP)                                                                                                                             | First-generation Managed Package (1GP)                                                                                                                                                                                                                             |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Org with a namespace</b> | <p>Fail.</p> <p>For example, a 1GP packaging org, 1GP patch org, Developer Edition org with namespace, or a scratch org with namespace</p> | <p>Pass.</p> <p>You can't install a no-namespace unlocked package in an org with a namespace.</p> | <p>Pass.</p> <p>Regardless of whether the namespace matches or is different from the org's namespace, you can install one or many namespaced unlocked packages.</p> | <p>Pass.</p> <p>If the namespace of the 1GP is different from the namespace of the org, you can install one or many packages.</p> <p>Fail.</p> <p>If the namespace of the 1GP is the same as the namespace of the org, you can't install the 1GP into the org.</p> |

| Installation Org               | No-namespace Unlocked Package                                        | Namespaced Unlocked Package                                        | Second-generation Managed Package (2GP)            | First-generation Managed Package (1GP)             |
|--------------------------------|----------------------------------------------------------------------|--------------------------------------------------------------------|----------------------------------------------------|----------------------------------------------------|
| <b>Org without a namespace</b> | Pass.<br>You can install one or many no-namespace unlocked packages. | Pass.<br>You can install one or many namespaced unlocked packages. | Pass.<br>You can install one or many 2GP packages. | Pass.<br>You can install one or many 1GP packages. |

To understand how namespaces affect the combination of packages that can be installed into one org, review this table.

| Namespace and Package Type                                      | Unlocked Package with Namespace Y                                                                                                              | Second-generation Managed Package (2GP) with Namespace Y                                      | First-generation Managed Package (1GP) with Namespace Y                                              |
|-----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <b>First-generation Managed Package (1GP) with namespace X</b>  | Pass.<br>If the 1GP and unlocked package use unique namespaces, you can install them in the same org.                                          | Pass.<br>If the 1GP and 2GP use unique namespaces, you can install them in the same org.      | Pass.<br>If each 1GP uses a unique namespace, you can install multiple 1GP packages in the same org. |
| <b>First-generation Managed Package (1GP) with namespace Y</b>  | Fail.<br>If the 1GP and unlocked package share a namespace, you can't install them in the same org.                                            | Fail.<br>If the 1GP and 2GP share a namespace, you can't install them in the same org.        | Fail.<br>If the 1GP packages share a namespace, you can't install them in the same org.              |
| <b>Second-generation Managed Package (2GP) with namespace X</b> | Pass.<br>You can install a 2GP and a namespaced unlocked package in the same org. The packages can share a namespace or use unique namespaces. | Pass.<br>You can install multiple 2GP packages with unique namespaces, or the same namespace. | Pass.<br>If the 1GP and 2GP use unique namespaces, you can install them in the same org.             |
| <b>Second-generation Managed Package (2GP) with namespace Y</b> | Pass.<br>You can install a 2GP and a namespaced unlocked package in the same org. The packages can share a namespace or use unique namespaces. | Pass.<br>You can install multiple 2GP packages with the same namespace in the same org.       | Fail.<br>If the 1GP and 2GP share a namespace, you can't install them in the same org.               |

## Customize Second-Generation Managed Package Installs and Uninstalls Using Scripts

Customize a second-generation managed package (managed 2GP) install or upgrade by specifying an Apex post install script to run automatically after a subscriber installs or upgrades a managed 2GP package. You can also specify an Apex uninstall script to run automatically when a subscriber uninstalls a managed 2GP package.

For more information, see [Running Apex on Package Install/Upgrade](#) and [Running Apex on Package Uninstall](#).

Specify post install and uninstall scripts in the `sfdx-project.json` file.

```
"packageDirectories": [
 {
 "path": "expenser-schema",
 "default": true,
 "package": "Expense Schema",
 "versionName": "ver 0.3.2",
 "versionNumber": "0.3.2.NEXT",
 "postInstallScript": "PostInstallScript",
 "uninstallScript": "UninstallScript",
 "postInstallUrl": "https://expenser.com/post-install-instructions.html",
 "releaseNotesUrl": "https://expenser.com/winter-2020-release-notes.html"
 },
],
 {
 "namespace": "db_exp_manager",
 "sfdcLoginUrl": "https://login.salesforce.com",
 "sourceApiVersion": "47.0",
 "packageAliases": {
 "Expenser Schema": "0HoB00000004CzHKAU",
 "Expenser Schema@0.1.0-1": "04tB0000000719qIAA"
 }
 }
]
```

You can also use the `--postinstallscript` and the `--uninstallscript` Salesforce CLI parameters with the `force:package:create` command. The CLI parameters override the scripts specified in the `sfdx-project.json` file.

 **Note:** Include the Apex classes for your post-install and uninstall scripts with the metadata in your package.

You can designate an active Dev Hub org user to receive email notifications for Apex gacks, and install, upgrade, or uninstall failures associated with your packages. In Salesforce CLI run `sfdx force:package:create --errornotificationusername me@devhub.org` or `sfdx force:package:update --errornotificationusername me@devhub.org`. In Tooling API, use the `PackageErrorUsername` field on the `Package2` object.

## Behavior of Specific Metadata in Second-Generation Managed Packages

Learn how profiles and namespace visibility are handled for second-generation managed packages.

### [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, My Domain settings, and whether enhanced domains are enabled.

### How We Handle Profile Settings in Second-Generation Managed Packages

When you package a profile in an unlocked or second-generation managed package, the build system inspects the contents of the profile during package creation and preserves only the profile settings that are directly related to the metadata in the package. The profile itself, and any profile settings unrelated to the package's metadata are discarded from the package.

### Namespace-Based Visibility for Apex Classes in Second-Generation Managed Packages

The `@NamespaceAccessible` makes public Apex in a package available to other packages that use the same namespace. Without this annotation, Apex classes, methods, interfaces, and properties defined in a second-generation managed package aren't accessible to the other packages with which they share a namespace. Apex that is declared global is always available across all namespaces, and needs no annotation.

### Package Connected Apps in Second-Generation Packaging

Add a connected app to a second-generation managed package.

### Test and Respond to the New Order Save Behavior

If you created any type of package that includes the Order object, the installed package sometimes doesn't work, and package upgrades or new package installations are blocked. Here's why. The [Order Save Behavior Release Update](#) addresses an issue where Salesforce didn't correctly evaluate custom application logic on records associated with the Order object.

## 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, My Domain settings, and whether enhanced domains are enabled.



**Note:** Because enhanced domains meet the latest browser requirements, they're the future standard. However, until they're enabled in all Salesforce orgs, packages must accommodate all possible URL formats. If your package includes references to full hostnames or URLs, use dynamically generated hostnames to support your customers during the transition to enhanced domains.

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
```

## How We Handle Profile Settings in Second-Generation Managed Packages

When you package a profile in an unlocked or second-generation managed package, the build system inspects the contents of the profile during package creation and preserves only the profile settings that are directly related to the metadata in the package. The profile itself, and any profile settings unrelated to the package's metadata are discarded from the package.

During package installation, the preserved profile settings are applied only to existing profiles in the subscriber org. The profile itself is not installed in the subscriber org.



**Note:** Packages that contain only profiles and no additional metadata aren't allowed and fail during package version creation.

| When you select...            | The packaged profile settings are applied to...                                                                                                                                                                                                                                 | This installation option is available via...                                                                                                                                                                                                                                                                      |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Install for Admins Only       | <p>The System Administrator profile in the subscriber org.</p> <p>CRUD access to custom objects is granted automatically to the System Administration profile.</p>                                                                                                              | <ul style="list-style-type: none"> <li>The package installer page</li> <li>Salesforce CLI <code>sfdx force:package:install</code> command</li> </ul> <p>The default behavior for CLI-based package installs is to install for admins only.</p>                                                                    |
| Install for All Users         | <p>The System Administrator profile and all cloned profiles in the subscriber org.</p> <p>CRUD access to custom objects is granted automatically to the System Administration profile, and all cloned profiles.</p> <p><a href="#">Standard profiles</a> can't be modified.</p> | <ul style="list-style-type: none"> <li>The package installer page</li> <li>Salesforce CLI <code>sfdx force:package:install</code> command</li> </ul> <p>To install for all users via the CLI, include the security type parameter.</p> <pre><code>sfdx force:package:install --securitytype AllUsers</code></pre> |
| Install for Specific Profiles | Specific profiles in the subscriber org. This selection lets the person installing your package determine how to map the profile settings you packaged to specific profiles in their org.                                                                                       | <ul style="list-style-type: none"> <li>The package installer page</li> </ul> <p>Not available for CLI-based package installations.</p>                                                                                                                                                                            |

To test the behavior of your packaged profile, install your package in a scratch org.

- From Setup, enter `Profile` in the Quick Find box, and then locate and inspect the profiles you selected during package installation.
- Check whether your profile settings have been applied to that profile.

Repeat this step for any other profile you expect to contain your profile settings. Don't look for the profile name you created; we apply profile settings to existing profiles in the subscriber org.

Whenever possible, use package permission sets instead of profile settings. Subscribers who install your package can easily assign your permission set to their users.



**Note:** During a push upgrade, some profile settings related to Apex classes and field-level security aren't automatically assigned to the System Admin profile. Communicate with your customer to ensure that user access is set up correctly after a push upgrade. Make sure you review and update your profile settings after push upgrade.

## Retain License Settings in Unlocked Packages

By default, license settings in profiles are removed during package creation. To retain these settings, specify the `includeProfileUserLicenses` parameter in your `sfdx-project.json` file. In this scenario, the license settings are retained and applied to the profiles in the subscriber org that are selected during package installation.

```
"packageDirectories": [
 {
 "package": "PackageA",
 "path": "common",
 "versionName": "ver 0.1",
 "versionNumber": "0.1.0.NEXT",
 "default": false,
 includeProfileUserLicenses: true
 }
]
```

## Namespace-Based Visibility for Apex Classes in Second-Generation Managed Packages

The `@NamespaceAccessible` makes public Apex in a package available to other packages that use the same namespace. Without this annotation, Apex classes, methods, interfaces, and properties defined in a second-generation managed package aren't accessible to the other packages with which they share a namespace. Apex that is declared global is always available across all namespaces, and needs no annotation.

### Considerations for Apex Accessibility Across Packages

- You can't use the `@NamespaceAccessible` annotation for an `@AuraEnabled` Apex method.
- You can add or remove the `@NamespaceAccessible` annotation at any time, even on managed and released Apex code. Make sure that you don't have dependent packages relying on the functionality of the annotation before adding or removing it.
- When adding or removing `@NamespaceAccessible` Apex from a package, consider the impact to customers with installed versions of other packages that reference this package's annotation. Before pushing a package upgrade, ensure that no customer is running a package version that would fail to fully compile when the upgrade is pushed.
- If a public interface is declared as `@NamespaceAccessible`, then all interface members inherit the annotation. Individual interface members can't be annotated with `@NamespaceAccessible`.
- If a public or protected variable or method is declared as `@NamespaceAccessible`, its defining class must be either global or public with the `@NamespaceAccessible` annotation.
- If a public or protected inner class is declared as `@NamespaceAccessible`, its enclosing class must be either global or public with the `@NamespaceAccessible` annotation.

This example shows an Apex class marked with the `@NamespaceAccessible` annotation. The class is accessible to other packages within the same namespace. The first constructor is also visible within the namespace, but the second constructor isn't.

```
// A namespace-visible Apex class
@NamespaceAccessible
public class MyClass {
 private Boolean bypassFLS;

 // A namespace-visible constructor that only allows secure use
 @NamespaceAccessible
 public MyClass() {
 bypassFLS = false;
```

```
}

// A package private constructor that allows use in trusted contexts,
// but only internal to the package
public MyClass (Boolean bypassFLS) {
 this.bypassFLS = bypassFLS;
}
@NamespaceAccessible
protected Boolean getBypassFLS() {
 return bypassFLS;
}
}
```

## Package Connected Apps in Second-Generation Managed Packaging

Add a connected app to a second-generation managed package.

Prerequisites: [Create a connected app](#).

1. Create a first-generation managed package (1GP) and add the connected app. It's fine if the connected app is the only component in the package. Use the same namespace as the 2GP package for the 1GP package.  
Take note of the version number of the connected app; you'll use this number later.
2. From your packaging org, upload the 1GP package to create a package version.
3. Promote the 1GP version to the released state.  
Promoting the 1GP version allows the connected app to be included in a second-generation managed package. You don't need to install the 1GP version into an org.
4. Navigate to the source for your connected app, or pull the source from the org where the connected app is being developed.
5. Create a source .xml file in your 2GP directory and reference the connected app you want to include. See the *Sample Source File* section.
6. Create a second-generation managed package and add in the source code for the connected app. Add the source code manually. You can't use `force:source:pull` or the `retrieve()` Metadata API call to add the source code.



### Example: Sample Source File

```
<ConnectedApp xmlns="http://soap.sforce.com/2006/04/metadata">
 <developerName>db_0110_ns4__A_Connected_App</developerName>
 <label>A Connected App</label>
 <version>1.0</version>
</ConnectedApp>
```

The `developerName` is the combination of your namespace (db\_0110\_ns4) and the name of your connected app (A\_Connected\_App).

The `version` specified in the source file is the version number of the connected app. Use decimal formatting when specifying the version number. The version number must match the version number of the connected app before it was added to the 1GP managed package.



**Note:** When you add a connected app to a 1GP package, and upload the package, the version number of the connected app is auto incremented. For example, when version 4.0 of a connected app is added to a 1GP package, the package version increments the version number of the connected app from 4.0 to 5.0. When creating the source file for your 2GP package, specify the version number of the connected app before it was uploaded into a 1GP package, in this case, 4.0.

## Test and Respond to the New Order Save Behavior

If you created any type of package that includes the Order object, the installed package sometimes doesn't work, and package upgrades or new package installations are blocked. Here's why. The [Order Save Behavior Release Update](#) addresses an issue where Salesforce didn't correctly evaluate custom application logic on records associated with the Order object.

To ensure the expected behavior, you must test the Enable New Order Save Behavior release update. Starting in Winter '21, if a subscriber org relies on a different order save behavior than their installed packages, the installed packages sometimes don't work, and package upgrades or new package installations are blocked.

After the Enable New Order Save Behavior release update is enabled, Salesforce evaluates and runs these customizations whenever an update to an order item record changes the parent order record.

- Order and order item validation rules
- Order and order item Apex triggers and classes
- Order and order item workflow rules
- Order and order item flows and processes

 **Note:** The New Order Save Behavior release update affects all package types: unlocked, unmanaged, first-generation managed package (1GP), and second-generation managed package (2GP).

After you verify that your package works with the new order save behavior and that all your packages associated with your Dev Hub org work with the new order save behavior, you can either enable the release update in your Dev Hub org or wait for it to be auto-enabled in Summer '22. We recommend supporting both the new and old order save behavior during the Release Update window.

The scenarios in the table take effect in Winter '21 and continue through the release update window in Summer '22.

**Table 2: Success and Failure Scenarios for Packages and Order Save Behavior**

Package Uses	Subscriber Org Has Enabled New Order Save Behavior	Subscriber Org Has Enabled Old Order Save Behavior
New Order Save Behavior	<p>SUCCEED</p> <p>Package upgrades and installations of major and minor package versions succeed. All package types succeed in this condition.</p> <p>FAIL</p> <p><b>Winter '21</b></p> <p>Patch upgrades for 1GP packages fail. If you enabled new order save behavior, create minor versions instead of patch versions of packages.</p> <p><b>Summer '21 and later</b></p> <p>Patch upgrades fail for 1GP packages that contain Salesforce Flow. If you enabled new order save behavior for packages with Salesforce Flow, create minor versions instead of patch versions of packages. All other patch upgrades for 1GP packages are successful.</p>	<p>FAIL</p> <p>If the subscriber org and the package are specifying different order save behaviors, package upgrades and installations are blocked.</p> <p>All package types fail in this condition.</p>

Package Uses	Subscriber Org Has Enabled New Order Save Behavior	Subscriber Org Has Enabled Old Order Save Behavior
Old Order Save Behavior	<b>FAIL</b> If the subscriber org and the package are specifying different order save behaviors, package upgrades and installations are blocked. All package types fail in this condition.	<b>SUCCEED</b> New package installations and major, minor, and patch upgrades succeed. All package types succeed in this condition.
Both New and Old Order Save Behavior	<b>SUCCEED</b> New package installations of major and minor package version upgrades succeed.	<b>SUCCEED</b> New package installations of major and minor package version upgrades succeed.

### Test Unmanaged and First-Generation Managed Packages

- From Setup, in the Quick Find box, enter *Release Updates*, and then select **Release Updates**. Locate the Enable New Order Save Behavior tile, and select **Enable Test Run**.
  - Test the impact of the new behavior when an order or order item is edited. Review any custom application logic such as validation rules, Apex triggers and classes, workflow rules, flows, and processes.
- We recommend supporting both the new and old order save behavior during the Release Update window.
- To indicate that your package is compatible with both new and old order save conditions, from Setup, in the Quick Find box, enter *Package*. Select the package that you tested and select **Upload**.
  - Locate the Package Requirements section and disable **New Order Save Behavior**.

When this setting is disabled and the release update is enabled, subscriber orgs using either the new or old order save behavior can install your package.

### Test Unlocked and Second-Generation Managed Packages

- After creating a scratch org, enable the Release Update in it. From Setup, in the Quick Find box, enter *Release Updates*, and then select **Release Updates**. Locate the Enable New Order Save Behavior tile, and select **Enable Test Run**.
- Test the impact of the new behavior when an order or order item is edited. Review any custom application logic such as validation rules, Apex triggers and classes, workflow rules, flows, and processes.

When you're ready to create a package version, specify the order save behavior in the definition file.

**Table 3: Order Save Behavior Options**

To Specify	Set Features in Scratch Org Definition File To
Old Order Save Behavior	<pre>{   "features": [],   "settings": {     "orderSettings": {       "enableOrders": true     }   } }</pre>

To Specify	Set Features in Scratch Org Definition File To
New Order Save Behavior	<pre>{   "features": ["OrderSaveLogicEnabled"],   "settings": {     "orderSettings": {       "enableOrders": true     }   } }</pre>
New and Old Order Save Behavior	<pre>{   "features": ["OrderSaveBehaviorBoth"],   "settings": {     "orderSettings": {       "enableOrders": true     }   } }</pre>

## Remove Metadata Components from Second-Generation Managed Packages

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

### Impact of Component Removal in Subscriber Orgs

During 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 remain in the subscriber org after package upgrade, and are marked as deprecated. When a package is upgraded in the subscriber org, the Setup Audit Trail logs which components were removed. Admins of a subscriber org can delete deprecated metadata.

You can remove the following metadata components from second-generation managed packages.

Metadata Component	Upon Package Upgrade, the Metadata Component is ...
Analytic Snapshot	Marked as deprecated
Apex Class (excluding global Apex classes)	Hard deleted
Apex Trigger	Hard deleted
Aura Definition Bundle	Marked as deprecated
Compact Layout	Marked as deprecated
Custom Application	Marked as deprecated
Custom Application Component	Marked as deprecated
Custom Field	Marked as deprecated

Metadata Component	Upon Package Upgrade, the Metadata Component is ...
Custom Labels	Marked as deprecated
Custom Object	Marked as deprecated
Custom Permission	Marked as deprecated
Custom Tab	Marked as deprecated
Dashboard	Marked as deprecated
Dashboard Folder	Marked as deprecated
Document	Marked as deprecated
Field Set	Marked as deprecated
FlexiPage	Marked as deprecated
Flow	Marked as deprecated
Layout	Marked as deprecated
List View	Marked as deprecated
Permission Set	Marked as deprecated
Profile	Marked as deprecated
Quick Action	Marked as deprecated
Record Type	Marked as deprecated
Remote Site Setting	Marked as deprecated
Report	Marked as deprecated
Report Folder	Marked as deprecated
Report Type	Marked as deprecated
Sharing Reason	Marked as deprecated
Static Resource	Marked as deprecated
Validation Rule	Marked as deprecated
Visualforce Component (excluding global components)	Hard deleted
Visualforce Page	Marked as deprecated
WebLink (Custom Button or Custom Link)	Marked as deprecated

**How to Remove Metadata Components**

To request access to this feature, log a case at [Salesforce Partner Community](#).

After your request is approved, remove the metadata component's source file from your Salesforce DX project, and create a package version. Test the new package version to ensure it's working properly without the removed metadata.

### Before You Remove Metadata Components from Second-Generation Managed Packages

To ensure you can successfully remove metadata components from a second-generation managed package, keep these details in mind.

- Request access to the feature, if you haven't already.
- Familiarize yourself with the list of metadata components that can be removed.
- Ensure that there aren't dependencies on the metadata you plan to remove. If any component in the package depends on or references the component you're removing, the package version creation operation fails. After you remove a component, you can't access any customizations that depend on the removed component.

### Remove Metadata Dependencies Within a Package

If there are dependencies to the metadata component you plan to remove, first resolve the dependency before removing the metadata component.

For example, before deleting a custom field that is referenced in a page layout, edit the page layout and remove the reference to the custom field. Then remove the custom field from your source file, and create a package version.

Some scenarios require a two-step approach to component removal. For example, let's say you plan to remove a Visualforce page that contains a Visualforce component and replace it with a Lightning page that contains a Lightning component. Removing both the Visualforce page and Visualforce component in a single upgrade could cause issues for your subscribers. These issues occur because Visualforce components are deleted, and Visualforce pages are deprecated during package upgrade.

To avoid issues for your subscribers in this example, remove the reference to the Visualforce component from the Visualforce page, create a package version, and push the upgrade. Then remove the Visualforce page from your package version, and push this upgrade to subscribers.

### Remove Dependencies Located in Other Packages

Before you remove a metadata component, first remove all references to the metadata, including references in other packages that depend on that metadata component. For example, if you're removing a public Apex class, ensure your other packages aren't referencing that class using the `@namespaceAccessible` annotation.

In this section, PackageA refers to the package in which you plan to remove a metadata component. And PackageB is any package that depends on the metadata you're removing from PackageA. If you have references to the metadata component or Apex class in PackageB, follow these steps:

1. Remove the reference to the metadata component from PackageB.
2. Create a version of PackageB.
3. Push the new version of PackageB to your subscribers.
4. Repeat these steps if any other packages include a reference to the metadata you plan to remove from PackageA.

After you've removed all references to the metadata component, remove the metadata component's source file from the Salesforce DX project of PackageA. Then create a version of PackageA. Before pushing this upgrade to subscribers, test the new package version to ensure it's working properly.

#### [What to Consider Before Removing Metadata Components](#)

In most cases, removing metadata components from a second-generation managed package marks the component as deprecated and doesn't hard delete the component from the subscriber org. This approach to component removal ensures that package upgrades don't disrupt a subscriber's org.

## What to Consider Before Removing Metadata Components

In most cases, removing metadata components from a second-generation managed package marks the component as deprecated and doesn't hard delete the component from the subscriber org. This approach to component removal ensures that package upgrades don't disrupt a subscriber's org.

But there's a scenario where a deprecated component can lead to a package upgrade issue. This issue only pertains to deprecated components, and no action is needed for hard deleted components.

To see which components are deprecated and which are deleted, see [Remove Metadata Components from Second-Generation Managed Packages](#).

Here's an example scenario of how a deprecated component leads to a package upgrade issue.

1. Subscriber A installs version 1.0 of a managed package.
2. A package developer removes project\_\_c custom object, and creates package version 2.0.
3. Subscriber A upgrades from version 1.0 to version 2.0, and project\_\_c is now marked as deprecated in their org. Any integration with project\_\_c that the subscriber created continues to work.
4. The package developer continues to refine their app, and then releases several new versions.
5. During development of version 5.0, the package developer adds a component named project\_\_c to the package.
6. A new subscriber, Subscriber B, successfully installs version 5.0.
7. Subscriber A tries to upgrade to version 5.0, but the installation fails because the admin at Subscriber A never deleted project\_\_c from their org.
8. The package developer has two paths to unblock Subscriber A.
  - a. Ask Subscriber A to remove all references to project\_\_c, and then delete the component from their org.
  - b. Remove project\_\_c from the package and release a new package version.

To prevent this kind of API name collisions in your packages, here are some best practices.

### Communicate within Your Team and Company

Before you remove any metadata, assess the impact to the package and to any packages that depend on that package. If you remove metadata in one package, that action has the potential to break the functionality of a package that depends on the removed metadata. Communicate within your team and company so that other developers are aware of this change.

### Document Package Changes for Future Developers

If you internally document the major changes that your package undergoes, including the name of metadata components that were removed, you can help alert future package developers about previously used API names.

### Communicate Changes with Your Subscribers

Educate your customers about the potential impact from any components you remove. In the Release Notes for your upgraded package, list all components you've removed and notify customers of any necessary actions.

## Delete a Second-Generation Managed Package or Package Version

Use the `force:package:version:delete` and `force:package:delete` commands to delete packages and package versions that you no longer need.

To delete a package or package version, users need the Delete Second-Generation Packages user permission. Before you delete a package, first delete all associated package versions.

Package Type	Can I delete beta packages and package versions?	Can I delete released packages and package versions?
Second-Generation Managed Packages	Yes	No
Unlocked Packages	Yes	Yes

**Considerations for Deleting a Package or Package Version**

- Deletion is permanent.
- Attempts to install a deleted package version will fail.
- Before deleting, ensure that the package or package version isn't referenced as a dependency.

**Examples:**

```
$ sfdx force:package:delete -p "Your Package Alias"
```

```
$ sfdx force:package:delete -p 0Ho...
```

```
$ sfdx force:package:version:delete -p "Your Package Version Alias"
```

```
$ sfdx force:package:version:delete -p 04t...
```

These CLI commands can't be used with first-generation managed packages or package versions. To delete a first-generation managed package, see [View Package Details](#) in the *ISVforce Guide*.

**Frequently Used Packaging Operations for Second-Generation Managed Packages**

For a complete list of Salesforce CLI packaging commands, see: [Salesforce Command Line Reference Guide](#).

Salesforce CLI command	What it Does
force:package:create	Creates a package. When you create a package, you specify its package type and name, among other things.
force:package:version:create	Creates a package version.
force:package:install	Installs a package version in a scratch, sandbox, or production org.
force:package:uninstall	Removes a package that has been installed in an org. This process deletes the metadata and data associated with the package.
force:package:version:promote	Changes the state of the package version from beta to the managed-released state.
force:org:create	Creates a scratch org.
force:org:open	Opens an org in the browser.

## Transfer a Second-Generation Managed Package to a Different Dev Hub

You can transfer the ownership of a second-generation managed package from one Dev Hub org to another. These transfers can occur either internally between two Dev Hub orgs your company owns, or you can transfer a package externally to another Salesforce Partner or ISV. This change provides a way to sell a second-generation managed package to a different company.



**Note:** This package transfer feature is available only to unlocked packages and second-generation managed packages. Dev Hub orgs aren't used with first-generation managed packages or unmanaged packages, so this feature doesn't apply to those package types.

### Request a Package Transfer to a Different Dev Hub

Start by logging a case in the Salesforce Partner Community, and provide the:

- Dev Hub org ID for the source org.
- Subscriber package ID of the package you're transferring. This ID starts with 033.

To verify the 033 ID of your package, run the `force:package:list` command with the `--verbose` flag on the source Dev Hub org.

- Dev Hub org ID for the destination org.
- Namespace of the package being transferred.
- Details about whether this package transfer is internal or external.

An external transfer occurs when you transfer a package to a Salesforce Partner or ISV who doesn't work at your company.

If you're transferring more than one package, file a separate case for each package.

After your case has been reviewed and approved, someone from Salesforce Partner Support will contact you to arrange a time to initiate the package transfer.



**Note:** For security reasons, package transfers between a Dev Hub located in Government Cloud and a Dev Hub located outside Government Cloud aren't permitted.

### Package Transfers to External Customers

If you're transferring a package to another Salesforce Partner or ISV, provide:

- The source code and config settings needed to properly set up their Salesforce DX environment.  
All config settings needed to properly set up the `sfdx-project.json` file, and a complete list of features and settings that must be specified in their scratch org definition file.
- The login credentials to the namespace org. This information is required to link the package namespace to their Dev Hub org.

### Prepare to Transfer Your Package

Here's how you can help ensure a smooth package transfer.

- Keep the namespace linked to the source Dev Hub. Before the package transfer, the namespace must be linked to both the source and destination Dev Hub orgs.
- Before the package transfer process is initiated, ensure all push upgrades or package version creation processes have completed.
- Delete package versions that are no longer needed.

- If specified, clear the package's Error Notification User using the `sfdx force:package:update` command. If you're transferring the package to a Dev Hub org that you own, you can set the Error Notification User to a user in the destination Dev Hub after the package transfer is complete.

## During the Package Transfer Process

All push upgrades or package version creation processes must be complete before the package transfer process is initiated. Salesforce Partner Support will alert you about the date the package transfer will occur.

## After the Package Transfer Is Complete

Run `sfdx force:package:list` and verify that the package is no longer associated with your Dev Hub.

If the transferred package is still visible in your CLI output, and the recipient of the package transfer indicates the package transfer succeeded, log a case with Salesforce Partner Support to remove the association of the package with your Dev Hub org.

Next, unpublish your existing AppExchange listing for this package.

## Impact of Package Transfers on Package IDs

ID Type	ID starts with	After package transfer is complete
Subscriber Package ID	033	This ID remains the same. ...
Subscriber Package Version ID	04t	This ID remains the same.
Package ID	0Ho	The transferred package receives a new and unique package ID.

## Update Your Package Project File

Before you create new packages or package versions on your Dev Hub, update your `sfdx-project.json` file and remove all references to the transferred package from the package directory and package alias sections.

If you have packages in your Dev Hub that depend on the package that you're transferring, update the package dependency section in your `sfdx-project.json` file to explicitly specify the 04t ID of the transferred package that you depend on.

For example, if you transferred pkgA to a different Dev Hub, and your `sfdx-project.json` file lists the package dependency like this.

```
"dependencies": [
 {
 "package": "pkgA"
 "versionNumber": "2.0.0.LATEST"
 }
]
```

Update the dependency to either specify the 04t ID of pkgA.

```
"dependencies": [
 {
 "package": "04tB000000UzH5IAK"
```

```
 }
]
```

Or specify the dependency using a package alias.

```
"dependencies": [
 {
 "package": "pkgA2.0.0-1"
 }
]
"packageAliases": {
 "pkgA2.0.0-1": "04tB000000UzH5IAK"
}
]
```

## What Package History Is Transferred?

Regardless of whether the package transfer occurred between two Dev Hub orgs you own, or the package was transferred externally to a Dev Hub you don't own, we transfer the package version history.

We transfer:

- Package name, namespace, type, and IDs. One exception is that the transferred package gets a new OHo ID.
- Package version info. This includes all the info that is typically displayed when you run the `force:package:version:list` or `force:package:version:report` command.

We don't transfer:

- Push upgrade history.
- Package version create requests.
- The username of the Dev Hub user who received Apex and other types of error notifications. This optional user is set using `--errornotificationusername`.

### [Take Ownership of a Second-Generation Managed Package Transferred from a Different Dev Hub](#)

You can take ownership of a second-generation managed package that is transferred from another Dev Hug org.

## Take Ownership of a Second-Generation Managed Package Transferred from a Different Dev Hub

You can take ownership of a second-generation managed package that is transferred from another Dev Hug org.

To initiate a package transfer from your Dev Hub org, see [Transfer a Second-Generation Managed Package to a Different Dev Hub](#).

 **Note:** For security reasons, package transfers between a Dev Hub located in Government Cloud and a Dev Hub located outside Government Cloud aren't permitted.

## Transfers from External Customers

If you're receiving the package from another Salesforce Partner or ISV, make sure they provide the source code for the package, and an outline for the config settings needed to properly set up your Salesforce DX environment.

Request all the configuration settings required to properly set up the `sfdx-project.json` file, and a complete list of features and settings that must be specified in your scratch org definition file.

Also ensure that the company who is transferring the ownership of the package provides the login credentials for the namespace org they used. This information is needed to link the package namespace to your Dev Hub org.

## Receive a Package Transfer

For internal transfers, skip this step. Only log the case described in [Transfer a Second-Generation Managed Package to a Different Dev Hub](#).

If you're receiving a package from a different Salesforce Partner or ISV, start by linking the namespace of the package you are receiving to your Dev Hub org. See [Link a Namespace to a Dev Hub Org](#) in the *Salesforce DX Developer Guide*.

Next, log a case in the Salesforce Partner Community, and provide the:

- Dev Hub org ID for the source org.
- Subscriber package ID of the package you're receiving. This ID begins with 033.
- Dev Hub org ID for the destination org.

## After the Package Transfer Is Complete

After the package transfer is complete, you'll be notified by Salesforce Partner Support.

To verify that the transferred package is associated with your Dev Hub, run `sfdx force:package:list`.

## Impact of Package Transfers on Package IDs

ID Type	ID starts with	After package transfer is complete
Subscriber Package ID	033	This ID remains the same. ...
Subscriber Package Version ID	04t	This ID remains the same.
Package ID	0Ho	The transferred package receives a new and unique package ID.

## Update Your Package Project File

Open and review the contents of the `sfdx-project.json` file that you received from the original package owner.

Open and review the contents of any scratch org definition files that you received from the original package owner. Definition files help in setting up your scratch orgs during development. Use the `--definitionfile` parameter to specify a definition file when you create a new package version.

If the package directories section lists additional packages that weren't transferred to you, remove those references from the `sfdx-project.json` file.

Next, review the package alias section of the `sfdx-project.json` file, and remove any references to package aliases that aren't associated with the package that was transferred.

Update the package alias of the transferred package to specify its 0Ho package ID.

## Before You Create a New Package Version

Similar to how you go about creating any new package versions, you must update the `sfdx-project.json` file, and update the version number and ancestor ID. We recommend you set the ancestor ID to HIGHEST.

To designate a Dev Hub user to receive email notifications for unhandled Apex exceptions, and install, upgrade, or uninstall failures associated with your package, run the `sfdx force:package:update` command, and use the `--errornotificationusername` parameter.

## What Package History Is Transferred?

Regardless of whether the package transfer occurred between two Dev Hub orgs you own, or the package was transferred externally to a Dev Hub you don't own, we transfer the package version history.

We transfer:

- Package name, namespace, type, and IDs. One exception is that the transferred package gets a new 0Ho ID.
- Package version info. This includes all the info that is typically displayed when you run the `force:package:version:list` or `force:package:version:report` command.

We don't transfer:

- Push upgrade history.
- Package version create requests.
- The username of the Dev Hub user who received Apex and other types of error notifications.

## Next Steps

You've verified that the package is associated with your Dev Hub, you've updated your `sfdx-project.json` file, and perhaps you've even created a new package version. Congrats! There's still a couple more items of business left to complete.

### 1. Register the transferred package with your License Management Org.

If this is an external transfer, log a case with Salesforce Partner Support and request provide both your LMO org ID, and the 033 package ID.

### 2. [Publish Your Package on AppExchange](#)

## Best Practices for Second-Generation Managed Packages

---

We suggest that you follow these best practices when working with second-generation managed packages.

- We recommend that you work with only one Dev Hub, and enable Dev Hub in your partner business org.
- The Dev Hub org against which you run the `force:package:create` command becomes the owner of the package. If the Dev Hub org associated with a package expires or is deleted, its packages no longer work.
- Include the `--tag` option when you use the `package:version:create` and `package:version:update` commands. This option helps you keep your version control system tags in sync with specific package versions.
- Create user-friendly aliases for packaging IDs, and include those aliases in your Salesforce DX project file and when running CLI packaging commands. See: [Package IDs and Aliases for Second-Generation Managed Packages](#).

## Manage Licenses for Managed Packages

Use the License Management App (LMA) to manage leads and licenses for your AppExchange solutions. By integrating the LMA into your sales and marketing processes, you can better engage with prospects, retain existing customers, and grow your ISV business. The LMA is a managed package that is installed in all partner business orgs (PBO) and includes custom objects that track details on packages, package versions, and licenses.

I need to...	Permissions	For details, see...
Configure the LMA	System Admin profile	<a href="#">Get Started with the License Management App</a> on page 426
Bill subscribers or monitor license expiration	Object Permissions: Read	<a href="#">Lead and License Records in the LMA</a> on page 429
Convert trial subscriptions into paying customers	Object Permissions: Edit	<a href="#">Modify a License Record</a> on page 430
Customize the LMA	Object Permissions: Edit	<a href="#">Extend the LMA</a> on page 430
Provision licenses to a subscriber	Object Permissions: Edit	<a href="#">Modify a License Record</a> on page 430
Support subscribers with technical issues	Various permissions (see <a href="#">Assign Permissions to the Subscriber Support Console</a> on page 428 for details)	<a href="#">Troubleshoot Subscriber Issues</a>

### Editions

Available in: both Salesforce Classic and Lightning Experience

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

 **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 AppExchange Publishing Console.

### [Configure Permissions for the License Management App](#)

Determine who needs access to the LMA, and set object permissions. Consider using a permission set to assign user permissions.

## Install the License Management App

The License Management App (LMA) is a managed package that is installed in all partner business orgs. The org that the LMA is installed in is called the License Management Org (LMO).

### USER PERMISSIONS

To install packages:

- Download AppExchange Packages

We strongly recommend that you use your partner business org (PBO) as your LMO. However, you can choose to install the LMA in another production org. Consider installing the LMA in an org that your company is already using to manage sales, billing, and marketing.

Commercial use of the LMA is prohibited in Developer and Partner Developer Edition orgs. Installing the LMA in a Developer Edition org is allowed only if you're building integrations with the LMA and need an environment only for development and testing purposes. You can install the LMA in Enterprise, Unlimited, or Performance Edition production orgs.

 **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 in to the [Salesforce Partner Community](#).

- Click the question icon  and then click **Log a Case for Help**.
- Select **Salesforce Partner Program Support**.



**Tip:** If you don't see the Salesforce Partner Program Support tile, use the org picker to select the org that's associated with your Salesforce partnership account.

c. For product, enter and select **Partner Programs & Benefits**.

d. For topic, enter and select **ISV Technology Request**.

e. Provide any other required details, and then click **Create Case**.

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 AppExchange Publishing Console.

A single LMO can manage multiple 1GP and 2GP packages, but a package can be associated with only one LMO.

1. Connect your packaging org (for 1GP) or your Dev Hub org (for 2GP) to the publishing console on the Partner Community.

a. Log in to the [Partner Community](#), and select the **Publishing** tab.

a. Select the Organizations tab, and click **Connect Org**.

a. Enter the login credentials, and then click **Submit**.

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 publishing console.

2. Select the **Packages** tab.

3. Locate the package you want to link to the LMO, and then click **Register Package**. To register each package you own, repeat this step.

4. Set the default behavior you want for your package license, and then click **Save**.

You can view which packages are linked on the Packages tab 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.

### USER PERMISSIONS

To manage licenses in the Partner Community:

- Manage Listings

## Configure Permissions for the License Management App

Determine who needs access to the LMA, and set object permissions. Consider using a permission set to assign user permissions.

Ensure that you have the LMA installed, and that you've linked your package to the LMO and AppExchange Publishing Console.

1. Set object permissions for the license, package, and package version custom objects.

Custom Object	Object Permissions
License	To view license records: Assign READ permissions

Custom Object	Object Permissions
	To modify license records: Assign READ and EDIT permissions
Package	To view package records: Assign READ permissions To modify package records: Assign READ and EDIT permissions
Package Version	To view package version records: Assign READ permissions We recommend leaving all package version records as read-only.

2. Set field-level security in user profiles or permission sets.

Custom Object	Field-Level Permissions
License	Make all fields read-only.
Package	Make all fields read-only.
Package Version	Make all fields read-only.

3. Add related lists to page layouts.

To enable...	Add the Licenses related list to the...
License managers to view the licenses associated with a particular lead	Lead page layout
LMA users to view the licenses associated with a particular account	Account page layout
LMA users to view the licenses associated with a particular contact	Contact page layout

#### [Assign Permissions to the Subscriber Support Console](#)

Create a permission set to provide users access to the Subscriber Support Console.

#### **Assign Permissions to the Subscriber Support Console**

Create a permission set to provide users access to the Subscriber Support Console.

 **Note:** If you've already assigned these permissions via a profile or another permission set, you can skip this task.

1. From Setup, in the Quick Find box, enter *Permission Sets*, and select **Permission Sets**.
2. Click **New** and enter your permission set information.
3. On the Permission Set Overview page, locate the Apps section, and select **Visualforce Page Access**.
  - a. Click **Edit**.
  - b. Add **sfLma.LoginToPartnerBT** and **sfLma.SubscriberSupport** to the list of Enabled Visualforce pages, and then click **Save**.
4. On the Permission Set Overview page, locate the System section, and select **System Permissions**. Click **Edit**.
  - a. Select **Log in to Subscriber Organization**, and click **Save**.
5. From Setup, in the Quick Find box, enter *Profiles*, and select **Profiles**.
  - a. Click **Edit**.
  - b. Under Custom App Settings, select **License Management App**.
  - c. Under Custom Tab Settings, locate the Subscribers tab and select **Default On**.
  - d. Click **Save**.

## Lead and License Records in the License Management App

Each time a customer installs your managed package, the License Management App (LMA) creates lead and license records.

The key objects in the LMA are Package, Lead, and License.

- Package—The LMA includes a Package custom object and a Package Version custom object. These objects display details about each 1GP or 2GP package and package version you've listed on AppExchange.
- Lead —The Lead standard object gives you details about who installed your package, such as the installer's name, company, and email address. Lead records created by the LMA are just like the ones you use elsewhere in Salesforce, except the lead source is Package Installation. You can manually convert leads into accounts and contacts. When you convert a lead, the license record links to the converted account or contact.
- License—The License custom object gives you control over how many users in the customer's org can access your package and for how long. Each license record links to a lead record and a package record.

To understand which actions you must take and which actions the LMA handles for you, review this table.

Action	Who Takes This Step
Your package is installed by a new subscriber.	Customer or prospect
A lead record is created with the customer's name, company, and email address.	LMA
A license record is created according to the values you specified when you registered the package.	LMA
The lead record is converted to account and contact records. (Optional)	You (ISV partner)
Account and contact records are associated with the license record.	LMA

## Modify a License Record

You can change a customer's access to your offering by modifying a license record using the License Management App (LMA). For example, you can increase or decrease the number of seats included with a license or change the expiration date.

1. In the LMA, locate the license.

2. Click **Modify License**.

When the LMA is installed, the Edit button doesn't appear on the license page layout, and the Modify License button is included instead. This setup is intentional. Only edit license records on the Modify License page.

3. Update the field values as needed.

Field	Description
Expiration	Enter the last day that the customer can access your package, or select <b>Does not expire</b> .
Seats	Enter the number of licensed seats, or select <b>Site License</b> to make your package available to all users in the customer's org. You can allocate up to 99,000,000 seats.
Status	Select a value from the dropdown. <ul style="list-style-type: none"><li>• <b>Trial</b>—Lets the customer try your offering for up to 90 days. After the trial license converts to an active license, it can't return to a trial state.</li><li>• <b>Active</b>—Lets the customer use your package according to the license agreement.</li><li>• <b>Suspended</b>—Prohibits the customer from accessing your offering.</li></ul> <p> <b>Note:</b> When your offering is uninstalled, its status is set to Uninstalled, and the license can't be edited.</p>

4. Click **Save**.

## Refresh Licenses for a Managed Package

To sync all license records for a package across all subscriber installations, you refresh the license. Refreshing the license can also resolve discrepancies between the number of licenses in a subscriber's org and the number displayed in the License Management App (LMA). Refreshing is required when you move the LMA to a different org.

-  **Note:** For each package, you can refresh licenses only one time per week.

1. From the LMA, select the **Packages** tab.
2. Open the package record.
3. Click **Refresh Licenses**. In Lightning Experience, Refresh Licenses is located in the dropdown menu.

## Extending the License Management App

The License Management App (LMA) is a managed package that you can customize and extend. In addition to using the LMA to manage leads and licenses, many partners also integrate it into their existing business processes.

The LMA includes these custom objects:

- [License](#) on page 432
- [Package](#) on page 431
- [Package Version](#) on page 431

You can add custom fields to the objects as long as you don't mark your custom fields as required.

#### [Package and Package Version Object Fields](#)

The License Management App (LMA) includes a Package custom object and a Package Version custom object. These objects display details about each 1GP or 2GP package and package version you've listed on AppExchange.

##### [License Object Fields](#)

Use the License custom object to set limits on how many users in the subscriber's org can use your app and for how long.

##### [Adding Custom Automation to License Management App Objects](#)

Here are some examples of how you can use the License Management App (LMA) to grow your business and retain customers.

## Package and Package Version Object Fields

The License Management App (LMA) includes a Package custom object and a Package Version custom object. These objects display details about each 1GP or 2GP package and package version you've listed on AppExchange.

To view details about a package record, from the LMA, select the **Packages** tab, and then select the package name. You can view package versions in the Package Version related list.



**Note:** The LMA creates the package records, which contain critical information for tracking your licenses and packages. Treat these fields as read-only and ensure that your object permissions protect package records.

Package Custom Object Fields	Description
Developer Name	The name of the org that owns the package. For 1GP, the org name is the packaging org. For 2GP, it's the Dev Hub org.
Developer Org ID	The 18-character ID of the org that owns the package. For 1GP, the org ID is the packaging org ID. For 2GP, it's the Dev Hub org ID.
Last License Refresh	The date when the License Refresh tool was last run.
Latest Version	The most recent package version you've released.
Lead Manager	The owner of the lead records that the LMA creates when a customer installs your package.
Next Available Refresh	The date when the License Refresh tool can be run again.
Owner	The LMA owns all package records.
Package ID	The 18-character ID that identifies the package. This ID starts with 033.
Package Name	The name you specified when you created the package.

Package Version Object Fields	Description
Package	The package name and links to the package record's detail page.
Package Version Name	The name you specified when you created the package version.

Package Version Object Fields	Description
Release Date	The date you created this package version.
Version Number	The version number in major.minor.patch format. For example, 3.1.0.
Version ID	The 18-character ID of this package version.

## License Object Fields

Use the License custom object to set limits on how many users in the subscriber's org can use your app and for how long.

The License Management App (LMA) creates a license record every time your package is installed in an org. For example, if a subscriber installs two of your 1GP packages and three of your 2GP packages, you have five license records for that subscriber in your LMA. If you deliver a 2GP app that is composed of multiple packages, a unique license record is created for each package in the app. You can allocate up to 99,000,000 seats per subscriber license.

To view details about a license record, select the **Licenses** tab in the LMA, and then select and open the license record.

License records are automatically created and contain critical information for tracking licenses. Do not directly edit the license record. Instead, use the [Modify License](#) on page 430 tool to change the expiration date, license status, and the number of licensed seats.

License Custom Object Fields	Description
Account	A lookup field to the account record for a converted lead.
Contact	A lookup field to the contact record for a converted lead.
Created By	License records are always created by the LMA.
Expiration Date	Displays the expiration date or <code>Does not expire</code> (default).
Install Date	The date the subscriber installed this package version.
Instance	The Salesforce instance where the subscriber's org resides.
Lead	The lead record that the LMA created when the package was installed. A lead represents the user who owns the license.  If you convert the lead into an opportunity, the lead name is retained but the lead record no longer exists.
License Name	An auto-generated number that represents an instance of a license. License names are in the format of L-00001, and each new license is incremented by one.
Licensed Seats	Displays the number of licenses or <code>Site License</code> (default).
License Status	The type of license: Active, Suspended, Trial, or Uninstalled.
License Type	This is a legacy field and can be ignored.
Org Edition	The edition of the subscriber's org.
Org Expiration Date	Applies only if the subscriber installs your package in a trial org. Indicates the date when the trial org expires. It isn't related to the package license expiration.
Org Status	The status of the subscriber's org: Active, Free, or Trial.

License Custom Object Fields	Description
Owner	The LMA owns all license records. Don't edit this field.
Package Version	A lookup field that links to the package version associated with this license.
Package Version Number	The version number in major.minor.patch format. For example, 3.1.0.
Sandbox	Indicates whether the license is for a package installed in a sandbox org.
Subscriber Org ID	The 15-character ID representing the subscriber's org.
Used Licenses	<p>Displays the number of users who have a license to the package.</p> <p>This field is blank if:</p> <ul style="list-style-type: none"> <li>• A customer uninstalled the package.</li> <li>• Licensed Seats is set to Site License.</li> </ul>

## Adding Custom Automation to License Management App Objects

Here are some examples of how you can use the License Management App (LMA) to grow your business and retain customers.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

### Alert Sales Reps Before a License Expires

If you're managing licenses for several packages, it can be difficult to track the various expirations. If a license expires accidentally, you could even lose a customer. To help your customers with renewals, set up a workflow rule to email a sales rep on your team before the license expires.

To automatically email the sales rep, follow these high-level steps.

1. Create an email template for the notification.
2. Create a workflow rule with a filter that specifies enough time before the expiration date to discuss renewal options.
3. Associate the workflow rule with a workflow alert that sends an email to the appropriate team member or sales rep.

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

### 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.

1. To remove the association between the LMA and the org where it's currently installed, log a case with the Partner Community.
  - a. Log in to the [Salesforce Partner Community](#).
  - b. Click the question icon (?) and then click **Log a Case for Help**.
  - c. Select **Product or Technical Support**.
  - d. For product, enter and select **Platform**.
  - e. For topic, enter and select **AppExchange & Managed Packages**.
  - f. Provide any other required details, and then click **Create Case**.
2. [Install the LMA in the new org](#) on page 426.
3. [Associate your packages with the new org](#) on page 427.
4. [Refresh licenses for your packages](#) on page 430.

### 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.

### EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

#### [Leads and Licenses Aren't Being Created in the License Management App](#)

When a customer installs your package, leads and license records are created. If these records aren't being created, review these configurations in the License Management Org (LMO). If you resolve your issue using one of these recommendations, your missing licenses appear in the LMA within a few days.

#### [Proxy User Has Deactivated Message in the LMA](#)

If you're editing a license and see a "proxy user has deactivated" message, check whether the subscriber org is locked, deleted, or disabled.

## Leads and Licenses Aren't Being Created in the License Management App

When a customer installs your package, leads and license records are created. If these records aren't being created, review these configurations in the License Management Org (LMO). If you resolve your issue using one of these recommendations, your missing licenses appear in the LMA within a few days.

### Did the customer complete the package installation?

When a customer clicks **Get it Now** on your AppExchange listing, Salesforce counts this selection as an installation. However, the customer can cancel the installation before it's completed, or the installation could have failed. If the installation doesn't finish, a license isn't created.

### Is State and Country picklist validation enabled?

To avoid state and country picklist-related lead failures, you have two options. Use the standard picklist integration values, or add duplicate states and countries to your picklists.

### Standard picklist integration values

To implement this option, use the Salesforce standard state and country picklists in your org, and leave the integration values as-is. We recommend this option for most partners.

With this option, AppExchange leads propagate to your org with full state and country names, and the names match integration values in the standard picklists.

#### Add duplicate states and countries to your picklists.

Implement this option if you have a requirement to use the two-letter state or country abbreviations in your org. For example, you display abbreviations in the user interface or use them to integrate with other systems. Add duplicate states and countries to your picklists with different integration values. Set one value to the two-letter state or country abbreviation. Set the other value to the full state or country name. Make only the two-letter abbreviation picklist entries visible.

With this option, AppExchange leads propagate to your org with full state and country names, which match the full name integration values in your org. You also have two-letter integration values to use as needed.

#### Does the lead or license object have a trigger?

Don't use `before_create` or `before_update` triggers on leads and licenses. Instead, use `after_` triggers, or remove all triggers. If a trigger fails, it can block license creation.

#### Does the lead or license record have a required custom field?

If yes, remove the requirement. The LMA doesn't populate a required custom field, so it can prevent licenses or leads from being created.

#### Is the lead manager a valid, active user?

If not, the LMA can't create leads and licenses.

#### Does the lead or license record have a validation rule?

Validation rules often block the creation of LMA lead or license records because the required field isn't there.

#### Does the lead or license have a workflow rule?

Workflow rules sometimes prevent leads and licenses from being created. Remove the workflow rule.

#### Was the lead converted to an account?

When leads are converted to accounts, they're no longer leads.

#### Are you using standard duplicate rules for leads?

When a customer installs your package, the LMA checks for existing leads and contacts. If an existing contact matches the customer who installed your package, a lead record isn't created. To complete these checks, the LMA applies [standard lead duplicate rules](#) and [matching rules](#). If you prefer to have the LMA associate every license with a lead regardless of whether there's an existing contact match, [customize the standard duplicate rule for leads](#) and remove the matching rule for contacts.

## Proxy User Has Deactivated Message in the LMA

If you're editing a license and see a "proxy user has deactivated" message, check whether the subscriber org is locked, deleted, or disabled.

- If the org has been deleted, delete the corresponding license record.
- If the org is locked or if the package has been uninstalled, license records can't be updated.

## Best Practices for the License Management App

Follow these best practices when you use the License Management App (LMA).

- To take advantage of entitlements that are unique to AppExchange partners, use your partner business org as your License Management Org.
- Create a list view filter for leads created by installed packages. The filter helps your team separate subscriber-based leads from leads coming from other sources.

- Use the API to find licensed users. The `isCurrentUserLicensed` method determines if a user has a license to a managed package. For more information, see the [Apex Reference Guide](#).
- Treat the LMA custom objects as read-only. Use the Modify License page to edit licenses. Don't attempt to directly or programmatically edit license records.
- The LMA automatically creates package, package version, and license records. Customizations, such as adding required custom fields or creating workflow rules, triggers, or validation rules that require custom fields, can prevent the LMA from working properly.

## Troubleshoot Subscriber Issues

Use the Subscriber Support Console to access information about your subscribers. Subscribers can also grant you login access to troubleshoot issues directly within your app. After you're granted access, you can log in to the subscriber's org and view their configuration and data to troubleshoot and resolve issues.

To access the Subscriber Overview page, click the organization's name from the **Subscribers** tab in the LMA.

-  **Note:** This feature is available to eligible Salesforce partners. For more information on the Partner Program, including eligibility requirements, see [www.salesforce.com/partners](http://www.salesforce.com/partners).

### [Request Login Access from Subscribers](#)

To log in to a subscriber org, first request login access from the subscriber.

#### [Log In to Subscriber Orgs](#)

After your subscriber has granted you login access, you can log in to the subscriber org to troubleshoot the issue.

#### [Debug Subscriber Orgs](#)

After logging in to a subscriber's org, you can view logs, obfuscated code in your package, and initiate ISV Customer Debugger sessions.

## Request Login Access from Subscribers

To log in to a subscriber org, first request login access from the subscriber.

Ask the subscriber to enable either **Grant Account Login Access** or **Grant Login Access**. If they don't see your company listed, one of the following applies.

- A system admin disabled the ability for non-admins to grant access.
- The user doesn't have a license for the package.
- The package is licensed to the entire org. In this scenario, only an admin with the Manage Users permission can grant access.
- The org setting **Administrators Can Log in as Any User** is enabled.

-  **Note:** When the org setting **Administrators Can Log in as Any User** is disabled, login access is granted for a limited amount of time, and the subscriber can revoke access at any time.

Any changes you make while logged in as a subscriber are logged in the subscriber org's audit trail.

## Log In to Subscriber Orgs

After your subscriber has granted you login access, you can log in to the subscriber org to troubleshoot the issue.

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

### USER PERMISSIONS

To log in to subscriber orgs:

- Log in to Subscriber Org



**Note:** You can only log in to orgs with a Salesforce Platform or full Salesforce license. You can't log in to subscriber orgs on Government Cloud instances.

## Multi-Factor Authentication Required to Log In to a Subscriber Org

Starting in Spring '22, multi-factor authentication (MFA) is required when logging into the License Management Org (LMO). MFA is required only for LMO users who require access to the Subscriber Support Console. This requirement provides subscribers an extra layer of security by verifying the identity of the user accessing their org. You also have more control over which users log in to a subscriber org.

Determine which users require access to the Subscriber Support Console, and then [set up multi-factor authentication \(MFA\)](#) for those users.

## Log In to a Subscriber Org

After you've logged in to the LMO using multi-factor authentication (MFA), and your subscriber has granted you login access, you're ready to log in.

1. In the License Management App (LMA), click the **Subscribers** tab.
2. To find a subscriber org, enter a subscriber name or org ID in the search box, and click **Search**.
3. Click the name of the subscriber org.
4. On the Org Details page, click **Login** next to a user's name. You have the same permissions as the user you logged in as.
5. When you're finished troubleshooting, log out of the subscriber org.



**Note:** Some subscribers require MFA in addition to the MFA required for the LMO. Ask your subscriber if their org requires MFA to log in. If so, your login attempt sends an MFA notification to your subscriber, and your login is blocked until your subscriber responds to the notification. To ensure that your subscriber is available to respond to the MFA notification, consider coordinating a specific login time.

## Best Practices for Logging In

- Create an audit trial that indicates when and why a subscriber org login has occurred. You can create an audit trail by logging a case in your LMO before each subscriber org login.
- When you access a subscriber org, you're logged out of your LMO. You can set up a My Domain to not be automatically logged out of your LMO when you log in to a subscriber org. To set up a My Domain subdomain, from Setup, in the Quick Find box, enter *My Domain*, then select **My Domain**.
- Allow only trusted support and engineering personnel to log in to a subscriber's org. Because this feature can include full read/write access to customer data and configurations, it's vital to your reputation to preserve their security.
- Control who has login access by giving the Log in to Subscriber Org user permission to specific support personnel via a profile or permission set. See [Assign Permissions to the Subscriber Org Console](#) on page 428.

## Debug Subscriber Orgs

After logging in to a subscriber's org, you can view logs, obfuscated code in your package, and initiate ISV Customer Debugger sessions.

## Troubleshoot with Debug Logs

You can debug your code by generating Apex debug logs that contain the output from your managed package. Using this log information, you can troubleshoot issues that are specific to that subscriber.

1. If the user has access, set up a debug log: From Setup, in the Quick Find box, enter *Debug Logs*, and then select **Debug Logs**.
2. Launch the Developer Console.
3. Perform the operation, and view the debug log with your output.

Subscribers are unable to see the logs you set up or generate because they contain your unobfuscated Apex code.

You can also view and edit data contained in protected custom settings from your managed packages when logged in as a user.

## Troubleshoot with the ISV Debugger

Each License Management Org can use one free ISV Customer Debugger session at a time. The ISV Customer Debugger is part of the [Salesforce Extensions for Visual Studio Code](#). You can use the ISV Customer Debugger only in sandbox orgs, so you can initiate debugging sessions only from a customer's sandbox.

For details, see the [ISV Customer Debugger](#) documentation.

# Manage Features in Second-Generation Managed Packages

---

Take the License Management App (LMA) a step further by extending it with the Feature Management App (FMA).

Here at Salesforce, we sometimes run pilot programs, like the one we ran when we introduced Feature Management. Sometimes we dark-launch features to see how they work in production before sharing them with you. Sometimes we make features available to select orgs for limited-time trials. And sometimes we want to track activation metrics for those features.

With feature parameters, we're extending this functionality to you. Install the FMA in your License Management Org (LMO). The FMA extends the License Management App, and like the LMA, it's a managed package.

### [Feature Parameter Metadata Types and Custom Objects](#)

Feature parameters are represented as Metadata API types in your package metadata, as records of custom objects in your LMO, and as hidden records in your subscriber's org.

### [Set Up Feature Parameters](#)

Set up the Feature Management App in your License Management Org, define feature parameters, and add them to your package.

### [Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features](#)

Feature parameters with a data flow direction value of `LMO to Subscriber` are writable at your end and read-only in your subscriber's org. These feature parameters serve as permissions or limits. Use LMO-to-subscriber feature parameters to enable or disable new features or to control how many of a given resource your subscriber can use. Or, enable features for a limited trial period.

Assign values to LMO-to-subscriber feature parameters by updating junction object records in your LMO, and then check those values in your code.

### [Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters](#)

Use subscriber-to-LMO feature parameters to track feature activation in your subscriber's org. Parameter values are assigned on the subscriber's end and then sent to your LMO. To collect the values, update the feature parameters in your subscriber's org using Apex code. Check with your legal team before obtaining activation metrics from your customers. Use activation metrics to collect only aggregated data regarding feature activation.

### [Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs](#)

Occasionally, you want to include custom permissions or custom objects in a package but not show them to your subscribers. For example, if you're piloting a feature for a few select orgs, and want to hide custom permissions and custom objects related to the pilot feature.

### Best Practices for Feature Management

Here are some best practices when working with feature parameters.

### Considerations for Feature Management

Keep these considerations in mind when working with feature parameters.

## Feature Parameter Metadata Types and Custom Objects

Feature parameters are represented as Metadata API types in your package metadata, as records of custom objects in your LMO, and as hidden records in your subscriber's org.

## Feature Parameter Fields

Feature parameters are represented as Metadata API types and store boolean, integer, or date values.

The first time a subscriber installs your package, a `FeatureParameter__c` record is created in your LMO for each feature parameter. The feature parameter records include these fields:

- `FullName__c`
- `DataType__c` (Boolean, Integer, or Date)
- `DataFlowDirection__c`
- `Package__c`
- `IntroducedInPackageVersion__c`
- `Namespace_Prefix__c`

 **Note:** After a feature parameter is included and released in the package version, the data flow direction can't be changed.

## Lifecycle of a Feature Parameter

### Set Up the Feature Parameter

Start by defining your feature parameter in an XML file. [Create one XML file](#) for each feature parameter.

Depending on how you're using the feature parameter, you'll also write code that enables you to check access rights or collect usage information after the parameter is set up.

### Subscriber Installs Your Managed Package

When a subscriber installs or upgrades your package in their org, a `FeatureParameter__c` record for each feature parameter is created in the LMO. If these records were created during a previous installation or upgrade, this step is skipped.

During package installation, junction object records are created in both the subscriber org and your LMO. A junction object is a custom object with two master-detail relationships. In this case, the relationships are between `FeatureParameter__c` and `License__c` in the LMO. These records store the value of their associated feature parameter for the subscriber org.

### Utilize Your Feature Parameters

Use the junction objects to override the feature parameters' default values or to collect data. Depending on the value of each feature parameter's `DataFlowDirection__c` field, data flows to the subscriber org (from the LMO) or to the LMO (from the subscriber org). That data is stored in the junction object records.

## Set Up Feature Parameters

Set up the Feature Management App in your License Management Org, define feature parameters, and add them to your package.

### Install and Set Up the Feature Management App in Your License Management Org

Install the FMA in your LMO. Then add the Feature Parameters tab to your default view, and adjust your page layout for licenses to display related lists for your feature parameters.

#### Create Feature Parameters for Your Second-Generation Managed Package

To create a feature parameter for a 2GP managed package, create an individual XML file. Here are details on the file naming convention, folder structure, and the attributes you use when creating feature parameters.

## Install and Set Up the Feature Management App in Your License Management Org

Install the FMA in your LMO. Then add the Feature Parameters tab to your default view, and adjust your page layout for licenses to display related lists for your feature parameters.

1. To request access to the FMA, log a support case in the [Salesforce Partner Community](#). For product, specify **Partner Programs & Benefits**. For topic, specify **ISV Technology Request**. The FMA extends the License Management App, so be sure to install the LMA before requesting access to the FMA.
2. To install the FMA, follow the instructions in your welcome email.
3. Add the Feature Parameters tab to your default view. For details, see [Customize My Tabs](#) in Salesforce Help.
4. Update your page layout for licenses.
  - a. Navigate to a license record's detail page.
  - b. Click **Edit Layout**.
  - c. In the Related Lists section of the License Page Layout Editor, add these lists.
    - Feature Parameter Booleans
    - Feature Parameter Dates
    - Feature Parameter Integers
  - d. For each related list, add these columns.
    - Data Flow Direction
    - Feature Parameter Name
    - Full Name
    - Master Label
    - Value

## Create Feature Parameters for Your Second-Generation Managed Package

To create a feature parameter for a 2GP managed package, create an individual XML file. Here are details on the file naming convention, folder structure, and the attributes you use when creating feature parameters.



**Note:** Feature parameters for managed 1GP packages are created in the packaging org's UI, see [Create Feature Parameters in Your Packaging Org](#) in the *ISVforce Guide* for details.

### Folder Structure

Feature parameters are stored as files in your Salesforce DX project folder.

Under the root `force-app` folder, create a folder and name it `featureParameters`. Store your feature parameter files in the `featureParameters` folder. Each feature parameter you create must have its own separate file.



**Note:** It's not possible to create feature parameters using a scratch org's user interface.

### File Naming Convention

The naming format for feature parameter files is `<name>.featureParameter<type>-meta.xml`.

The name is the API name of the feature parameter.

The type is the feature parameter type. Feature parameters can be booleans, integers, or dates.

Type	File Name Format
Boolean	.featureParameterBoolean-meta.xml
Date	.featureParameterDate-meta.xml
Integer	.featureParameterInteger-meta.xml

### Feature Parameter Attributes

Feature parameters include these three fields.

Field Name	Description
dataflowDirection	Indicates which direction this parameter is transferring data. Each feature parameter value gets transferred in one of two directions: <ul style="list-style-type: none"><li>• From your LMO to a subscriber org (LmoToSubscriber)</li><li>• From a subscriber org to your LMO (SubscriberToLmo)</li></ul>
masterLabel	The label of the feature parameter. This label displays in the app.
value	The value of the feature parameter. Booleans, integers, and dates are all valid values. Integer values can't exceed nine digits.



**Note:** After a feature parameter is included and released in the package version, the data flow direction can't be changed.

### Examples of Feature Parameter file

#### AdvancedPricingEnabled.featureParameterBoolean-meta.xml

```
<FeatureParameterBoolean xmlns="http://soap.sforce.com/2006/04/metadata">
 <dataflowDirection>SubscriberToLmo</dataflowDirection>
 <masterLabel>Advanced Pricing Enabled</masterLabel>
 <value>true</value>
</FeatureParameterBoolean>
```

#### NumberofLedgers.featureParameterInteger-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<FeatureParameterInteger xmlns="http://soap.sforce.com/2006/04/metadata">
 <dataflowDirection>SubscriberToLmo</dataflowDirection>
 <masterLabel>Number of Ledgers</masterLabel>
 <value>7</value>
</FeatureParameterInteger>
```

**ProjectActivationDate.featureParameterDate-meta.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<FeatureParameterDate xmlns="http://soap.sforce.com/2006/04/metadata">
 <dataflowDirection>LmoToSubscriber</dataflowDirection>
 <masterLabel>Date of Activation of the Project</masterLabel>
 <value>2020-01-25</value>
</FeatureParameterDate>
```

## Use LMO-to-Subscriber Feature Parameters to Enable and Disable Features

Feature parameters with a data flow direction value of `LMO to Subscriber` are writable at your end and read-only in your subscriber's org. These feature parameters serve as permissions or limits. Use LMO-to-subscriber feature parameters to enable or disable new features or to control how many of a given resource your subscriber can use. Or, enable features for a limited trial period. Assign values to LMO-to-subscriber feature parameters by updating junction object records in your LMO, and then check those values in your code.

### [Assign Override Values in Your LMO](#)

To override the default value of a feature parameter in a subscriber's org, update the appropriate junction object record in your LMO.

### [Check LMO-to-Subscriber Values in Your Code](#)

You can reference feature parameters in your code, just like you'd reference any other custom object.

## Assign Override Values in Your LMO

To override the default value of a feature parameter in a subscriber's org, update the appropriate junction object record in your LMO.

1. Open the license record for a subscriber's installation of your package.
2. In the related list for Feature Parameter Booleans, Feature Parameter Integers, or Feature Parameter Dates, select the feature parameter whose value you want to update.
3. Click **Edit**.
4. Set a value.
5. Click **Save**.

## Check LMO-to-Subscriber Values in Your Code

You can reference feature parameters in your code, just like you'd reference any other custom object.

Use these Apex methods with LMO-to-subscriber feature parameters to check values in your subscriber's org.

- `System.FeatureManagement.checkPackageBooleanValue('YourBooleanFeatureParameter');`
- `System.FeatureManagement.checkPackageDateValue('YourDateFeatureParameter');`
- `System.FeatureManagement.checkPackageIntegerValue('YourIntegerFeatureParameter');`

## Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters

Use subscriber-to-LMO feature parameters to track feature activation in your subscriber's org. Parameter values are assigned on the subscriber's end and then sent to your LMO. To collect the values, update the feature parameters in your subscriber's org using Apex code. Check with your legal team before obtaining activation metrics from your customers. Use activation metrics to collect only aggregated data regarding feature activation.

- `System.FeatureManagement.setPackageBooleanValue('YourBooleanFeatureParameter', booleanValue);`
- `System.FeatureManagement.setPackageDateValue('YourDateFeatureParameter', datetimeValue);`
- `System.FeatureManagement.setPackageIntegerValue('YourIntegerFeatureParameter', integerValue);`

 **Warning:** The `value__c` field on subscriber-to-LMO feature parameters is editable in your LMO. But don't change it. The changes don't propagate to your subscriber's org, so your values will be out of sync.

## Hide Custom Objects and Custom Permissions in Your Subscribers' Orgs

Occasionally, you want to include custom permissions or custom objects in a package but not show them to your subscribers. For example, if you're piloting a feature for a few select orgs, and want to hide custom permissions and custom objects related to the pilot feature.

 **Note:** Check with your company's legal team before releasing hidden functionality.

To hide custom objects when creating your package, set the value of their `Visibility` field to `Protected`.

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');`

## 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.

- Limit the number of feature parameters in your package. Each package can include up to 25 feature parameters. To request more than 25 feature parameters, log a case in [Salesforce Partner Community](#).
- 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.

## Gaps Between First-Generation and Second-Generation Managed Packaging

---

The following functionality is supported in first-generation managed packaging, and not yet supported in second-generation managed packaging. We're working to address these feature gaps.

- Package versions can't be deprecated.
- [Apex VersionProvider](#) isn't supported.
- A default language for labels in packages can't be specified.

See the [Metadata Coverage Report](#), for the latest information on supported metadata types.

# CHAPTER 14 Partner Licensing Platform (Developer Preview)

## In this chapter ...

- [Enable the Partner Licensing Platform \(Developer Preview\)](#)
- [Quick Start: Get Started with the Partner Licensing Platform \(Developer Preview\)](#)
- [Partner Licensing Platform Components and Concepts \(Developer Preview\)](#)
- [Design Your Package Licensing Structure \(Developer Preview\)](#)
- [Implement Your Package Licensing \(Developer Preview\)](#)
- [Test Your Licensing Structure \(Developer Preview\)](#)
- [Manage Migrations to the Partner Licensing Platform \(Developer Preview\)](#)

The Partner Licensing Platform (PLP) is a robust new platform that revolutionizes how Salesforce Partners and ISVs price, license, and distribute their applications. Product owners, architects, and developers who build apps for AppExchange can transform their business and products with the power of PLP.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Our [forward-looking statement](#) applies to partner business org provisioning. Make your purchasing decisions based on currently available technology.

Now for the first time, you can develop and distribute multiple license types for your package that entitle specific features, without breaking up your package structure. With this functionality you can increase and diversify your revenue stream by selling multiple tiers, versions, or supplements for your products via these license types.

You can also create licenses that can be assigned only to specific types of users. For example, a license for internal Salesforce users with complete functionality, and a separate license for external Experience Cloud users with the same or limited functionality, for a lesser price. You can potentially recover revenue for products that were previously only contractually restricted.

This onboarding guide outlines how to sign up for the developer preview and design and implement your new licensing strategy. Starting now, you can get ready to take advantage of the incredible capabilities of the next generation of licensing for Salesforce Partners.

## What's Available in the Developer Preview?

The following features are available in the Summer '22 release:

- Custom foundation and supplement permission set licenses
- License expiration policies
- Licensed custom permissions
- User license restrictions

Partner business org provisioning of custom permission set license seats is an expected eventual capability of the Partner Licensing Platform.

For info and updates about the developer preview, join the [Partner Licensing Platform Developer Preview Partner Community group](#). For general news about PLP, join the [Partner Licensing Platform Partner Community group](#).

SEE ALSO:

[\*TrailheaDX '21 Presentation: Next Generation Licensing for ISV Partners\*](#)

[\*Salesforce Video: Partner Licensing Platform Demo\*](#)

# Enable the Partner Licensing Platform (Developer Preview)

Review requirements for the developer preview and set up necessary orgs before submitting a participation request.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

## Scope of the Developer Preview

The developer preview isn't available for your production development and packaging orgs. To participate in the developer preview, you must create **new** orgs for using and testing the new Partner Licensing Platform.

You're encouraged to use your existing package code to see how you would transform your package and product with the new licensing capabilities. You can also consider creating a new repository or branch in your package code source control for implementing the new functionality alongside your production package development.

Alternatively, we prepared a [demo application](#) that you can use to test out the new functionality with your new developer preview orgs.

## Requirements

The Partner Licensing Platform supports both 1GP and 2GP. To participate in the developer preview, you must have access to an [Environment Hub](#) (Env Hub) and a [Developer Hub](#) (Dev Hub). Both hubs are available to all AppExchange partners as part of your Partner Business Org (PBO).

Participants must be familiar with scratch org-based development using the Salesforce Developer Experience command-line interface ([SFDX CLI](#)). Scratch orgs are required as development and testing environments during the developer preview.

## Create Your New Orgs

You must create the following orgs for end-to-end development, packaging, and testing with the Partner Licensing Platform enabled:

- A new Partner Enterprise org with Dev Hub enabled
  - See [Create an Org from the Environment Hub](#) and create a Test/Demo org using the Partner Enterprise edition.
  - See [Enable Dev Hub Features in Your Org](#) to enable your Dev Hub.
  - See [Authorization](#) to authenticate the Salesforce CLI to your Dev Hub.
- A new Partner Developer org with a namespace prefix that starts with **plpdp\_**
  - See [Create an Org from the Environment Hub](#) and create a Development org using the Partner Developer edition.
  - See [Register a Namespace](#) and make sure that the namespace prefix you choose for your Partner Developer org starts with plpdp\_ (such as "plpdp\_mytest").
  - If testing with 1GP, use this Partner Developer org as your packaging org.
  - If testing with 2GP, [link a namespace to your Dev Hub](#) to allow creation of second-generation packages with your new namespace.

## Sign Up for the Developer Preview

After you create the required orgs, make note of their org IDs. Then, submit a participation request via the [Partner Licensing Platform Developer Preview](#) Partner Community group. After the request is reviewed and approved, Salesforce will enable the developer preview in the new orgs that you created.

Note that there is limited space and Salesforce may close the developer preview to new partners when the current cohort is full.

## Set Up the Development Environment

In the developer preview, you'll create your new custom permission set licenses in scratch orgs created from your new Dev Hub. After your request to join the developer preview is approved, the Partner Licensing Platform will be enabled for your Dev Hub. This will allow you to create scratch orgs with the `PartnerLicensingPlatform` feature enabled in the scratch org definition file. For more information about enabling features in scratch orgs, see [Build Your Own Scratch Org Definition File](#), or take a look at the demo application [scratch org definition](#).

After you implement your new licenses and settings, you're ready to create a new package with this metadata. For 1GP, deploy the metadata to the new packaging org and create a new package. For 2GP, add the metadata source to your package directory's source and create a new package using SFDX.

Finally, when you're ready to install and test your new package from the package subscriber's perspective, use a new scratch org created from your new Dev Hub. Packages with the new functionality can only be installed in scratch orgs that have the `PartnerLicensingPlatform` feature enabled in their scratch org definition files.

 **Note:** To use the Partner Licensing Platform, you must use API version 55.0 or later.

## Quick Start: Get Started with the Partner Licensing Platform (Developer Preview)

---

Get up and running with the Partner Licensing Platform with a demo application.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Our [forward-looking statement](#) applies to partner business org provisioning. Make your purchasing decisions based on currently available technology.

After your development environment is set up, you can use the [demo application](#) to try out the Partner Licensing Platform (PLP).

The demo app contains all of the code and metadata components needed to demonstrate how an AppExchange application's functionality can be gated using licensed custom permissions and custom permission set licenses. We go into detail about these new components in [Partner Licensing Platform Components and Concepts \(Developer Preview\)](#), but you can set up and use the demo app immediately.

Detailed instructions for setting up and using the demo application are located in the README file of the [demo application repository](#). Follow the steps outlined there to continue.

## Considerations During the Developer Preview

During the developer preview, custom permission set licenses can only be tested via an installed package, as a subscriber of your package. For testing in the developer preview, the subscriber org where the managed package is installed is provisioned a default count of 10

seats for each custom permission set license in the package. When the functionality is available, you'll be able to provision a number of seats for your custom permission set licenses to each of your subscribers from your partner business org.

## Using the Demo App

After you've finished setting up the demo app, you're ready to see how the Partner Licensing Platform works for a subscriber. First, log in to your subscriber org using the admin user and assign the Feature Access Demo permission set to the standard user. Then, log in to your subscriber org using the standard user and navigate to the Feature Access Visualforce page via the App Launcher in Lightning Experience.

The Feature Access page simulates access to package features that are gated behind licenses. Clicking the buttons allows you to see which features are accessible to the current user. Without the proper custom permission set licenses and permission sets assigned to the user, the user can't access the features on the page.

Now log in to your subscriber org as the admin user. Assign the **Feature\_A** custom permission set license to your standard user. Now, that user is entitled to use Feature A in that org.

While the standard user is **entitled** to Feature A via a license assignment, the user still can't access Feature A until the admin further **grants access** to the user to Feature A via the **FeatureA User** permission set.

This grant is only possible after the admin has assigned the **Feature\_A** custom permission set license to the standard user. The custom permission set license **entitles** the standard user to Feature A, and then the permission set **grants access** to the standard user for Feature A. This process of assigning licenses and permission sets results in a two-step access check for accessing a licensed feature.

Now, you'll see that the standard user can access Feature A, but not Feature B. To further illustrate how licenses and permission sets work together, try to grant access to the standard user to Feature B by assigning the **FeatureB User** permission set. This assignment isn't allowed because access to Feature B can only be **granted** via permission set to users who are first **entitled** to use Feature B via a license assignment. In this way, users are only able to **access** features that they're both **entitled** and **granted**.

You can explore combinations of custom permission set license and permission set assignments with the standard user and see how their access changes for gated features. Later in this guide, the concept of entitlements and granting access is explored in more detail.

In this demo app, we demonstrated how to gate access to your package features using custom permission set licenses and licensed custom permissions. You can also use this demo later as reference when you implement your package licensing.

## What's Next?

Now that you've used the demo app to try out PLP, it's time for you to take advantage of the power of the platform to transform your own business and products. You can develop and distribute multiple license types for your package that entitle specific features, without breaking up your package structure. With this functionality, you can increase and diversify your revenue stream by selling multiple tiers, versions, or supplements for your products via these license types. You can even create separate licenses that can be assigned only to specific types of users, such as internal Salesforce users and external Experience Cloud users, separately.

The following sections dive into the data model of the new Partner Licensing Platform, covering how the platform works and how this new metadata and permissioning lifecycle is structured. Then, we walk you through the end-to-end process for how to redesign your product licensing with these capabilities using an example product.

As you go through the journey, you're welcome to ask questions and collaborate on the [Partner Licensing Platform Developer Preview](#) Partner Community group. For general news about PLP, join the [Partner Licensing Platform](#) Partner Community group.

# Partner Licensing Platform Components and Concepts (Developer Preview)

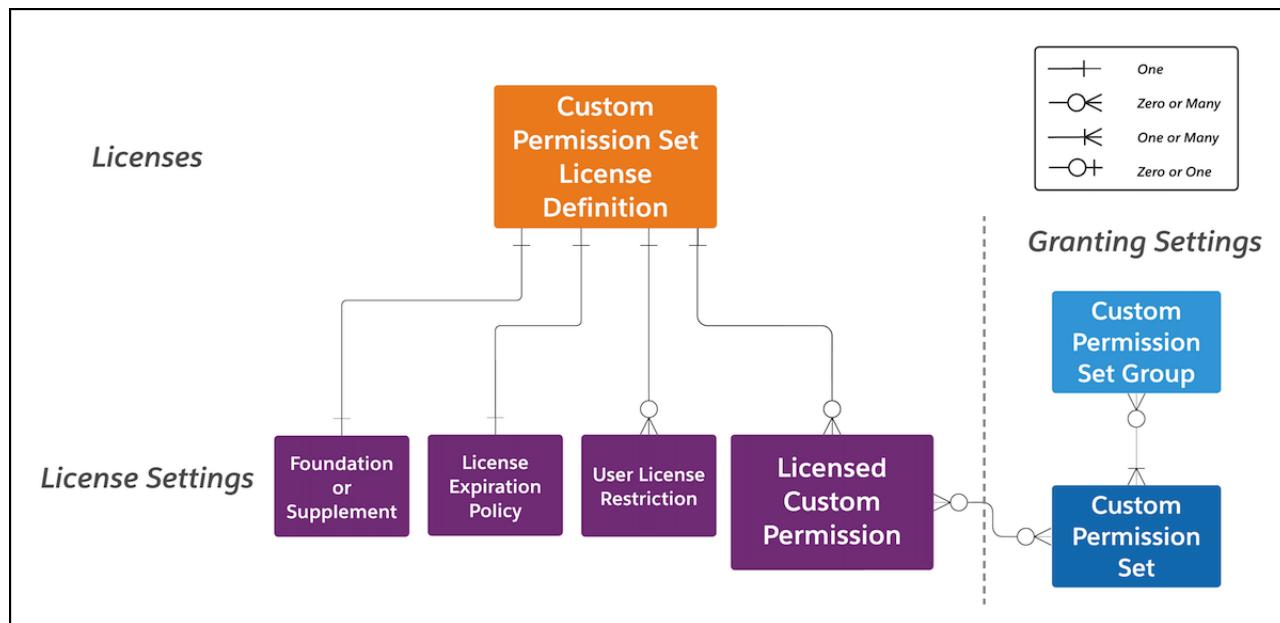
Learn about the components of the Partner Licensing Platform and how they're connected.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Our [forward-looking statement](#) applies to partner business org provisioning. Make your purchasing decisions based on currently available technology.

Before you dig into this material on the Partner Licensing Platform, we recommend that you take the [Salesforce Licensing](#) Trailhead module for an overview of licensing at Salesforce.

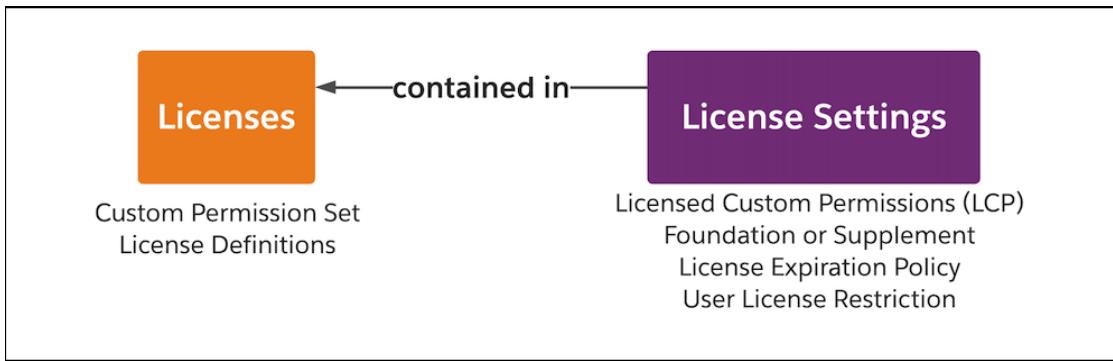
This diagram summarizes the main components that you create and how they're structured.



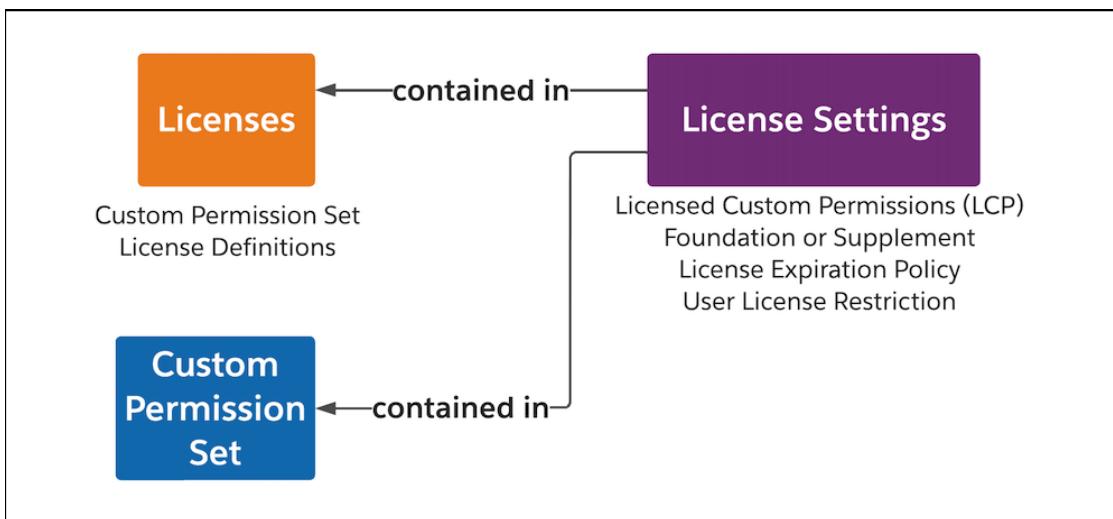
At a high level, you create custom permission set license definitions that are a part of your managed package metadata, just like custom objects and permissions. Then, you create license settings, such as licensed custom permissions, that are contained in the custom permission set license definitions. The licensed custom permissions are also contained within custom permission sets, which in turn can be bundled into permission set groups. When creating a custom permission set license definition, you also specify whether it's a foundation or supplement license and which user licenses can be assigned the license. We go into more detail on these settings in the other topics in this section.

Let's go one step further and look in this diagram how these components and their relationships are used to gate a feature in your package.

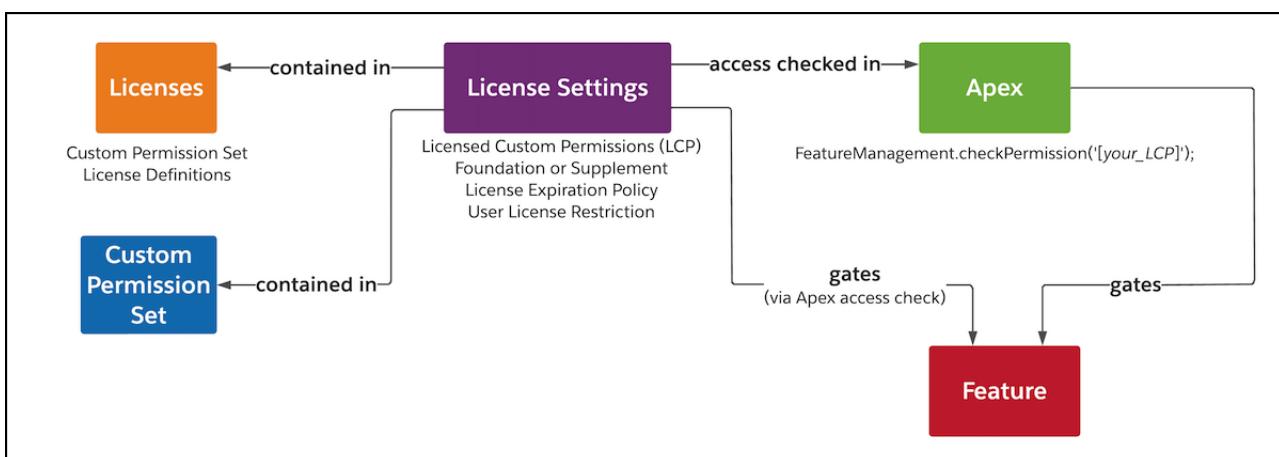
For example, you have a feature that you want to sell and provision. First, you create a licensed custom permission in your development scratch org, then put this custom permission into a new custom permission set license definition, which **entitles** this custom permission.



Next, you put the custom permission into a custom permission set, which **grants** this custom permission.



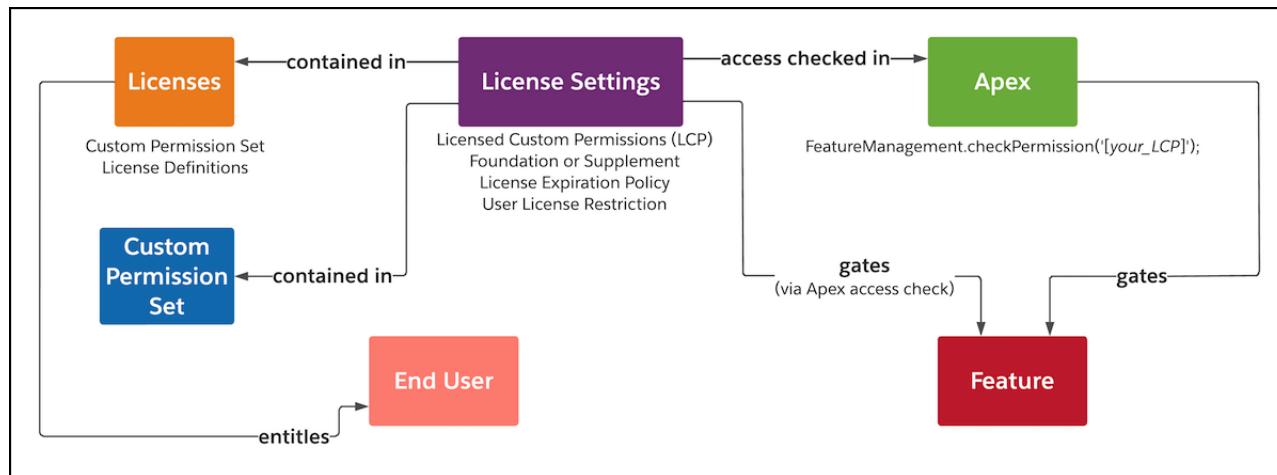
Finally, you put an access check for the licensed custom permission in your Apex code for a specific feature. This access check gates access to the feature.



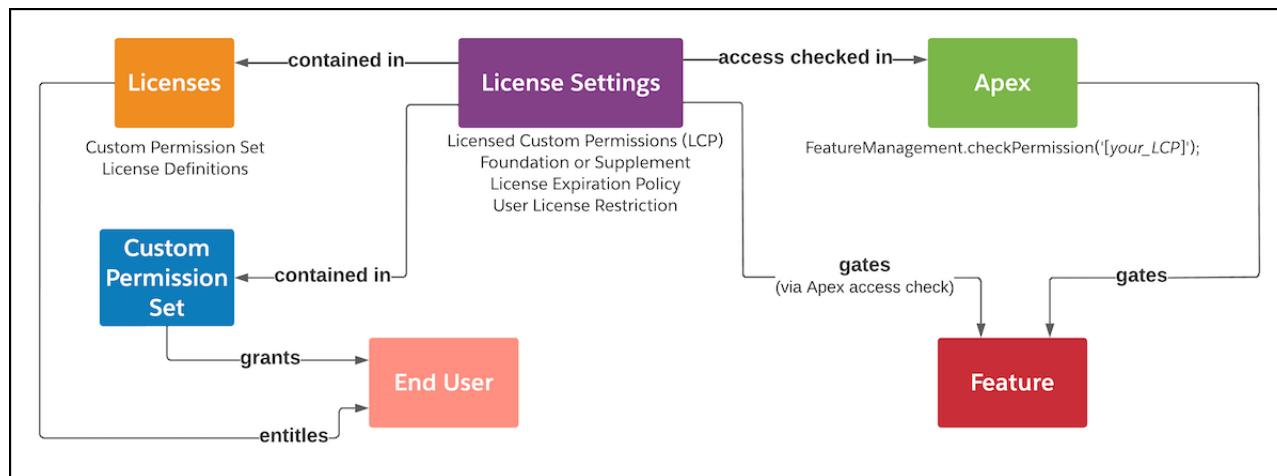
All that's left is uploading and shipping your managed package with these entities. The configuration of licensing as package metadata is separate from provisioning specific quantities of seats for your licenses for a subscriber.

Next, when the functionality is available, you'll provision a number of seats for your custom permission set licenses to each of your subscribers from your partner business org.

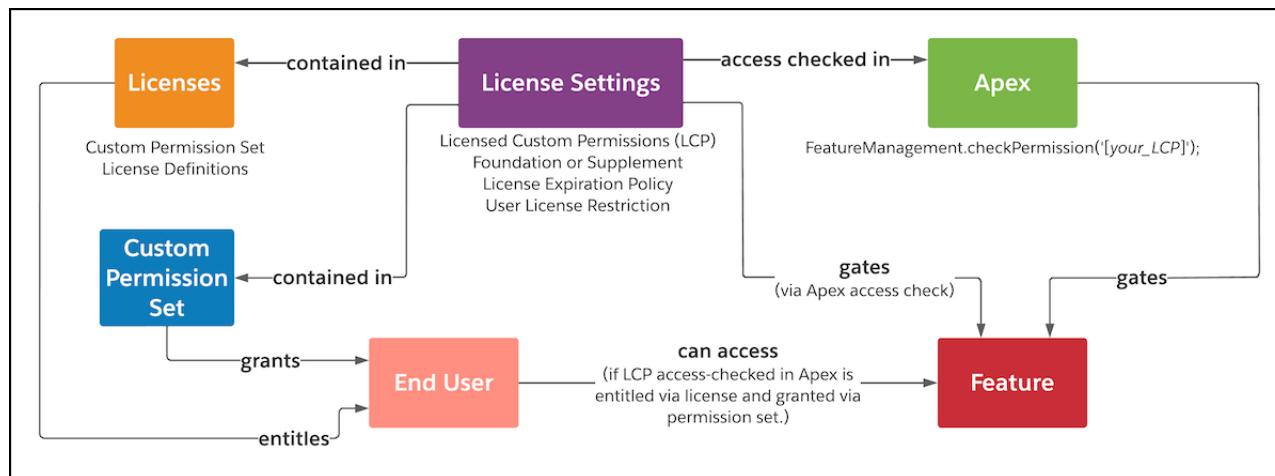
On the subscriber's side, they receive the new metadata as part of your package and a number of custom permission set license seats. First, the subscriber admin assigns the custom permission set license to an end user, which **entitles** the user the licensed custom permission, or feature. This entitlement allows the user to be granted the permission by the admin in the next step.



Next, the admin assigns the custom permission set to the end user, which **grants** that user the licensed custom permission, or the feature. Since this permission set contains a licensed custom permission, it can only be assigned to the user after the user has a license assigned with the licensed custom permission.



The end user can now access the feature gated by the licensed custom permission, because the user is both **entitled** by a license assignment and **granted** access by a permission set assignment.



This process of assigning licenses and permission sets results in a two-step access check for accessing a specific feature. Since the quantity of each license available to a subscriber is provisioned from your partner business org, once available, this mechanism allows both you and your subscribers granular control for which features are purchased and then how they are entitled and granted to each user.

**Note:** Partner business org provisioning of custom permission set license seats isn't available in the developer preview. The channels and interface for provisioning custom permission set license seats are to be determined and will likely be separate from the current License Management App. For testing in the developer preview, the subscriber org where the managed package is installed is provisioned a default count of 10 seats for each custom permission set license in the package.

In these topics, we go into more detail about individual licensing components as well as entitlements and grants.

#### Licensing Components (Developer Preview)

Review the different components that are created as part of your licensing and packaging structure and how these pieces fit together.

#### Entitlements and Grants (Developer Preview)

For users to be able to use a package feature, they must first have access to the overall package. Then, they must be entitled to the feature with a license assignment, and then granted access to the feature with a permission set assignment.

## Licensing Components (Developer Preview)

Review the different components that are created as part of your licensing and packaging structure and how these pieces fit together.

**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Our [forward-looking statement](#) applies to custom supplement permission set licenses, user license restrictions, and licensing and provisioning integration with Sales Cloud. Make your purchasing decisions based on currently available technology.

## Custom Permission Set Licenses

A custom permission set license lets Salesforce Partners and ISVs incrementally sell user-level functionality for individual users. By assigning custom permission set licenses to users, admins can extend a user's entitlements, just like Salesforce-managed [permission set licenses](#).

Admins in subscriber orgs can view their custom permission set licenses in the Company Information page in Setup, after their Salesforce org is provisioned custom permission set licenses. Then, they can assign custom permission set licenses, along with their related custom permission sets, to users who need the permissions in the license.

When an admin assigns a user a permission set that grants access to package components, the system first validates whether the user has a custom foundation permission set license that allows access to the package namespace. For the original managed package licenses, package access isn't validated until the user attempts to access the components. This delay can cause an inconsistent authorization experience, because admins can assign users permission sets that grant access to components in a package, while those users may not have access to the package itself via a license. For more information on package access validation when migrating to the Partner Licensing Platform, see [this topic](#).

## User-Level License Settings

User-level license settings gate individual features of your product for each user and are **entitled** by a license.

### Licensed Custom Permission

A regular custom permission but with the **License Required** checkbox enabled. The field indicates that this custom permission is licensed and must first be entitled via a custom permission set license for a user to be able to access it via a custom permission set.

### Foundation or Supplement

You can indicate whether the custom permission set license is a foundation or supplement license. The default, custom foundation permission set licenses, are like managed package licenses. They entitle and grant access to a user to enter the package namespace, which is required before any further use of the package or its components. A user's managed package license assignment can be replaced with a custom foundation permission set license and the user will retain the same level of access to all the "foundation" features of the package. Foundation features are the package components other than licensed custom permissions, such as non-licensed custom permissions, custom objects, Apex classes, and Visualforce pages. Custom foundation permission set licenses can also entitle the user to specific licensed features. Note that a permission set assignment is still required to grant the user access to specific components of the managed package, such as custom objects.

Custom supplement permission set licenses can only entitle the user to specific licensed features and not the overall package. Supplement licenses are usually used for additional functionality that you want to sell on top of your foundation features and access to your package, as these licenses require the user to have at least one foundation license assigned first.

### User License Restriction

You can specify the users that can be assigned the custom permission set license using user license restriction categories. For example, you can restrict the license to only be assigned to internal Salesforce users or external Experience Cloud users. For more information on the available categories and the included user licenses, see [User License Restriction Categories \(Developer Preview\)](#).

### License Expiration Policy

You can choose whether to allow or block package access when all custom foundation permission set licenses expire. If you allow package access, users with the expired licenses can continue to access the package namespace and its components. However, the subscriber admin can't assign any more of these licenses to users and can't reassign the expired license to a user after it's removed. If you block package access, users with the expired licenses can no longer access the package namespace or its components upon expiration. This option is how the original managed package licenses operate.

 **Note:** If only some of the licenses have expired, all licenses continue to allow access to the package namespace and its components until all licenses expire.

## Permission Sets and Permission Set Groups

Permission set and permission set groups **grant access** to a license's settings, such as a licensed custom permission, or other regular components of your package.

### Permissions Sets

Permission sets are groups of permissions and settings that extend users' functional access without changing their profiles. You create custom permission sets in your package that contain the licensed custom permissions. After a user becomes **entitled** to permissions via a license, admins use permission sets to actually **grant access** to these permissions. Permission sets are defined in the managed package as part of the package metadata.

### Permission Set Groups

Permission set groups bundle permission sets together, for example, based on user job functions. Users assigned to the permission set group receive the combined permissions of all the permission sets in the group. Depending on the personas for whom you're creating your features, consider using permission set groups. Permission set groups provide admins with flexibility, since the admins can choose to mute specific permissions in a permission set group, depending on their business needs.

## Order Management

You can use the Sales Cloud functionality that comes with your partner business org or another order management system (CRM). We recommend that you use Sales Cloud for its enhanced functionality and for built-in licensing and provisioning integration that may be available in a future release.

### SEE ALSO:

[Salesforce Help: Permission Sets](#)

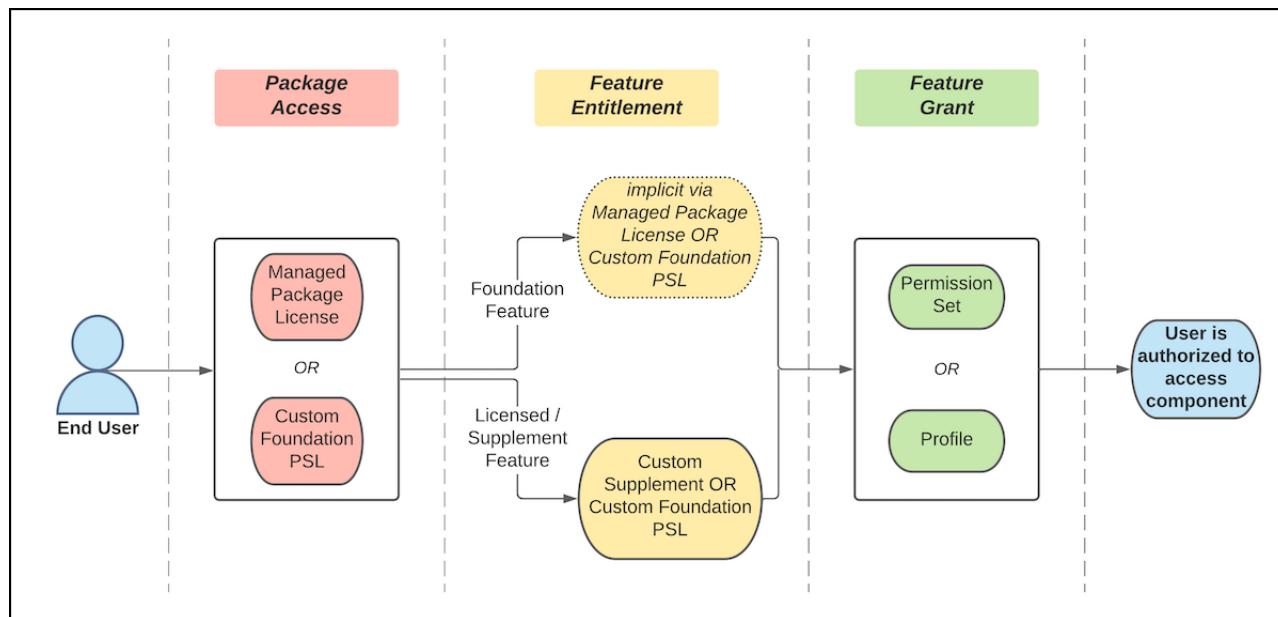
[Salesforce Help: Permission Set Groups](#)

## Entitlements and Grants (Developer Preview)

For users to be able to use a package feature, they must first have access to the overall package. Then, they must be entitled to the feature with a license assignment, and then granted access to the feature with a permission set assignment.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Our [forward-looking statement](#) applies to partner business org provisioning. Make your purchasing decisions based on currently available technology.



## Package Access

Package access is the ability for a user to enter the namespace of a package. This access doesn't include further entitlements or grants necessary for the user to access any components of the package. However, it's necessary for a user to first have package access before receiving any further use of the package or its components.

Either a managed package license or a custom foundation permission set license (PSL) can provide package access to a user for a given package or namespace. A permission set or profile isn't necessary to give the user package access.

## Foundation Feature Entitlement

Managed packages primarily contain foundation features, such as non-licensed custom permissions, custom objects, Apex classes, and Visualforce pages. All components of your package are considered foundation features, except for licensed custom permissions. Foundation features aren't explicitly licensed or named in a custom permission set license, which means a user doesn't require an explicit entitlement to access these features. Their entitlement is given to the user implicitly via the assignment of a managed package license or via a custom foundation permission set license. Both of these licenses provide implicit entitlement for all foundation features in your package, including as your package changes over time.

## Licensed Feature Entitlement

Licensed or supplement features are features that are explicitly licensed. These features are gated with licensed custom permissions that are named in a custom permission set license. A user must have an explicit entitlement to access these features. When the admin assigns the user a custom permission set license, it provides the **entitlement** to the license settings or features it contains. Admins can then grant users access to the features they're entitled to with a permission set. The assigned licenses define the maximum set of features entitled to the user, but don't grant access to those features. The only package components that can explicitly be both entitled and granted for end users are licensed custom permissions.

Licensed features can be included in either a custom foundation or a custom supplement permission set license. Custom supplement permission set licenses don't give the user package access or entitlement to access the foundation features of the package. They can only give entitlement for licensed or supplement features of a package.

## Feature Grant

Finally, the admin must assign the user a permission set to **grant access** to the license settings (or features) already entitled by a license assignment. Now, the user has access to the package and is authorized to access this feature.

This process of assigning licenses and permission sets results in a two-step access check for accessing a specific feature. Since the quantity of each license available to a subscriber is provisioned from your partner business org, once available, this mechanism allows both you and your subscribers granular control for which features are purchased and then how they are entitled and granted to each user.

For example, a subscriber purchases five seats of a custom permission set license that contains three features. The subscriber admin can assign one seat to a user, giving the user the **entitlement** for those three features. Then the admin can assign a permission set to the user, giving them the **grant** for only one of those features. In this scenario, the admin has authorized the user to access only one feature, and not the other two available with the license.

# Design Your Package Licensing Structure (Developer Preview)

---

To make sure that your license development process proceeds as smoothly as possible, design the feature breakdown of your product and its associated licenses carefully before you begin implementation.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

### [Outline Pricing Feature Strategy \(Developer Preview\)](#)

Define what features and functionality you want to sell as part of your product offering. Then, map these features to the products that you'll sell.

### [Identify Licenses \(Developer Preview\)](#)

After you develop your feature pricing strategy, identify the licenses that each product contains. Your licenses contain the underlying features that your products entitle.

### [Identify License Settings Use Cases \(Developer Preview\)](#)

Identify the correct settings to include in your licenses that gate specific functionality so that users have the access to the intended features.

### [Map Settings to Licenses \(Developer Preview\)](#)

After you confirm a list of license settings for your features, decide which settings to include in which custom permission set licenses.

### [Design Permission Sets \(Developer Preview\)](#)

The design of permission sets is based on your intended end-user personas and the appropriate access they require to perform their tasks. The goal is to enable your customers to have a convenient administration experience.

### [Review Your Structure \(Developer Preview\)](#)

Review your design to ensure that your licensing and permissions are structured as you need for your feature pricing and licensing strategy.

## Outline Pricing Feature Strategy (Developer Preview)

Define what features and functionality you want to sell as part of your product offering. Then, map these features to the products that you'll sell.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

## Plan Your Packaging Strategy

At this point in the design process, differentiate which features are available at a user-level vs. org-level. For user-level features, your customers purchase and assign a license for each user who needs access. For org-level features, your customers can purchase one product and all of their users, or their whole Salesforce org, have access to that feature.

For example, say you're developing a product called Travel Navigation, which includes:

- A basic set of functionality, such as being able to view, create, and edit visual maps with custom labels.
- An add-on for enhanced functionality, such as being able to color code labels and add symbols.
- An add-on for functionality around territories, which allows creating visual territories and subterritories.

You plan to sell each of these three sets of functionalities separately on a per-user basis.

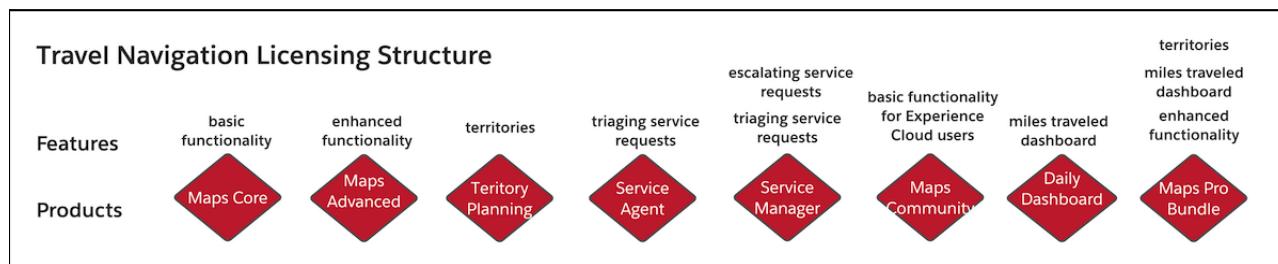
Your product also has a miles traveled dashboard functionality, which shows your customer how many miles in a given day that a sales or service person travels. You plan to sell this dashboard at the org-level.

## Identify Products

Next, map the features to the products that you want to sell and that entitle the customer to those features. For each of the distinct features that you define in your pricing and feature strategy, you can have one or more products that contain a feature. You can also have more than one feature in one product.

For our Travel Navigation example, you create this matrix to outline your product mapping.

Feature	Relationship	Product
basic functionality	contained in	Maps Core
enhanced functionality	contained in	Maps Advanced
territories	contained in	Territory Planning
triaging service requests	contained in	Service Agent
triaging service requests	contained in	Service Manager
escalating service requests	contained in	Service Manager
basic functionality for Experience Cloud users	contained in	Maps Community
miles traveled dashboard	contained in	Daily Dashboard
territories	contained in	Maps Pro Bundle
miles traveled dashboard	contained in	
enhanced functionality	contained in	



Note that the triaging service requests feature is contained in both Service Agent and Service Manager products and that Service Manager contains two features.

We recommend using a visual diagramming tool to build your licensing design.

#### SEE ALSO:

[Licensing Components \(Developer Preview\)](#)

## Identify Licenses (Developer Preview)

After you develop your feature pricing strategy, identify the licenses that each product contains. Your licenses contain the underlying features that your products entitle.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Consider what custom permission set licenses you must create. Custom permission set licenses are assigned to users and give the entitlements contained within the custom permission set license to the users.

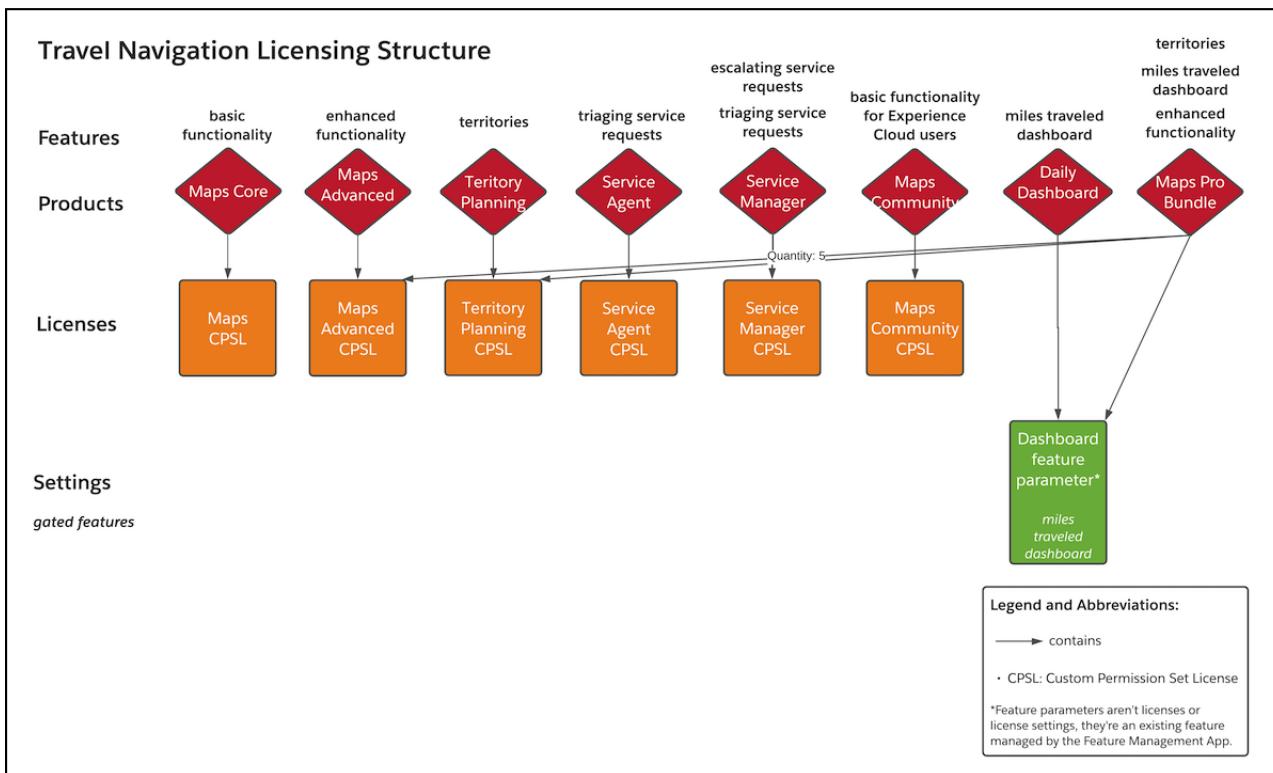
Next, consider how to include your set of licenses into your set of products. When a product is purchased by your customers, you'll provision the underlying licenses to their org. The customer then assigns these licenses and permission sets to users to access the functionality entitled by the license.

For Travel Navigation, you create this matrix to outline the relationship between your products and custom permission set licenses (CPSLs).

Product	Relationship	License
Maps Core	contains 1	Maps CPSL
Maps Advanced	contains 1	Maps Advanced CPSL
Territory Planning	contains 1	Territory Planning CPSL
Service Agent	contains 1	Service Agent CPSL
Service Manager	contains 1	Service Manager CPSL
Maps Community	contains 5	Maps Community CPSL
Daily Dashboard	contains 1	Dashboard feature parameter*
Maps Pro Bundle	contains 1	Dashboard feature parameter*

Product	Relationship	License
	contains 1	Territory Planning CPSL
	contains 5	Maps Advanced CPSL

\*Feature parameters aren't licenses. They are an existing feature and are managed by the Feature Management App.



Note that the Maps Pro Bundle product contains multiple licenses, and some individual licenses are contained within multiple products. This many-to-many mapping of licenses and products is common.

#### SEE ALSO:

[Manage Features](#)

[Licensing Components \(Developer Preview\)](#)

## Identify License Settings Use Cases (Developer Preview)

Identify the correct settings to include in your licenses that gate specific functionality so that users have the access to the intended features.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

In this table, we summarize the licensing mechanism for each type of access for your product.

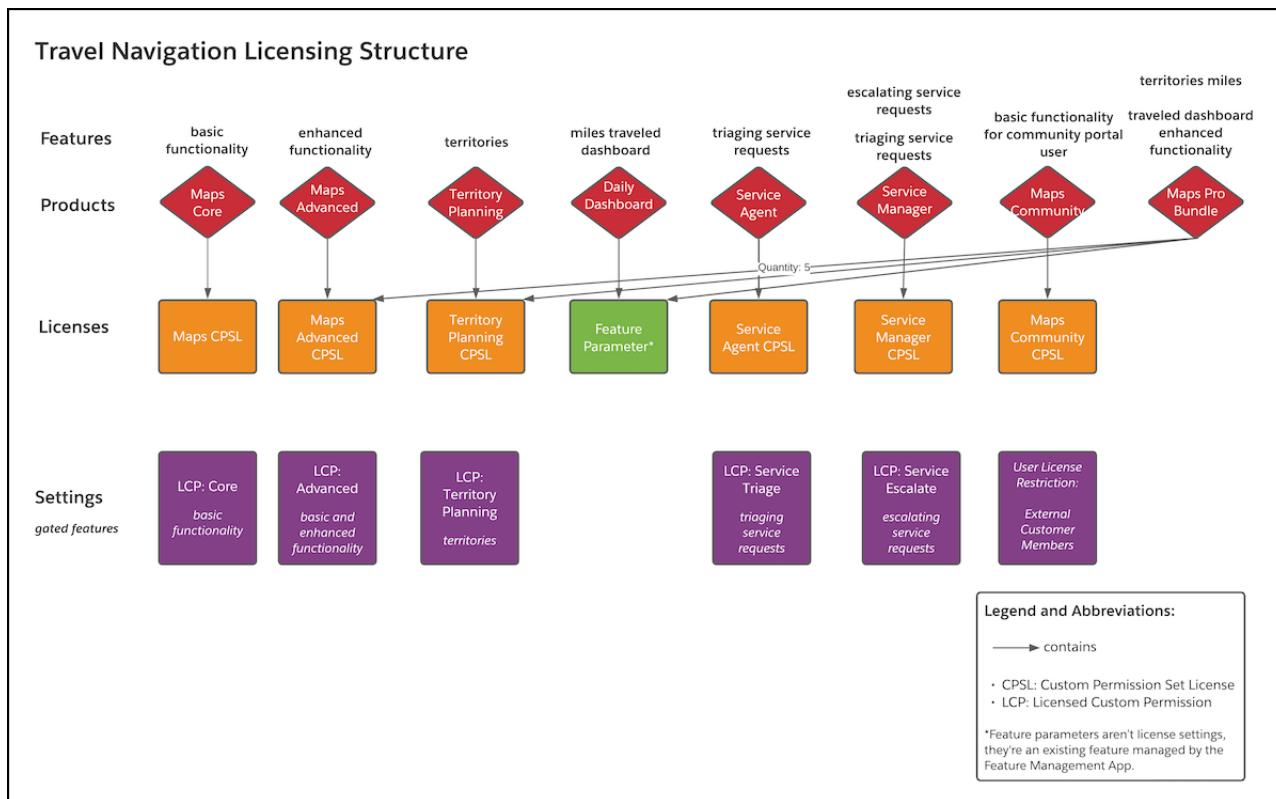
	<b>Feature Access</b>	<b>Object Access</b>	<b>Package Access</b>	<b>Usage Entitlement</b>
<b>User-Level</b>	Licensed Custom Permission	<i>In Consideration</i>	Managed Package License or Custom Foundation Permission Set License	Not Applicable
<b>Org-Level</b>	LMO-to-Subscriber Boolean Feature Parameter*	Custom Object	Site-wide license setting in LMA	Subscriber-to-LMO Integer Feature Parameter (usage metric) and LMO-to-Subscriber Integer Feature Parameter (usage limit) pairing*

\*Feature parameters aren't license settings. They are an existing feature and are managed by the Feature Management App.

Continuing our Travel Navigation example, we create this matrix to capture the exact settings and licensed custom permissions (LCPs), and their relation to your identified features.

<b>Setting</b>	<b>Relationship</b>	<b>Feature</b>
Custom Foundation Permission Set License	gates	overall package namespace access
LCP: Core	gates	basic functionality
LCP: Advanced	gates	enhanced functionality
LCP: Territory Planning	gates	territories
LCP: Service Triage	gates	triaging service requests
LCP: Service Escalate	gates	escalating service requests
LCP: Core and User License Restrictions: External Customer Members	gates	basic functionality for Experience Cloud users
Dashboard feature parameter*	gates	miles traveled dashboard

\*Feature parameters aren't license settings. They are an existing feature and are managed by the Feature Management App.



For examples of gating access in other pricing scenarios, refer to these topics.

#### User-Level Feature Gating (Developer Preview)

When you're ready to design your user-level license settings, review these common scenarios and suggested solutions.

#### Org-Level Feature Gating (Developer Preview)

When you're ready to design your org-level licensing strategy, review these common scenarios and suggested solutions.

#### Advanced Use Cases and Compound Gating (Developer Preview)

Review advanced pricing scenarios that require compound solutions or specific configurations.

#### SEE ALSO:

[Licensing Components \(Developer Preview\)](#)

[Drive App Behavior with LMO-to-Subscriber Feature Parameters](#)

[Track Preferences and Activation Metrics with Subscriber-to-LMO Feature Parameters](#)

[User License Restriction Categories \(Developer Preview\)](#)

## User-Level Feature Gating (Developer Preview)

When you're ready to design your user-level license settings, review these common scenarios and suggested solutions.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands,

parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Requirements	Solution
You want to sell a feature via a license that must be bought and assigned individually for each user.	Licensed custom permissions. These permissions are checked in your Apex code during runtime. The user is given access to the gated feature if the user has a custom permission set license containing the licensed custom permission and a permission set assigned that grants them the licensed custom permission.
You want to sell access to your whole package via a license that must be bought and assigned individually for each user. You're migrating away from managed package licenses, which perform a similar function.	A custom foundation permission set license. This license gives the user the ability to enter the package namespace and the entitlement to access all <a href="#">foundation features</a> , similar to managed package licenses.
You want to sell a feature that your subscriber can purchase separately for their standard internal Salesforce users and external Experience Cloud site users. For example, you want to sell a product and license for internal users for \$20 per user per month and a similar but separate product and license for external users for \$2 per user per month. You want to ensure that the subscriber doesn't use the cheaper license for a more expensive type of user.	<a href="#">User license restrictions</a> specified in two custom permission set licenses for each user category.

#### SEE ALSO:

- [Identify License Settings Use Cases \(Developer Preview\)](#)
- [Licensing Components \(Developer Preview\)](#)
- [Org-Level Feature Gating \(Developer Preview\)](#)
- [Advanced Use Cases and Compound Gating \(Developer Preview\)](#)

## Org-Level Feature Gating (Developer Preview)

When you're ready to design your org-level licensing strategy, review these common scenarios and suggested solutions.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Requirements	Solution
You want to sell a feature via a product that must be bought only one time to give all users in the Salesforce org access to the feature.	<p>Feature parameter of type boolean. This is access checked in your Apex code during runtime and gives any user in the org access to the gated feature. No permission set or custom permission set license assignment is required for a user to access the feature.</p> <p> <b>Note:</b> Feature parameters are managed by the Feature Management App. The Feature Management App is an</p>

Requirements	Solution
<p>You want to sell access to your whole package via a product that must be bought only one time to give all users in the Salesforce org access to your whole package.</p>	<p>existing feature that is separate from the Partner Licensing Platform.</p> <p>Use the License Management App to set licensing for a particular subscriber to be site-wide.</p> <p> <b>Note:</b> The License Management App is an existing feature that is separate from the Partner Licensing Platform.</p> <p>Note that permission set assignments for various permissions, such as custom object permissions, are still required for users in the org to access those features.</p>
<p>You want to sell a consumption-based entitlement, such as a number of API calls or call center minutes, in a specific quantity for the entire Salesforce org. This quantity is consumed by any user accessing this entitlement in the org.</p>	<p>At a high level, this requirement can be achieved via a <a href="#">LMO-to-Subscriber feature parameter</a> to track the total sold quantity and a <a href="#">Subscriber-to-LMO feature parameter</a> for tracking the used quantity from your package code. There may be other considerations depending on your specific needs, such as billing, usage periods, overages, and restricting or allowing access when the total sold quantity is consumed.</p> <p> <b>Note:</b> Feature parameters are managed by the Feature Management App. The Feature Management App is an existing feature that is separate from the Partner Licensing Platform.</p>

#### SEE ALSO:

- [Identify License Settings Use Cases \(Developer Preview\)](#)
- [Licensing Components \(Developer Preview\)](#)
- [User-Level Feature Gating \(Developer Preview\)](#)
- [Advanced Use Cases and Compound Gating \(Developer Preview\)](#)

## Advanced Use Cases and Compound Gating (Developer Preview)

Review advanced pricing scenarios that require compound solutions or specific configurations.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

### Example: Org-Level Feature with User-Level Licenses

**Requirement:** You want to sell access to a feature via a product that must be bought one time for the org and a license that must be bought and assigned individually for each user. The Salesforce org must have both the org-level product and the user-level license assigned for a user to be able to access the feature.

For example, an intelligence dashboard, which you want to sell for \$200/month for the org to have access as the standard price, and then an incremental \$25/user/month for each user to access the dashboard.

**Solution:** A combination of a boolean feature parameter and a licensed custom permission. Both the feature parameter and licensed custom permission are access checked in Apex code during runtime. If the user has a custom permission set license containing the licensed custom permission and the org has the feature parameter turned on, the user gets access to the feature. Remember that the user must also have a permission set assigned that grants them access to the licensed custom permission.



**Note:** Feature parameters are managed by the Feature Management App. The Feature Management App is an existing feature that is separate from the Partner Licensing Platform.

### Example: Org-Level Feature with Admin-Controlled Access

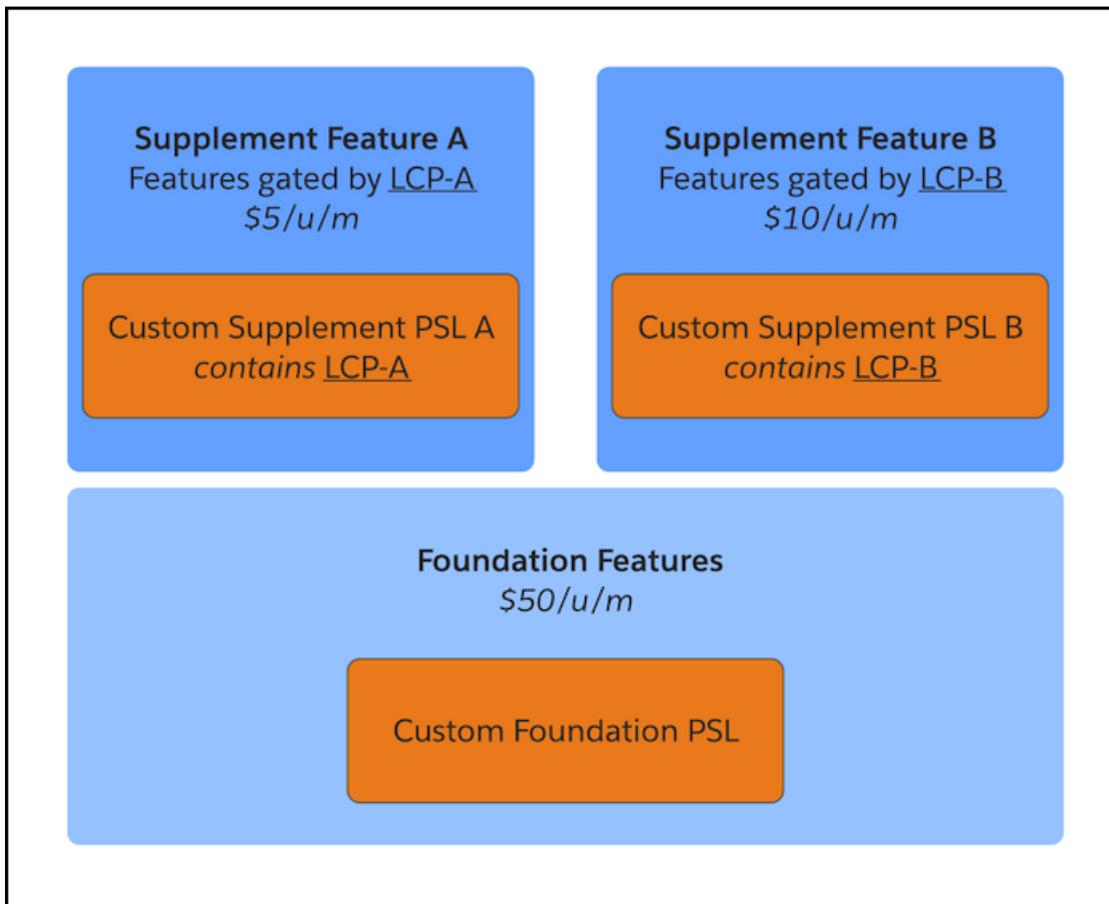
**Requirement:** You want to sell access to a product that must be bought one time for the Salesforce org and no further licenses are needed for each user. However, you want to ensure that the org admin decides which users get access to this feature.

**Solution:** Use a boolean feature parameter to gate access to the feature for an org, and then a regular custom permission to gate access for the user. Both the feature parameter and custom permission are access checked in Apex code during runtime to give access to the feature. Note that the user doesn't require a license for the feature and the org admin can assign the user a permission set that grants them the custom permission.

### Example: Foundation and Supplement Features

**Requirement:** You want to sell access to foundation features in one license, and then sell supplement features in supplement licenses. You want to ensure that the supplement licenses don't give access to the foundation features and that a user must first be assigned the foundation license to receive the supplement licenses.

For example, you sell access to foundation features and your package namespace for \$50 per user per month. Then, you sell supplement feature A for \$5 per user per month, and supplement feature B for \$10 per user per month. You want to ensure that the customer doesn't get access to the foundation features and your overall package only by assigning the cheaper supplement license.



**Solution:** Create a custom foundation permission set license for your foundation features. This license entitles access to all your foundation features, including your package namespace, and doesn't entitle access to the supplement features A or B.

Then, create custom supplement permission set license A containing licensed custom permission A, and custom supplement permission set license B containing licensed custom permission B. Your feature A and feature B are gated by licensed custom permissions A and B in Apex code, respectively. Since these two licenses are supplement licenses, they don't give access to your foundation features, including your package namespace.

Since your custom supplement permission licenses don't give access to your foundation features, a user must first have your custom foundation permission set license assigned in order to receive either or both of the custom supplement permission set licenses. This setup ensures that the customer can't assign only a cheaper supplement license and get access to all the foundation features, including your package namespace.

Additionally, users sometimes only require access to the foundation features and not the additional supplement features. These users can be assigned only the custom foundation permission set license and not the supplement licenses.

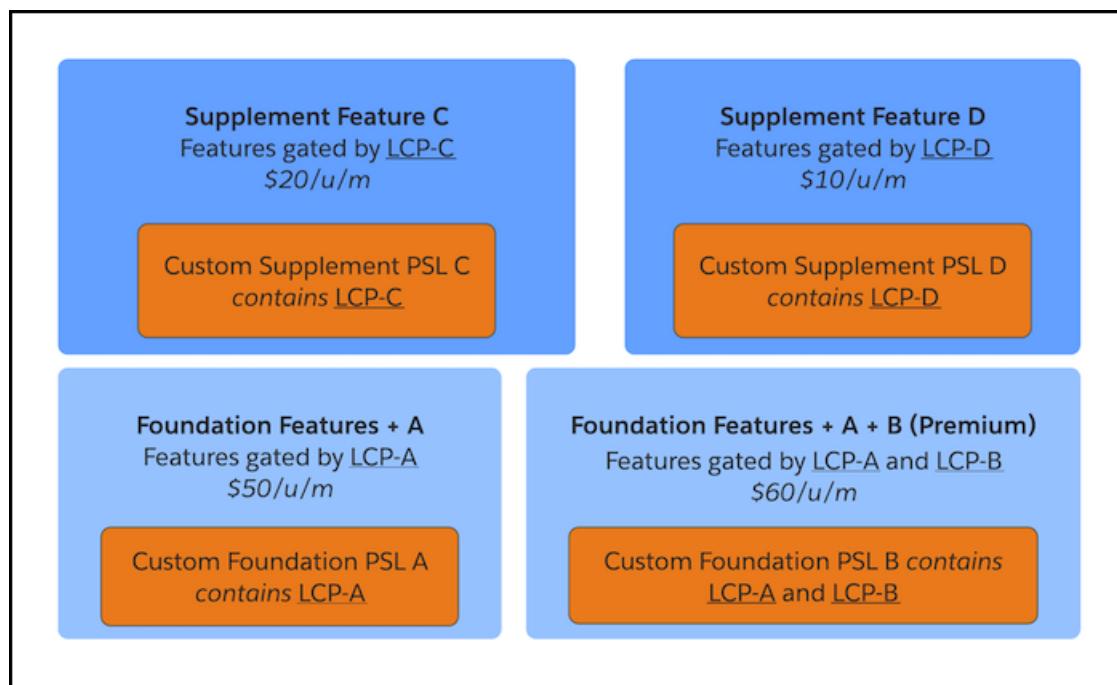
### Example: Multiple Foundation and Supplement Features

**Requirement:** You want to sell two versions of foundation licenses, where every user of your package must have at least one of the two foundation licenses assigned. One of the foundation licenses contains an additional feature A and the other contains additional features A and B.

Then, you also want to sell supplement features in supplement licenses. You want to ensure that the supplement licenses don't give access to the foundation features and that a user must first be assigned a foundation license to receive the supplement licenses.

For example, you sell access to the foundation features, your package namespace, and feature A for \$50 per user per month. You also sell access to the foundation features, your package namespace, and features A and B for \$60 per user per month. These two licenses are only differentiated by which additional features they entitle, and are considered as two “versions” of your product, one being more premium with an additional feature.

Then, you sell supplement feature C for \$20 per user per month, and supplement feature D for \$10 per user per month. You want to ensure that the customer doesn’t get access to the foundation features and your overall package only by assigning the cheaper supplement license.



**Solution:** Create a custom foundation permission set license A containing licensed custom permission A. Create a second custom foundation permission set license B containing licensed custom permissions A and B. Your feature A and feature B are gated by licensed custom permissions A and B in Apex code, respectively. These licenses entitle access to all your foundation features, including your package namespace, and either feature A or features A and B. They don’t entitle access to the supplement features C or D.

Then, create custom supplement permission set license C containing licensed custom permission C, and custom supplement permission set license D containing licensed custom permission D. Your feature C and feature D are gated by licensed custom permissions C and D in Apex code, respectively. Since these two licenses are supplement licenses, they don’t give access to your foundation features, including your package namespace.

Since your supplement licenses don’t give access to your foundation features, a user must first have one of your custom foundation permission set licenses assigned in order to receive either or both of the custom supplement permission set licenses. This setup ensures that the customer can’t assign only a cheaper supplement license and get access to all the foundation features, including your package namespace.

Additionally, users sometimes only require access to the foundation features and not the supplement features. These users can be assigned only one of the custom foundation permission set licenses and not the supplement licenses.

An alternative approach is that you could consider having only the licensed custom permission B in your premium custom foundation permission set license. In this case, you would gate access to feature A with either licensed custom permission A or B. This way, the user with the premium custom foundation permission set license would get access to both features A and B with only one “premium” licensed custom permission B. This may be desired if you’d like to present licensed custom permission B as giving access to both features or make

it simpler for your customer admins to authorize both features with one permission. This example illustrates the flexibility of licensing and access check capabilities to meet your desired feature pricing and access strategy.

**!** **Important:** Be deliberate when creating foundation vs. supplement licenses for your package features. Most often, you'll create foundation licenses, as these licenses allow the user to enter the package namespace and are a replacement for the original managed package licenses. Supplement licenses are usually used for additional functionality that you want to sell on top of your foundation features, as these licenses require the user to have at least one foundation license assigned first.

#### SEE ALSO:

- [Identify License Settings Use Cases \(Developer Preview\)](#)
- [Licensing Components \(Developer Preview\)](#)
- [User-Level Feature Gating \(Developer Preview\)](#)
- [Org-Level Feature Gating \(Developer Preview\)](#)

## Map Settings to Licenses (Developer Preview)

After you confirm a list of license settings for your features, decide which settings to include in which custom permission set licenses.

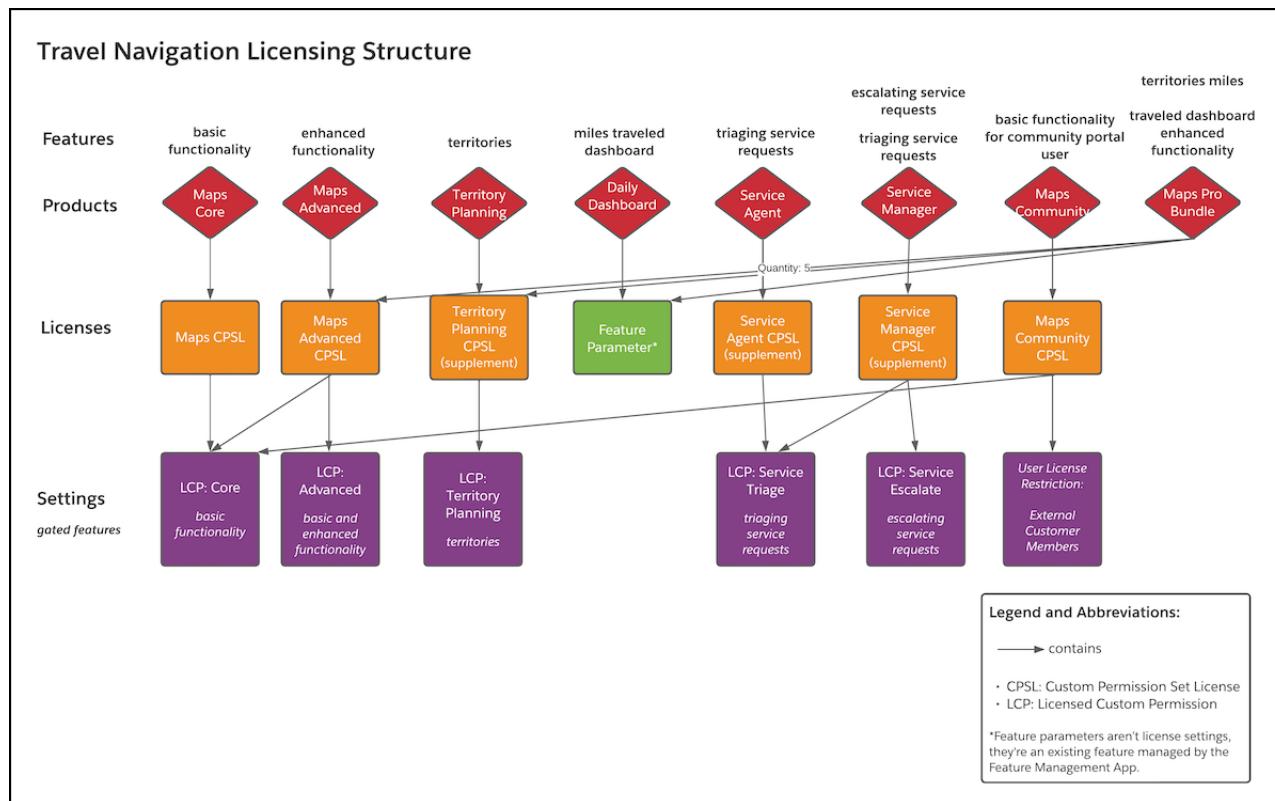
**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Licenses can contain multiple settings, and one setting can be contained in multiple licenses.

For the Travel Navigation product, you create this matrix to match up your settings, or licensed custom permissions (LCPs), to your custom permission set licenses. You also specify which licenses are foundation licenses, which provide access to the package namespace and foundation features, and which are supplement licenses, which only entitle users to additional features, not the package overall.

License	Relationship	Setting
Maps (foundation license)	contains	LCP: Core
Maps Advanced (foundation license)	contains	LCP: Core
		LCP: Advanced
Territory Planning (supplement license)	contains	LCP: Territory Planning
Service Agent (supplement license)	contains	LCP: Service Triage
Service Manager (supplement license)	contains	LCP: Service Triage
		LCP: Service Escalate
Maps Community (foundation license)	contains	LCP: Core and User License Restriction: External Customer Members
Daily Dashboard (product, not license)	contains	Dashboard feature parameter*

\*Feature parameters aren't license settings. They are an existing feature and are managed by the Feature Management App.



**Note:** The Territory Planning, Service Agent, and Service Manager licenses are custom supplement permission set licenses that must be assigned to a user in conjunction with a custom foundation permission set license to be functional.

#### SEE ALSO:

[Licensing Components \(Developer Preview\)](#)

[User License Restriction Categories \(Developer Preview\)](#)

## Design Permission Sets (Developer Preview)

The design of permission sets is based on your intended end-user personas and the appropriate access they require to perform their tasks. The goal is to enable your customers to have a convenient administration experience.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

So far, you've completed the bulk of your licensing design. You have a set of products, licenses, and license settings that will **entitle** your package features according to your feature pricing and licensing strategy. Now, it's time to design your permission sets to enable your customers to easily **grant access** to your package features out of the box.

Your customers often have multiple other products with their own licenses and permission sets. Giving your customers easy-to-understand and out-of-box personas packaged as custom permission sets enables them to administer and onboard onto your product faster.

## Permission Sets

First, review your customer personas. Think through what types of user personas will access your product functionality, what features of your package they'll need access to, and in what capacity.

Next, design the permission sets for these personas. Think of each persona as having one permission set that gives them the required access. In many cases, you can define one permission set that contains all the permissions for the features you intend for a persona.

Keep in mind that your customer admin can always create custom permission sets in their org and assign any combination of permissions from your licenses as well as Salesforce licenses. However, such a permission set can only be assigned to a user who has all the necessary underlying licenses assigned to them.

As a best practice, make sure that no permission set contains more permissions than what its intended, related custom permission set license contains. If one permission set contains permissions from multiple custom permission set licenses, break up that permission set into multiple permission sets. Then use a permission set group to group those permission sets together. This practice ensures that each permission set can be associated with a single custom permission set license, such that it grants some or all of the entitlements from that license, and not more. Also note there can be multiple permission sets that contain license settings from one custom permission set license.

For our Travel Navigation example, you've identified these personas:

Persona	Relationship	Feature
Org Manager	can access	enhanced functionality
		territories
		triaging service requests
		escalating service requests
Service Agent	can access	triaging service requests
Service Manager	can access	triaging service requests
		escalating service requests
Service Support	can access	escalating service requests
Territory Planner	can access	territory planning

Here's one process by which you can design the permission sets for these personas.

1. Create one permission set, which contains one or more licensed custom permissions (LCPs), for each custom permission set license. Then, map them to each other.

Settings/Permission	Relationship	Permission Set	Relationship	Custom Permission Set License
LCP: Core	is contained in	Maps Core	maps to	Maps
LCP: Advanced	is contained in	Maps Advanced	maps to	Maps Advanced
LCP: Territory Planning	is contained in	Territory Planning	maps to	Territory Planning
LCP: Service Triage	is contained in	Service Agent	maps to	Service Agent
LCP: Service Triage	is contained in	Service Manager	maps to	Service Manager

Settings/Permission	Relationship	Permission Set	Relationship	Custom Permission Set License
LCP: Service Escalate				

2. Consider if a particular persona needs only a subset of the access from a custom permission set license. Let's say that you have a custom permission set license that maps to four permissions, but the persona only needs two of them. Create another permission set with the limited access.

In this example, the Service Support persona doesn't need all the permissions licensed by the Service Manager custom permission set license, and similarly needs less access than the Service Manager persona. So we add the following permission set:

Settings/Permission	Relationship	Permission Set	Relationship	Custom Permission Set License
LCP: Service Escalate	is contained in	Service Support	maps to	Service Manager

## Permission Set Groups

A [permission set group](#) streamlines permissions assignment and management by grouping together multiple permission sets based on user job functions. Users assigned the permission set group receive the combined permissions of all the permission sets in the group.

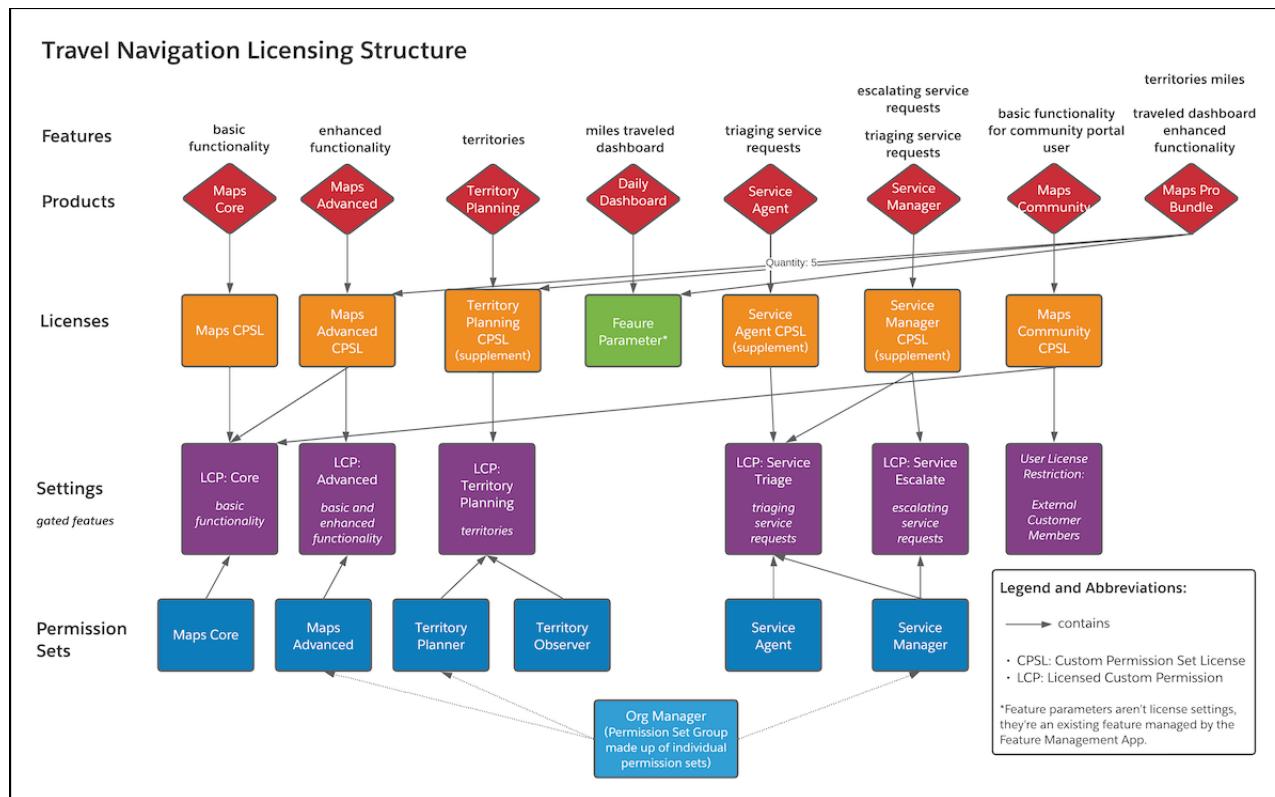
In our example, we check if a particular persona needs a superset of access from more than one permission set, and group those permission sets into a permission set group. The Org Manager overseeing both the maps and service teams must access the advanced maps functionality, territories functionality, and all service agent functionality. Note that the feature access of this persona crosses over multiple custom permission set licenses.

We create the following permission set group:

Permission Set Group	Relationship	Permission Set	Relationship	Settings/Permission
Org Manager	contains	Maps Advanced	contains	LCP: Advanced
		Territory Planner	contains	LCP: Territory Planning
		Service Manager	contains	LCP: Service Triage
				LCP: Service Escalate

Remember that assigning the Org Manager permission set group to a user grants the user all permissions from the underlying permission sets. The permission sets themselves aren't directly assigned on their own. If subscriber admins don't want to grant every permission from the permission set group, they can mute a specific permission in the group [with a muting permission set](#).

Finally, assess whether any permission set is unnecessary, meaning the personas you designed are getting the access they need through another permission set or permission set group.



## Design Notes

- The Maps Advanced custom permission set license includes both Core and Advanced licensed custom permissions, but the Maps Advanced permission set only includes the Advanced permission. This structure implies that both the basic Core functionality and enhanced Advanced functionality can be accessed by the Advanced licensed custom permission. The Apex code gating those two functionalities accounts for this design choice. It access checks for either the Core or Advanced licensed custom permission when gating basic functionality, and access checks only the Advanced permission when gating enhanced functionality. Alternatively, you could include both Core and Advanced licensed custom permissions in the Maps Advanced permission set, and access check basic and enhanced functionality separately with Core and Advanced licensed custom permissions, respectively.
- A user needing only basic functionality can still be assigned the Maps Advanced permission set license. Then, the user is assigned a permission set with the Core licensed custom permission only so that they're granted access only to basic functionality while having the Maps Advanced permission set license.
- In this example, an Experience Cloud functionality permission set isn't designed. Instead, we're using the External Customer Member [user license restriction category](#) to control who can be assigned the permission set license.

 **Note:** There are many ways by which you can design your permission sets. This topic is a guideline to help you think through your personas, but you can design your permission sets in a different way. Ultimately, ease of administration and out-of-box configuration for your customers is the goal.

## SEE ALSO:

- [Licensing Components \(Developer Preview\)](#)
- [Entitlements and Grants \(Developer Preview\)](#)
- [Salesforce Help: Permission Sets](#)
- [Salesforce Help: Permission Set Groups](#)

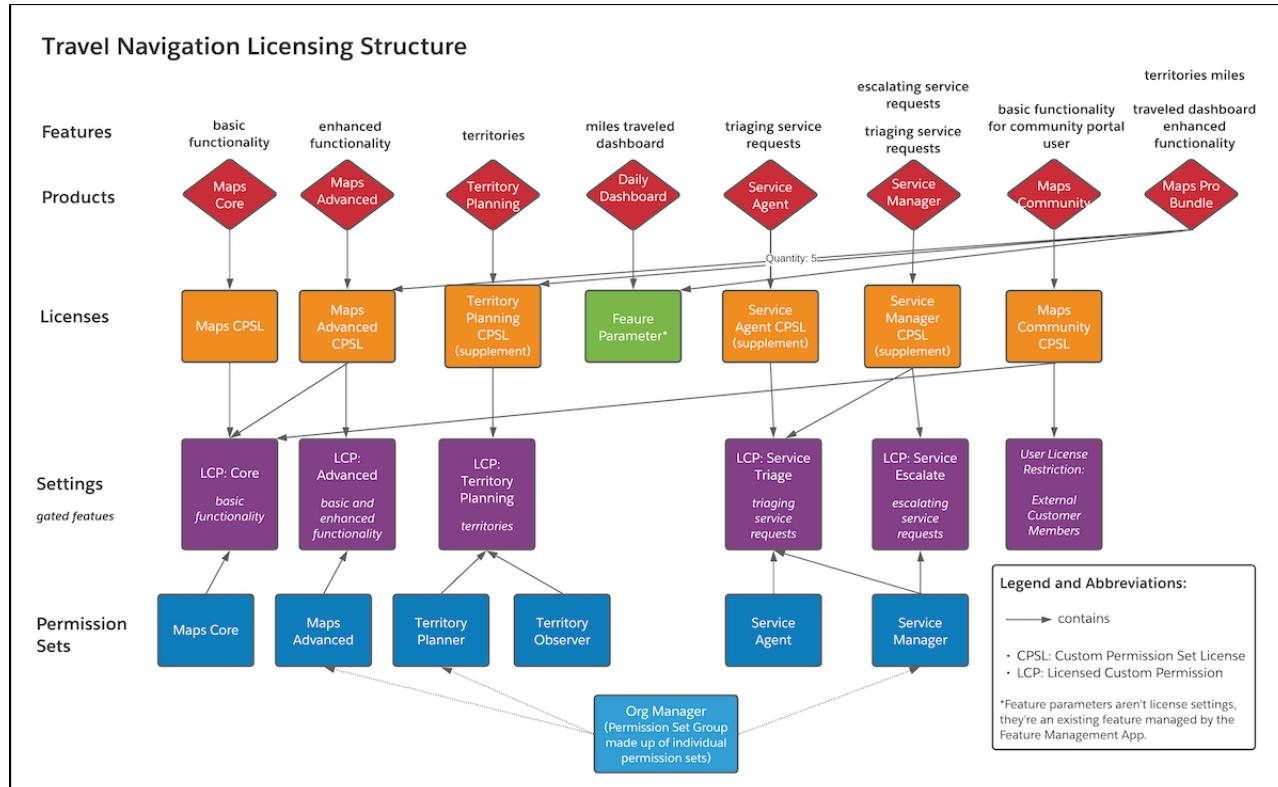
## Review Your Structure (Developer Preview)

Review your design to ensure that your licensing and permissions are structured as you need for your feature pricing and licensing strategy.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

What a journey! By this point, you've designed your whole feature pricing and licensing structure. You first designed products, licenses, and licensed settings. Next, you mapped settings to licenses, and finally designed the permission sets for your personas.

Here's the final result of this exercise for the Travel Navigation example again.



Before you implement your design, we encourage you to review it thoroughly. While feedback from the implementation and testing phases can inform your design via iteration, it's harder to restructure your design later on.

During the design process, it's common to think of new scenarios and change your design. Make sure to think through various use cases for your customers. Also consider potentially faulty licensing structures, such as over-permissioning or under-permissioning, which can allow your subscriber to get more access without paying or receive less access than they expect. After your implementation phase, you'll want to establish a thorough test plan to ensure that everything works as expected.

In this design guide, we walked you through a "top-down" approach, starting from main features and overall pricing strategy, and working through to the individual components. This exercise can also be performed the other way around, or "bottom-up": starting with the individual license settings and mapping them through permission sets, licenses, products, and then your overall pricing strategy. It may be helpful to think through your design this way to surface additional considerations.

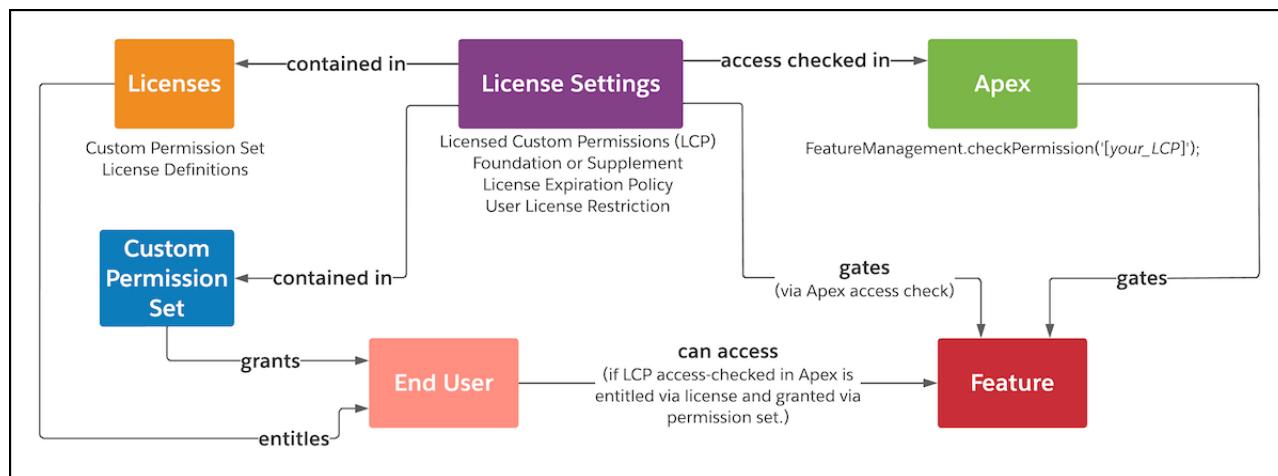
## Implement Your Package Licensing (Developer Preview)

After completing your licensing and permissioning design, you're ready to create your licensing components using point-and-click tools and Apex.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

In addition to this guidance, we prepared a [demo application](#) that you can reference as you implement your package licensing, or simply use to deploy and test the new functionality.

Create your licensing components in a development scratch org created from your new Dev Hub that is enabled with the Partner Licensing Platform. To enable the Partner Licensing Platform in this org, make sure that you have the `PartnerLicensingPlatform` feature in your scratch org definition file.



You first create licensed custom permissions in your development scratch org, then put these custom permissions into custom permission set license definitions, which entitle the custom permissions. Next, you put the custom permissions into custom permission sets, which grant the custom permissions. Finally, you put access checks for the licensed custom permissions in your Apex code for specific features. The access checks gate access to the features. From there, you're ready to deploy and create your new package.

[Create Licensed Custom Permissions \(Developer Preview\)](#)

Licensed custom permissions gate access to your package features.

[Define Custom Permission Set Licenses \(Developer Preview\)](#)

The custom permission set license contains your package's licensed custom permissions and provides the entitlement for the features gated by those settings.

[Create Custom Permission Sets \(Developer Preview\)](#)

The permission set provides the grant to access the package features already entitled by a license.

[Create Custom Permission Set Groups \(Developer Preview\)](#)

Permission set groups bundle permission sets together based on user job functions. Create permission set groups if your licensing structure requires a persona to have permissions from multiple custom permission set licenses.

[Create Access Checks in Apex \(Developer Preview\)](#)

Create access checks for your licensed custom permissions in your Apex code to gate access to your features.

[Deploy Your Licensing Structure and Create Your Package \(Developer Preview\)](#)

After you implement your new licenses and settings, you're ready to create a new package with this metadata.

## Create Licensed Custom Permissions (Developer Preview)

Licensed custom permissions gate access to your package features.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Create your licensed custom permissions in a scratch org created from your new Dev Hub.

1. From Setup, in the Quick Find Box, enter *Custom Permissions*.
2. Select **Custom Permissions**.
3. Click **New**.
4. Enter a label. The Name field is generated automatically.
5. Enter a description.
6. Select **License Required**. This setting indicates that this permission can be granted to a user only if that user also has a custom permission set license containing this licensed custom permission.



**Note:** After a custom permission is packaged and released, you can't edit this setting.

7. Click **Save**.

### SEE ALSO:

[Salesforce Help: Custom Permissions](#)

[Partner Licensing Platform Components and Concepts \(Developer Preview\)](#)

[Identify Licenses \(Developer Preview\)](#)

## Define Custom Permission Set Licenses (Developer Preview)

The custom permission set license contains your package's licensed custom permissions and provides the entitlement for the features gated by those settings.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Create your custom permission set license definitions in a scratch org created from your new Dev Hub. Make sure to review the [Licensing Components](#) topic before creating your license definitions.



**Note:** Custom permission set licenses are provisioned and assignable license seats on the subscriber side. On the developer side, they're called permission set license **definitions**, as they're the package metadata, not the actual seats of the license.

1. In your scratch org, from Setup, in the Quick Find Box, enter *Custom Permission Set License Definitions*.
2. Select **Custom Permission Set Licenses Definitions**.
3. Click **New**.
4. Enter a label. The Name field is generated automatically.
5. Enter a description.
6. For License Expiration Policy, indicate whether access to the package is blocked for assigned users when all custom permission set licenses expire.
7. If this license is a supplement license, select **Supplement license**.



**Note:** This setting can't be edited after the license is created. Supplement licenses don't give access to the package's foundation features, including the package namespace.

8. Select which users can be assigned this permission set license using the user license restriction categories. By default, when no categories are selected, all user licenses can be assigned the permission set license. After you select one user license category, only users with licenses contained in the selected categories can be assigned the permission set license.



**Note:** For which user licenses are included in each category, see [User License Restriction Categories \(Developer Preview\)](#).

9. Click **Save**.
10. Under Licensed Custom Permissions, click **Add Licensed Custom Permission**.
11. Select the permission that [you created previously](#) and click **Save**.



**Note:** You can edit the permissions that you include in your license and upload a new version. However, if you remove permissions, this change only affects new customers. Existing customers retain the same entitlements as before.

[User License Restriction Categories \(Developer Preview\)](#)

Review which user licenses are included in each user license restriction category. These categories are used to specify which users can be assigned your custom permission set licenses.

## SEE ALSO:

- [Partner Licensing Platform Components and Concepts \(Developer Preview\)](#)
- [Identify Licenses \(Developer Preview\)](#)
- [Entitlements and Grants \(Developer Preview\)](#)

## User License Restriction Categories (Developer Preview)

Review which user licenses are included in each user license restriction category. These categories are used to specify which users can be assigned your custom permission set licenses.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.



**Note:** Some of these licenses are supported but are no longer available for purchase.

Category	Category API Value	User Licenses
All Internal Users	\${internal}	<ul style="list-style-type: none"> <li>● Analytics Cloud Integration User</li> <li>● B2BMA Integration User</li> <li>● Chatter External</li> <li>● Chatter Free</li> <li>● Chatter Only</li> <li>● Company Communities</li> <li>● Content Only</li> <li>● CPQ Integration User</li> <li>● Cross-Org Usages</li> <li>● Cross Org Data Proxy</li> <li>● Database.com Admin</li> <li>● Database.com Light</li> <li>● Database.com User</li> <li>● Developer</li> <li>● Force.com - App Subscription</li> <li>● Force.com - Free</li> <li>● Force.com - One App</li> <li>● Ideas Only</li> <li>● Identity</li> <li>● Knowledge Only</li> <li>● Licensing API</li> </ul>

Category	Category API Value	User Licenses
		<ul style="list-style-type: none"> <li>● Partner App Subscription</li> <li>● Partner Network</li> <li>● Premier Support</li> <li>● Sales Insights Integration User</li> <li>● Salesforce</li> <li>● Salesforce Limited Access - Free</li> <li>● Salesforce Platform</li> <li>● Salesforce Platform Light</li> <li>● Salesforce Platform One</li> <li>● SalesforcelQ Integration User</li> <li>● Service Cloud</li> <li>● Service User CMT Query Metrics</li> <li>● Service User Deep Sea Token Access</li> <li>● Service User DI Cross Org Access</li> <li>● Service User EDW Bulk Query</li> <li>● Service User GPU CMF Integration</li> <li>● Siteforce Only</li> <li>● Work.Com Only</li> </ul>
Internal Platform Users	\${platform}	<ul style="list-style-type: none"> <li>● Chatter Only</li> <li>● Company Communities</li> <li>● Force.com - App Subscription</li> <li>● Force.com - Free</li> <li>● Force.com - One App</li> <li>● Identity</li> <li>● Partner App Subscription</li> <li>● Salesforce Platform</li> <li>● Salesforce Platform Light</li> <li>● Salesforce Platform One</li> <li>● Siteforce Only</li> </ul>
All External Customer and Partner Members	\${communities}	<ul style="list-style-type: none"> <li>● Authenticated Website</li> <li>● Bronze Partner</li> <li>● Channel Account</li> <li>● Chatter External</li> <li>● Company Communities</li> <li>● Customer Community</li> <li>● Customer Community Plus</li> <li>● Customer Portal Manager</li> </ul>

Category	Category API Value	User Licenses
		<ul style="list-style-type: none"> <li>• Customer Portal Manager Custom</li> <li>• Customer Portal Manager Standard</li> <li>• Customer Portal Manager User</li> <li>• External Apps</li> <li>• External Apps Plus</li> <li>• External Identity</li> <li>• External Who</li> <li>• Gold Partner</li> <li>• Guest</li> <li>• Guest License</li> <li>• High Volume Customer Portal</li> <li>• High Volume Portal</li> <li>• Ideas Only Site</li> <li>• Internal Portal User</li> <li>• Partner</li> <li>• Partner Community</li> <li>• Silver Partner</li> <li>• Standard Partner</li> </ul>
All External Customer and Partner Logins	\${ communitiesLogin }	<ul style="list-style-type: none"> <li>• Customer Community Login</li> <li>• Customer Community Plus Login</li> <li>• External Apps Login</li> <li>• External Apps Plus Login</li> <li>• IdeasOnlyPortal</li> <li>• Limited Customer Portal Manager Custom</li> <li>• Limited Customer Portal Manager Standard</li> <li>• Overage Authenticated Website</li> <li>• Overage Customer Portal Manager Custom</li> <li>• Overage Customer Portal Manager Standard</li> <li>• Overage High Volume Customer Portal</li> <li>• Partner Community Login</li> </ul>
External Customer Members	\${ customerCommunities }	<ul style="list-style-type: none"> <li>• Authenticated Website</li> <li>• Chatter External</li> <li>• Company Communities</li> </ul>

Category	Category API Value	User Licenses
		<ul style="list-style-type: none"> <li>• Customer Community</li> <li>• Customer Community Plus</li> <li>• Customer Portal Manager</li> <li>• Customer Portal Manager Custom</li> <li>• Customer Portal Manager Standard</li> <li>• Customer Portal Manager User</li> <li>• External Apps</li> <li>• External Who</li> <li>• External Identity</li> <li>• Guest</li> <li>• Guest License</li> <li>• High Volume Customer Portal</li> <li>• High Volume Portal</li> <li>• Ideas Only Site</li> <li>• Internal Portal User</li> </ul>
External Customer Logins	<code>\$(customerCommunitiesLogin)</code>	<ul style="list-style-type: none"> <li>• Customer Community Login</li> <li>• Customer Community Plus Login</li> <li>• External Apps Login</li> <li>• IdeasOnlyPortal</li> <li>• Limited Customer Portal Manager Custom</li> <li>• Limited Customer Portal Manager Standard</li> <li>• Overage Authenticated Website</li> <li>• Overage Customer Portal Manager Custom</li> <li>• Overage Customer Portal Manager Standard</li> <li>• Overage High Volume Customer Portal</li> </ul>
External Partner Members	<code>\$(partnerCommunity)</code>	<ul style="list-style-type: none"> <li>• Bronze Partner</li> <li>• Channel Account</li> <li>• External Apps Plus</li> <li>• Gold Partner</li> <li>• Partner</li> <li>• Partner Community</li> <li>• Silver Partner</li> <li>• Standard Partner</li> </ul>

Category	Category API Value	User Licenses
External Partner Logins	<code>\$(partnerCommunityLogin}</code>	<ul style="list-style-type: none"> <li>External Apps Plus Login</li> <li>Partner Community Login</li> </ul>

## SEE ALSO:

- [Salesforce Help: User Licenses](#)  
[Licensing Components \(Developer Preview\)](#)

## Create Custom Permission Sets (Developer Preview)

The permission set provides the grant to access the package features already entitled by a license.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Create your permission sets in a scratch org created from your new Dev Hub.

1. In your scratch org, from Setup, in the Quick Find Box, enter *Permission Sets*.
2. Select **Permission Sets**.
3. Click **New**.
4. Enter a label. The API Name field is generated automatically.
5. Enter a description.
6. For Select the type of users who will use this permission set, select **None**.

 **Note:** Packaging permission sets constrained to custom permission set licenses isn't currently supported.

7. Select the permissions to include in your permission set. When selecting the permissions, include your licensed custom permission.

## SEE ALSO:

- [Salesforce Help: Permission Sets](#)  
[Partner Licensing Platform Components and Concepts \(Developer Preview\)](#)  
[Design Permission Sets \(Developer Preview\)](#)

## Create Custom Permission Set Groups (Developer Preview)

Permission set groups bundle permission sets together based on user job functions. Create permission set groups if your licensing structure requires a persona to have permissions from multiple custom permission set licenses.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Create your permission set groups in a scratch org created from your new Dev Hub.

1. In your scratch org, from Setup, in the Quick Find Box, enter *Permission Set Groups*.
2. Select **Permission Set Groups**.
3. Click **New**.
4. Enter a label. The API Name field is generated automatically.
5. Enter a description.
6. In the Permission Sets section, click **Permission Sets in Group**.
7. Click **Add Permission Set**.
8. Add all the permission sets related to the custom permission set license.

#### SEE ALSO:

[Salesforce Help: Permission Set Groups](#)

[Partner Licensing Platform Components and Concepts \(Developer Preview\)](#)

[Design Permission Sets \(Developer Preview\)](#)

## Create Access Checks in Apex (Developer Preview)

Create access checks for your licensed custom permissions in your Apex code to gate access to your features.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Create your access checks in a scratch org created from your new Dev Hub.

Use the [FeatureManagementClass Apex class](#) to gate your feature via the licensed custom permission. For example:

```
Boolean userHasCustomPermission =
FeatureManagement.checkPermission('{Licensed_Custom_Permission}');

if (userHasCustomPermission) {
 // your gated feature
}
```

For a more robust mechanism for doing licensing access checks, check out the [LicensingUtil](#) class in our demo application.



**Note:** Your new licensing can only be tested via an installed package, as a subscriber of your package. Testing your new licensing directly in a developer scratch org with deployed unmanaged metadata and code isn't currently supported, since assignable and provisioned custom permission set license seats aren't available in the developer org. For testing in the developer preview, the subscriber org where the managed package is installed is provisioned a default count of 10 seats for each custom permission set license in the package.

For this reason, the `LicensingUtil` class in our demo app bypasses the access check in certain developer orgs, while maintaining the access check in a subscriber context where the package is installed. This configuration allows all users to access the licensed functionality in those developer contexts as well as in automated Apex tests.

#### SEE ALSO:

- [Partner Licensing Platform Components and Concepts \(Developer Preview\)](#)
- [Entitlements and Grants \(Developer Preview\)](#)

## Deploy Your Licensing Structure and Create Your Package (Developer Preview)

After you implement your new licenses and settings, you're ready to create a new package with this metadata.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

1. When creating your new package, make sure to include all necessary components, including your licensed custom permissions, custom permission set license definitions, permission sets, and permission set groups.
2. For 1GP, [deploy the metadata to your packaging org](#) and [create a new managed package](#).
3. For 2GP, add the metadata source to your package directory's source and [create a new managed package using SFDX](#).

## Test Your Licensing Structure (Developer Preview)

---

After creating your new package, you can test your new licensing structure end-to-end.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Test your new licensing structure in a testing scratch org created from your new Dev Hub that is enabled with the Partner Licensing Platform. To enable the Partner Licensing Platform in this org, make sure that you have the `PartnerLicensingPlatform` feature in your scratch org definition file.

In this scratch org, install your newly created package to begin testing. Your new licensing can only be tested via an installed package, as a subscriber of your package. Testing your new licensing directly in a developer scratch org with deployed unmanaged metadata and code isn't currently supported, since assignable and provisioned custom permission set license seats aren't available in the developer org. For testing in the developer preview, the subscriber org where the managed package is installed is provisioned a default count of 10 seats for each custom permission set license in the package.

#### [Best Practices for Testing Your Package Licensing \(Developer Preview\)](#)

Verify your new licensing structure and whether your existing package works as expected with the new components.

[Example: Test Plan for Validating License Design \(Developer Preview\)](#)

Review this sample licensing test plan. You can adapt it to fit your package's needs.

**SEE ALSO:**

[Salesforce Architects Blog: Find Bugs Earlier with Second-Generation Packaging](#)

## Best Practices for Testing Your Package Licensing (Developer Preview)

Verify your new licensing structure and whether your existing package works as expected with the new components.

-  **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

At minimum, we recommend that you test these categories.

- Perform regression tests to make sure product functionality is working as before.
- Manually validate your licensing structure and ensure your new licenses and permission sets are authorizing access to features appropriately.
  - Check for under-permissioning, or if users have less access than expected when assigned permission sets and custom permission set licenses.
  - Check for over-permissioning, or if users have more access than expected when assigned permission sets and custom permission set licenses.
- Make sure that users can only be assigned the intended custom permission set licenses as specified using user license restriction categories. For example, users with licenses in the External Customer Logins category user can only be assigned the license meant for them.

Since testing your new licensing directly in a developer scratch org isn't currently supported, the [LicensingUtil](#) class in our demo app bypasses the access check in certain developer orgs, while maintaining the access check in a subscriber context where the package is installed. This configuration allows all users to access the licensed functionality in those developer contexts.

Your current automated Apex tests may begin to fail with the new licensing access checks and may need refactoring. Currently, automated testing of licensed functionality via license and permission set assignment to a user isn't supported, and may only work on the subscriber side where the package is installed. You may need to bypass your new access checks during automated testing, as shown in the [LicensingUtil](#) class example.

-  **Note:** Scratch orgs always operate with "site-wide" package licensing. As a result, all users in your subscriber scratch org are entitled to your foundation package features, such as Visualforce pages, without a managed package license or a custom (foundation) permission set license. Real subscribers of your package will need either a managed package license or custom permission set license to access your package's foundation features.

## Example: Test Plan for Validating License Design (Developer Preview)

Review this sample licensing test plan. You can adapt it to fit your package's needs.

-  **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Our [forward-looking statement](#) applies to custom supplement permission set licenses. Make your purchasing decisions based on currently available technology.

This test plan continues the Travel Navigation example from the [Design Your Package Licensing Structure](#) section of this guide.

## Package Access

Testing Scenario	Expected Behavior
User is assigned Territory Planning CPSL	User can't access Travel Navigation package
User is assigned Maps CPSL	User can access Travel Navigation package
User is assigned Maps Advanced CPSL	User can access Travel Navigation package
User is assigned Territory Planning CPSL and Maps CPSL	User can access Travel Navigation package

## Feature Access

Testing Scenario	Expected Behavior
User is assigned: <ul style="list-style-type: none"> <li>• Maps CPSL</li> <li>• Territory Planning CPSL</li> <li>• Service Agent CPSL</li> <li>• Service Manager CPSL</li> <li>• Org Manager permission set group</li> </ul>	User can access: <ul style="list-style-type: none"> <li>• Travel Navigation package</li> <li>• Territory planning</li> <li>• Triaging service requests</li> <li>• Escalating service requests</li> </ul>
User is assigned: <ul style="list-style-type: none"> <li>• Maps CPSL</li> <li>• Territory Planning CPSL</li> <li>• Service Agent CPSL</li> <li>• Service Manager CPSL</li> </ul> User isn't assigned: <ul style="list-style-type: none"> <li>• Org Manager permission set group</li> </ul>	User can access: <ul style="list-style-type: none"> <li>• Travel Navigation package</li> </ul> User can't access: <ul style="list-style-type: none"> <li>• Territory planning</li> <li>• Triaging service requests</li> <li>• Escalating service requests</li> </ul>
User is assigned: <ul style="list-style-type: none"> <li>• Maps CPSL</li> <li>• Territory Planning CPSL</li> <li>• Territory Planner permission set</li> </ul>	User can access: <ul style="list-style-type: none"> <li>• Travel Navigation package</li> <li>• Territory planning</li> </ul> User can't access: <ul style="list-style-type: none"> <li>• Triaging service requests</li> <li>• Escalating service requests</li> </ul>

Testing Scenario	Expected Behavior
User is assigned: <ul style="list-style-type: none"> <li>• Maps CPSL</li> <li>• Territory Planning CPSL</li> </ul> User isn't assigned: <ul style="list-style-type: none"> <li>• Territory Planner permission set</li> </ul>	User can access: <ul style="list-style-type: none"> <li>• Travel Navigation package</li> </ul> User can't access: <ul style="list-style-type: none"> <li>• Territory planning</li> <li>• Triaging service requests</li> <li>• Escalating service requests</li> </ul>
User is assigned: <ul style="list-style-type: none"> <li>• Maps CPSL</li> <li>• Territory Planner permission set</li> </ul> User isn't assigned: <ul style="list-style-type: none"> <li>• Territory Planning CPSL</li> </ul>	Territory Planner permission set assignment fails, because Territory Planning CPSL isn't assigned. User can access: <ul style="list-style-type: none"> <li>• Travel Navigation package</li> </ul>

## Permission Set License Assignments

Testing Scenario	Expected Behavior
Internal user is assigned Maps Community PSL	Error when assigning permission set license
External user is assigned Maps CPSL	Error when assigning permission set license

## Manage Migrations to the Partner Licensing Platform (Developer Preview)

If you have existing customers for your product, you can take steps to ensure that both your new and existing license settings work while you transition to your new licensing design.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

When you're ready, communicate the changes to your licensing structure and your timeline for migrating to the new licensing design to your existing customers.

### [Manage a Hybrid Licensing Model During Migrations \(Developer Preview\)](#)

Set up your access checks so that your new custom permission set licenses work alongside your existing managed package licenses, and both new and existing customers can use your features without disruption.

### [Licensing for Existing Custom Permissions \(Developer Preview\)](#)

Set up your existing custom permissions and new licensed custom permissions so that both new and existing customers can access your features as you migrate to your new licensing design.

[Package Access Validation in a Hybrid Licensing Model \(Developer Preview\)](#)

After a subscriber's org is provisioned custom permission set licenses, going forward, package access for the namespace is validated whenever permission sets are assigned to users.

## Manage a Hybrid Licensing Model During Migrations (Developer Preview)

Set up your access checks so that your new custom permission set licenses work alongside your existing managed package licenses, and both new and existing customers can use your features without disruption.

 **Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

New customers will use your new licensing structure and settings from the beginning. However, if you upgrade your package code with your new access checks and settings, this change can cause your existing customers to lose access to those newly gated existing features, because users may not have the necessary licenses and permissions assigned for your new license settings at the time of upgrade. By setting up conditional behavior, both new and existing customers can access features without disruption.

Wherever you have access checks for your new licensed custom permissions, add an OR condition that also checks whether the user has the original managed package license for the package. Use the [hasPackageLicense](#) method, which returns true if the context user has a license to the managed package via a package license only.

For example:

```
Boolean hasCustomPermission =
FeatureManagement.checkPermission('{Licensed_Custom_Permission}');
if (hasCustomPermission ||
UserInfo.hasPackageLicense([Your_Package_Id]) {
//your gated feature}
```

 **Note:** For a more robust mechanism for doing licensing access checks, including this hybrid licensing model, check out the [LicensingUtil](#) class in our demo application. However, because scratch orgs always operate with "site-wide" package licensing, the [hasPackageLicense](#) method always returns `true` in this context. Currently, this hybrid licensing access check will only be applicable for real subscribers of your package, who will need either a managed package license or custom (foundation) permission set license to access your package.

To break down how both new and existing customers interact with this access check:

- All new users or customers start with your new licenses. If they have the appropriate licensed custom permission, they pass the access check.
- All existing customers' users with managed package licenses retain their original access by passing the `hasPackageLicense` access check.
- As users migrate onto your new licenses, their original managed package licenses should also be removed. At this point, users can only access the feature if they have the appropriate licensed custom permission.

With this approach, your customers have time to understand and migrate onto your new licenses without losing access.

## Licensing for Existing Custom Permissions (Developer Preview)

Set up your existing custom permissions and new licensed custom permissions so that both new and existing customers can access your features as you migrate to your new licensing design.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

After a licensed custom permission is packaged and released, you can't edit the **License Required** attribute, which means that existing custom permissions can't be converted to a licensed custom permission. This behavior is because users who already have the custom permission may not have the necessary license assigned when the custom permission becomes licensed at the time of upgrade.

Because you can't directly convert existing custom permissions to licensed ones, set up your access checks to support both existing custom permissions and new licensed custom permissions. First, create your licensed custom permission. Then, locate in your access checks where you currently check for the original custom permission. Instead of replacing the original custom permission with your new licensed custom permission, put an **OR** clause between the two custom permissions. As a result, existing customers and users can continue to get uninterrupted access to the gated feature, although without a license. New customers and users get access to the gated feature via the new licensed custom permission and new license.

After you communicate the change and migration expectation to your existing customers and give them time to adopt the new licensed custom permission, you can remove the old custom permission from your Apex access checks to enforce the migration and license assignment.

## Package Access Validation in a Hybrid Licensing Model (Developer Preview)

After a subscriber's org is provisioned custom permission set licenses, going forward, package access for the namespace is validated whenever permission sets are assigned to users.



**Note:** The Partner Licensing Platform is available as a developer preview. The Partner Licensing Platform isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. All commands, parameters, and other features are subject to change or deprecation at any time, with or without notice. Don't implement functionality developed with these commands or tools in your production package.

Our [forward-looking statement](#) applies to partner business org provisioning. Make your purchasing decisions based on currently available technology.

When an admin assigns a user a permission set that grants access to package components, the system first validates whether the user has a custom foundation permission set license, or managed package license, that allows access to the package namespace. Previously, package access wasn't validated until the user attempted to access the components. This delay can cause an inconsistent authorization experience, because admins can assign users permission sets that grant access to components in a package, while those users may not have access to the package itself via a license.

For your existing customers, users with managed package licenses can continue to access package components via a permission set assignment. However, for the users who were assigned permission sets that grant access to package components—but who don't have a managed package license or custom foundation permission set license assigned—this validation will now trigger an error. This behavior makes for a consistent authorization experience across both Salesforce and ISV products and licenses. The subscriber admin must either remove the permission set assignment, assign the user a custom permission set license, or remove access to the managed package components from the permission set.

# CHAPTER 15 Unlocked Packages

## In this chapter ...

- [What's a Package](#)
- [Package-Based Development Model](#)
- [Before You Create Unlocked Packages](#)
- [Know Your Orgs](#)
- [Create Org-Dependent Unlocked Packages](#)
- [Workflow for Unlocked Packages](#)
- [Configure Packages](#)
- [How We Handle Profile Settings in Unlocked Packages](#)
- [Create a Package](#)
- [Push a Package Upgrade](#)
- [Install a Package](#)
- [Migrate Deprecated Metadata from Unlocked Packages](#)
- [Uninstall a Package](#)
- [Transfer an Unlocked Package to a Different Dev Hub](#)

Salesforce offers different types of packages, and unlocked packages are especially suited for internal business apps. Unless you plan to distribute an app on AppExchange, an unlocked package is the right package type for most use cases. You can use unlocked packages to organize your existing metadata, package an app, extend an app that you've purchased from AppExchange, or package new metadata. Unlocked packages follow a source-driven development model. The source of truth of the metadata contained in the package is your version control system, not what's in an org. This model brings with it all the benefits of modern source-driven development models.



**Note:** If you're an AppExchange partner that plans to distribute your app to customers via AppExchange, second-generation managed packaging is the preferred tool. See [Second-Generation Managed Packages](#) for more information.

## What's a Package

If you're new to packaging, you can think about a package as a container that you fill with metadata. It contains a set of related features, customizations, and schema. You use packages to move metadata from one Salesforce org to another.

Each unlocked package has a distinct life cycle. You add metadata to a package, and create a new package version. While the package is continually evolving, each package version is an immutable artifact.

A package version contains the specific metadata and features associated with the package version, at the time it was created. As you iterate on your package, and add, remove, or change the packaged metadata, you create many package versions.

You can install a package version in a scratch, sandbox, trial, developer edition, or production org. Installing a package version is similar to deploying metadata. Each package version has a version number, and subscribers can install a new package version into their org through a package upgrade.



**Note:** Since package versions are immutable, they can also be used as artifacts for Continuous Integration (CI) and Continuous Delivery (CD) processes.

You can repeat the package development cycle any number of times. You can change metadata, create a package version, test the package version, and finally deploy or install the package to a production org. This distinct app development lifecycle lets you control exactly what, when and how your metadata is rolled out. In the installed org, you can inspect which metadata came from which package and the set of all metadata associated with a specific package.

## Package-Based Development Model

To demonstrate the power of unlocked packages, here's how packaging works in the traditional development model. For most production orgs, metadata traditionally is contained in two buckets: a set of managed packages installed from AppExchange, and unpackaged metadata.

Customers often invest in Salesforce customizations to support business processes and extend the power of the Salesforce platform. In the development model, your Salesforce org's monolith of unpackaged metadata contains all the metadata that belongs to a custom app or extension. Because that metadata isn't isolated or organized, it can be difficult to understand, upgrade, and maintain.

In the package development model, you can organize your unpackaged metadata in your production org into well-defined packages. And you can use Salesforce DX projects to organize your source into package directories with everything managed in a version control system of your choice. Your end goal is to create packages using those directories that are versionable, easy to maintain, update, install, and upgrade.

Unlocked packages allow you to declare multi-level dependencies on one or many managed and unlocked packages, which keeps your packages small and modular. You can use the command line to execute unlocked packaging operations, or you can include packaging-specific Salesforce CLI commands in a script and automate your package development.

## Before You Create Unlocked Packages

When you use unlocked packaging, to be sure that you set it up correctly, verify the following.

Did you?

- [Enable Dev Hub in Your Org](#)
- [Enable Second-Generation Managed Packaging](#)
- Install [Salesforce CLI](#)



**Note:** Unlocked packaging is available with these licenses: Salesforce or Salesforce Limited Access - Free (partners only).

Developers who work with unlocked packages need the correct permission set in the Dev Hub org. Developers need either the System Administrator profile or the Create and Update Second-Generation Packages permission. For more information, see [Add Salesforce DX Users](#).

The maximum number of unlocked package versions that you can create from a Dev Hub per day is the same as your daily scratch org allocation. To request a limit increase, contact Salesforce Customer Support.

Scratch orgs and packages count separately, so creating an unlocked package doesn't count against your daily scratch org limit. To view your scratch org limits, use the CLI:

```
sfdx force:limits:api:display -u <Dev Hub username or alias>
```

For more information on scratch org limits, see [Scratch Orgs](#).

## Know Your Orgs

---

Some of the orgs that you use with unlocked packaging have a unique purpose.

### Choose Your Dev Hub Org

Use the Dev Hub org for these purposes.

- As owner of all unlocked packages
- To link your namespaces if you want to create namespaced unlocked packages
- To authorize and run your `force:package` commands

When you create an unlocked package using Salesforce CLI, you associate the package with a specific Dev Hub org. The Dev Hub org owns the package, and you can't transfer package ownership from one Dev Hub org to another. When you're ready to define and create a package for production use, be sure to create the package using the Dev Hub in one of your production orgs.

### Namespace Org

If you are using a namespace, you'll need a namespace org to acquire a package namespace. If you want to use the namespace strictly for testing, choose a disposable namespace.

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

### Other Orgs

When you work with packages, you also use these orgs:

- You can create scratch orgs on the fly to use while testing your packages.
- The target or installation org is where you install the package.

## Create Org-Dependent Unlocked Packages

Org-dependent unlocked packages are a variation of unlocked packages that allow you to create packages that depend on unpackaged metadata in the org where you plan to install the package (installation org).

Untangling your production org metadata can be a daunting project. But now you have a solution that enables you to package metadata without completely accounting for all metadata dependencies: org-dependent unlocked packages. When you use org-dependent unlocked packages, metadata validation occurs during package installation, instead of during package version creation.

Longstanding and large production orgs often accumulate large amounts of metadata that are difficult to modularize when adopting a package-based Application Lifecycle Management (ALM) approach. Instead, you can package metadata that depends on unpackaged metadata in the installation org.

 **Note:** Org-dependent unlocked packages are a variation of unlocked packages, and not a separate package type. They follow the same [package development steps](#), and use the same supported [metadata types](#) as unlocked packages.

To create an org-dependent unlocked package, specify the `--orgdependent` CLI parameter on the `package:create` CLI command.

```
sfdx force:package:create -t Unlocked -r force-app -n MyPackage --orgdependent
```

### USER PERMISSIONS

To create packages:

- Create and Update Second-Generation Packages

Scenario	Unlocked Packages	Org Dependent Unlocked Packages
Build once, install anywhere	Yes	No. These packages are designed for specific production and sandbox orgs. You can install them only in orgs that contain the metadata that the package depends on.
Dependency validation	Occurs during package version creation	Occurs during package installation
Can depend on other packages	Yes	No
Requires dependencies to be resolved to create the package	Yes	No
Supported metadata types	See the unlocked packaging channel of the <a href="#">Metadata Coverage Report</a> .	See the unlocked packaging channel of the <a href="#">Metadata Coverage Report</a> .
Recommended development and testing environment	Use scratch orgs to develop and test your unlocked packages.	Use a sandbox that contains the dependent metadata. Consider enabling <a href="#">Source Tracking in Sandboxes</a> to develop your org-dependent unlocked package. Then, test the package in a sandbox org before installing it in your production org.
Code coverage requirement	Before you can promote and release an unlocked package, the Apex code must meet a minimum 75% code coverage requirement.	We don't calculate code coverage, but we recommend that you ensure the Apex code in your package is well tested.

To review which of your packages are org-dependent unlocked packages, use `sfdx force:package:list --verbose`.

## Workflow for Unlocked Packages

You can create and install an unlocked package directly from the Salesforce command line.

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

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

1. Create a DX project.

```
sfdx force:project:create --outputdir expense-manager-workspace --projectname expenser-app
```

2. Authorize the Dev Hub org, and create a scratch org.

```
sfdx auth:web:login --setdefaultdevhubusername
```

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

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

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

```
sfdx force:org:create --definitionfile config/project-scratch-def.json --targetusername MyScratchOrg1
```

4. Verify that all package components are in the project directory where you want to create a package.

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

```
sfdx force:package:create --name "Expense Manager" --path force-app
--packagetype Unlocked
```

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

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

Notice the placeholder values for `versionName` and `versionNumber`.

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

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

```
sfdx force:package:version:create --package "Expense Manager" --installationkey test1234
--wait 10
```

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

```
sfdx force:package:install --package "Expense Manager@0.1.0-1" --targetusername MyTestOrg1
--installationkey test1234 --wait 10 --publishwait 10
```

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

```
sfdx force:org:open --targetusername MyTestOrg1
```

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

## Configure Packages

You include an entry in the `sfdx-project.json` file for each package to specify its alias, version details, dependencies, features, and org settings. From the command line, you can also set or change options, such as specifying an installation key, update the package name, or add a description.

### [Project Configuration File for Unlocked Packages](#)

The project configuration file is a blueprint for your project. The settings in the file create an outline of your package and determine the package attributes and package contents.

### [Unlocked Packaging Keywords](#)

A keyword is a variable that you can use to specify a package version number.

### [Package Installation Key](#)

To ensure the security of the metadata in your package, you must specify an installation key when creating a package version. Package creators provide the key to authorized subscribers so they can install the package. Package installers provide the key during installation, whether installing the package from the CLI or from a browser. An installation key is the first step during installation. The key ensures that no package information, such as the name or components, is disclosed until the correct installation key is supplied.

### [Extract Dependency Information from Unlocked Packages](#)

For an installed unlocked package, you can now run a simple SOQL query to extract its dependency information. You can also create a script to automate the installation of unlocked packages with dependencies.

### [Understanding Namespaces](#)

A namespace is a 1-15 character alphanumeric identifier that distinguishes your package and its contents from other packages in your org.

### [Share Release Notes and Post-Install Instructions](#)

Share details about what's new and changed in a released unlocked package with your users.

### [Specify Unpackaged Metadata or Apex Access for Apex Tests \(Unlocked Packages\)](#)

### [Best Practices for Unlocked Packages](#)

We suggest that you follow these best practices when working with unlocked packages.

#### [Package IDs and Aliases for Unlocked Packages](#)

During the package lifecycle, packages and package versions are identified by an ID or package alias. When you create a package or package version, Salesforce CLI creates a package alias based on the package name, and stores that name in the `sfdx-project.json` file. When you run CLI commands or write scripts to automate packaging workflows, it's often easier to reference the package alias, instead of the package ID or package version ID.

#### [Frequently Used Packaging Operations](#)

## Project Configuration File for Unlocked Packages

The project configuration file is a blueprint for your project. The settings in the file create an outline of your package and determine the package attributes and package contents.

Here are the parameters you can specify in the project configuration file.

Name	Required?	Default if Not Specified
apexTestAccess	No	<p>None. Assign permission sets and permission set licenses to the user in context when your Apex tests run at package version creation.</p> <pre>"apexTestAccess": {     "permissionSets": [         "Permission_Set_1",         "Permission_Set_2"     ],     "permissionSetLicenses": [         "SalesConsoleUser"     ] }</pre> <p>See <a href="#">Specify Unpackaged Metadata or Apex Access for Apex Tests (Unlocked Packages)</a></p>
branch	No	<p>None. If your package has an associated branch, but your package dependency is associated with a different branch, use this format.</p> <pre>"dependencies": [     {         "package": "pkgB",         "versionNumber":         "1.3.0.LATEST",         "branch": "featureC"     } ]</pre>

Name	Required?	Default if Not Specified
		<p>If your package has an associated branch, but your package dependency doesn't have an associated branch, use this format.</p> <pre>"dependencies": [     {         "package": "pkgB",         "versionNumber": "1.3.0.LATEST",         "branch": ""     } ]</pre> <p>See <a href="#">Use Branches in Unlocked Packaging</a></p>
default	Yes, if you've specified more than one package directory	<p>true</p> <p>Indicates the default package directory. Use the <code>force:source:pull</code> command to copy metadata from your scratch org to your default package directory.</p> <p>There can be only one package directory in which the default is set to true.</p>
definitionFile	No	<p>None. A reference to an external <code>.json</code> file used to specify the features and org settings required for the metadata of your package, such as the scratch org definition.</p> <p>Example:</p> <pre>"definitionFile": "config/project-scratch-def.json",</pre>
dependencies	No	<p>None. Specify the dependencies on other packages.</p> <p>To specify dependencies for unlocked packages within the same Dev Hub, use either the package version alias or a combination of the package name and the version number.</p> <pre>"dependencies": [     {         "package": "MyPackageName@0.1.0.1"     } ]</pre> <pre>"dependencies": [     {         "package": "MyPackageName",         "versionNumber": "0.1.0.LATEST"     } ]</pre>

Name	Required?	Default if Not Specified
		<p>To specify dependencies for unlocked packages outside of the Dev Hub use:</p> <pre>"dependencies": [   {     "package": "OtherOrgPackage@1.2.0"   } ]</pre> <p> <b>Note:</b> You can use the LATEST keyword for the version number to set the dependency.</p> <p>To denote dependencies with package IDs instead of package aliases, use:</p> <ul style="list-style-type: none"> <li>• The <code>0Ho</code> ID if you specify the package ID along with the version number</li> <li>• The <code>04t</code> ID if you specify only the package version ID</li> </ul> <p>If the package has more than one dependency, provide a comma-separated list of packages in the order of installation. For example, if a package depends on the package Expense Manager - Util, which in turn depends on the package External Apex Library, the package dependencies are:</p> <pre>"dependencies": [   {     "package" : "External Apex Library - 1.0.0.4"   },   {     "package": "Expense Manager - Util",     "versionNumber": "4.7.0.LATEST"   } ]</pre> <p>See: <a href="#">Extract Dependency Information from Unlocked Packages</a></p>
includeProfileUserLicenses	No	<p>False. Setting this parameter to <code>true</code> ensures that user licenses associated with profiles in unlocked packages are retained during package version creation. By default, unlocked packages remove profile information not pertinent to the packaged metadata.</p> <pre>"packageDirectories": [   {     "package": "PackageA",     "path": "common",     "versionName": "ver 0.1",     "versionNumber": "0.1.0.NEXT",     "default": false,   } ]</pre>

Name	Required?	Default if Not Specified
		<pre>"includeProfileUserLicenses": true     } ]</pre>
namespace	no	None. A 1–15 character alphanumeric identifier that distinguishes your package and its contents from packages of other developers.
package	Yes	The package name specified in the project.json file.
packageAliases	Yes	Salesforce CLI updates the project file with aliases when you create a package or package version. You can also manually update this section for existing packages or package versions. You can use the alias instead of the cryptic package ID when running CLI <code>force:package</code> commands.
path	Yes	If you don't specify a path, Salesforce CLI uses a placeholder when you create a package.
postInstallUrl	No	None. A URL to post-install instructions for subscribers.
releaseNotesUrl	No	None. A URL to release notes.
seedMetadata	No	<p>None.</p> <p>Specify the path to your seedMetadata directory.</p> <p>Seed metadata is available to standard value sets only. If your package depends on standard value sets, you can specify a seed metadata directory that contains the value sets.</p> <p>Example:</p> <pre>"packageDirectories": [     {         "seedMetadata": {             "path": "my-unpackaged-seed-directory"         }     }, ]</pre>
unpackagedMetadata	No	None. See <a href="#">Specify Unpackaged Metadata or Apex Access for Apex Tests (Unlocked Packages)</a>
versionDescription	No	None.
versionName	No	If not specified, the CLI uses <code>versionNumber</code> as the version name.

Name	Required?	Default if Not Specified
versionNumber	Yes	<p>None. Version numbers are formatted as major.minor.patch.build. For example, 1.2.1.8.</p> <p>To automatically increment the build number to the next available build for the package, use the keyword NEXT (1.2.1.NEXT).</p>

When you specify a parameter using Salesforce CLI, it overrides the value listed in the project definition file.

The Salesforce DX project definition file is a JSON file located in the root directory of your project. Use the `force:project:create` CLI command to generate a project file that you can build upon. Here's how the parameters in `packageDirectories` appear.

```
{
 "namespace": "",
 "sfdcLoginUrl": "https://login.salesforce.com",
 "sourceApiVersion": "47.0",
 "packageDirectories": [
 {
 "path": "util",
 "default": true,
 "package": "Expense Manager - Util",
 "versionName": "Winter '20",
 "versionDescription": "Welcome to Winter 2020 Release of Expense Manager Util Package",
 "versionNumber": "4.7.0.NEXT",
 "definitionFile": "config/scratch-org-def.json"
 },
 {
 "path": "exp-core",
 "default": false,
 "package": "Expense Manager",
 "versionName": "v 3.2",
 "versionDescription": "Winter 2020 Release",
 "versionNumber": "3.2.0.NEXT",
 "postInstallUrl": "https://expenser.com/post-install-instructions.html",
 "releaseNotesUrl": "https://expenser.com/winter-2020-release-notes.html",
 "definitionFile": "config/scratch-org-def.json",
 "dependencies": [
 {
 "package": "Expense Manager - Util",
 "versionNumber": "4.7.0.LATEST"
 },
 {
 "package": "External Apex Library - 1.0.0.4"
 }
]
 }
],
 "packageAliases": {
 "Expense Manager - Util": "0HoB00000004CFpKAM",
 "External Apex Library@1.0.0.4": "04tB0000000IB1EIAW",
 }
}
```

```

 "Expense Manager": "0HoB00000004CFuKAM"
}

```

## What If I Don't Want My Salesforce DX Project Automatically Updated?

In some circumstances, you don't want to have automatic updates to the `sfdx-project.json` file. When you require more control, use these environment variables to suppress automatic updates to the project file.

For This Command	Set This Environment Variable to True
force:package:create	SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_CREATE
force:package:version:create	SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_VERSION_CREATE

## Unlocked Packaging Keywords

A keyword is a variable that you can use to specify a package version number.

You can use keywords to automatically increment the value of the package build numbers, ancestor version numbers, set the package dependency to the latest version, or the latest released and promoted version.

Use the Keyword	Example
LATEST to specify the latest version of the package dependency when you create a package version.	<pre> "dependencies": [   {     "package": "MyPackageName",     "versionNumber": "0.1.0.LATEST"   } ] </pre>
NEXT to increment the build number to the next available for the package version.	<pre> "versionNumber": "1.2.0.NEXT" </pre>
RELEASED to specify the latest promoted and released version of the package dependency when you create a package version.	<pre> "dependencies": [   {     "package": "pkgB",     "versionNumber": "2.1.0.RELEASED"   } ] </pre>
HIGHEST to automatically set the package ancestor to the highest promoted and released package version number.  Use only with ancestor version or ancestor ID.	<pre> "packageDirectories": [   {     "path": "util",     "package": "Expense Manager - Util",     "versionNumber": "4.7.0.NEXT",     "ancestorVersion": HIGHEST   }, ] </pre>

Use the Keyword	Example
<p>NONE in the ancestor version or ancestor ID field.</p> <p>Ancestry defines package upgrade paths. If the package ancestor is set to NONE, an existing customer can't upgrade to that package version.</p>	<pre data-bbox="829 280 1444 492"> "packageDirectories": [ { "path": "util", "package": "Expense Manager - Util", "versionNumber": "4.7.0.NEXT", "ancestorVersion": NONE }, </pre>

## Package Installation Key

To ensure the security of the metadata in your package, you must specify an installation key when creating a package version. Package creators provide the key to authorized subscribers so they can install the package. Package installers provide the key during installation, whether installing the package from the CLI or from a browser. An installation key is the first step during installation. The key ensures that no package information, such as the name or components, is disclosed until the correct installation key is supplied.

To set the installation key, add the `--installationkey` parameter to the command when you create the package version. This command creates a package and protects it with the installation key.

```

sfdx force:package:version:create --package "Expense Manager" --installationkey
"JSB7s8vXU93fI"

```

Supply the installation key when you install the package version in the target org.

```

sfdx force:package:install --package "Expense Manager" --installationkey "JSB7s8vXU93fI"

```

## Change the Installation Key for an Existing Package Version

You can change the installation key for an existing package version with the `force:package:version:update` command.

```

sfdx force:package:version:update --package "Expense Manager@1.2.0-4" --installationkey
"HIF83kS8kS7C"

```

## Create a Package Version Without an Installation Key

If you don't require security measures to protect your package metadata, you can create a package version without an installation key.

```

sfdx force:package:version:create --package "Expense Manager" --directory common \
--tag 'Release 1.0.0' --installationkeybypass

```

## Check Whether a Package Version Requires an Installation Key

To determine whether a package version requires an installation key, use either the `force:package:version:list` or `force:package:version:report` CLI command.

## Extract Dependency Information from Unlocked Packages

For an installed unlocked package, you can now run a simple SOQL query to extract its dependency information. You can also create a script to automate the installation of unlocked packages with dependencies.

The `SubscriberPackageVersion` Tooling API object now provides dependency information. Using a SOQL query on `SubscriberPackageVersion`, you can identify the packages on which your unlocked package has a dependency. You can get the (04t) IDs and the correct install order for those packages.



**Example:** Package B has a dependency on package A. Package D depends on packages B and C. Here's a sample `sfdx-project.json` that you would have specified while creating a package version. Package D dependencies are noted as packages A, B, and C.

```
{
 "packageDirectories": [
 {
 "path": "pkg-a-workspace",
 "package": "pkgA",
 "versionName": "ver 4.9",
 "versionNumber": "4.9.0.NEXT",
 "default": true
 },
 {
 "path": "pkg-b-workspace",
 "package": "pkgB",
 "versionName": "ver 3.17",
 "versionNumber": "3.17.0.NEXT",
 "default": false,
 "dependencies": [
 {
 "package": "pkgA",
 "versionNumber": "3.3.0.LATEST"
 }
]
 },
 {
 "path": "pkg-c-workspace",
 "package": "pkgC",
 "versionName": "ver 2.1",
 "versionNumber": "2.1.0.NEXT",
 "default": false
 },
 {
 "path": "pkg-d-workspace",
 "package": "pkgD",
 "versionName": "ver 1.1",
 "versionNumber": "1.1.0.NEXT",
 "default": false,
 "dependencies": [
 {
 "package": "pkgA",
 "versionNumber": "3.3.0.LATEST"
 },
 {
 "package": "pkgB",
 "versionNumber": "3.12.0.LATEST"
 },
 {
 "package": "pkgC",
 "versionNumber": "2.1.0.LATEST"
 }
]
 }
]
}
```

```

 }
]
}
],
"namespace": "",
"sfdcLoginUrl": "https://login.salesforce.com",
"sourceApiVersion": "44.0",
"packageAliases": {
 "pkgA": "0HoB000000080q6KAE",
 "pkgB": "0HoB000000080qBKAU",
 "pkgC": "0HoB000000080qGKAU",
 "pkgD": "0HoB000000080qGKAQ"
}
}
}

```

Before installing pkgD (with ID=04txx000000082hAAA), use this SOQL query to determine its dependencies. The username is typically the target subscriber org where the unlocked package is to be installed.

```
sfdx force:data:soql:query -u {USERNAME} -t
-q "SELECT Dependencies FROM SubscriberPackageVersion
WHERE Id='04txx000000082hAAA'" --json
```

You see this output when you run the query, with the (04t) IDs for pkgA, pkgB, and pkgC in that order.

```
"Dependencies": {"Ids": [
 {"subscriberPackageVersionId": "04txx000000080vAAA"},
 {"subscriberPackageVersionId": "04txx000000082XAAQ"},
 {"subscriberPackageVersionId": "04txx0000000AiGAAU"}]}
```

## Understanding Namespaces

A namespace is a 1-15 character alphanumeric identifier that distinguishes your package and its contents from other packages in your org.

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

You can choose to create unlocked packages with or without a specific namespace. A namespace is assigned to a package at the time that it's created and can't be changed.

Use No-Namespace Packages If	Use Namespace Packages If
You want to migrate metadata from your org's monolith of unpackaged metadata to unlocked packages. Creating a no-namespace package gives you more control over how you organize and distribute parts of an application.	You're new to packaging and you're adopting packages in several stages. Using a namespace prefix such as <code>Acme__</code> can help you identify what's packaged and what's still unpackaged metadata in your production orgs
You want to retain the API name of previously unpackaged metadata elements.	You have more than one development team. A namespace can ensure your API names don't collide with another team.  In general, working with a single namespace is easier, and you can easily share code across packages that share a namespace.

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

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

- You can develop more than one unlocked package with the same namespace but you can associate each package with only a single namespace.
- If you work with more than one namespace, we recommend that you set up one project for each namespace.

### Create and Register Your Namespace

With unlocked packages, you can share a single namespace with multiple packages. Since sharing of code is much easier if your package shares the same namespace, we recommend that if you use namespaces, you use a single namespace for your namespaced unlocked packages.

#### Avoid Namespace Collisions

Namespaces impact the combination of package types you can install in an org.

#### Namespace-Based Visibility for Apex Classes in Unlocked Packages

The `@namespaceAccessible` makes public Apex in a package available to other packages that use the same namespace. Without this annotation, Apex classes, methods, interfaces, and properties defined in an unlocked package aren't accessible to the other packages with which they share a namespace. Apex that is declared global is always available across all namespaces, and needs no annotation.

## Create and Register Your Namespace

With unlocked packages, you can share a single namespace with multiple packages. Since sharing of code is much easier if your package shares the same namespace, we recommend that if you use namespaces, you use a single namespace for your namespaced unlocked packages.

To create a namespace:

1. Sign up for a new Developer Edition org.
2. In Setup, enter *Package Manager* in the Quick Find box, and select **Package Manager**.
3. In Developer Settings, click **Edit**, and under Change Developer Settings, click **Continue**.
4. In Namespace Prefix enter a namespace, and select **Check Availability**.
5. For **Package to be managed**, select **None**, then click **Review My Selections**.
6. Review your selections, and then click **Save**.

To register a namespace:

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

## Avoid Namespace Collisions

Namespaces impact the combination of package types you can install in an org.

To understand how namespaces affect the types of packages you can install in a namespaced or no-namespace org, review this table.

Installation Org	No-namespace Unlocked Package	Namespaced Unlocked Package	Second-generation Managed Package (2GP)	First-generation Managed Package (1GP)
<b>Org with a namespace</b>	<p>Fail.</p> <p>For example, a 1GP packaging org, 1GP patch org, Developer Edition org with namespace, or a scratch org with namespace</p>	<p>Pass.</p> <p>You can't install a no-namespace unlocked package in an org with a namespace.</p>	<p>Pass.</p> <p>If the namespace of the unlocked package is different from the namespace of the org, you can install one or many packages.</p>	<p>Pass.</p> <p>If the namespace of the 2GP is different from the namespace of the org, you can install one or many packages.</p> <p>Fail.</p> <p>If the namespace of the 1GP is the same as the namespace of the org, you can't install the 1GP into the org.</p>
<b>Org without a namespace</b>	<p>Pass.</p> <p>Can install one or many no-namespace unlocked packages.</p>	<p>Pass.</p> <p>Can install one or many namespaced unlocked packages.</p>	<p>Pass.</p> <p>Can install one or many 2GP packages.</p>	<p>Pass.</p> <p>Can install one or many 1GP packages.</p>

To understand how namespaces affect the combination of packages that can be installed into one org, review this table.

Namespace and Package Type	Unlocked Package with Namespace Y	Second-generation Managed Package (2GP) with Namespace Y	First-generation Managed Package (1GP) with Namespace Y
<b>First-generation Managed Package (1GP) with namespace X</b>	<p>Pass.</p> <p>If the 1GP and unlocked package use unique namespaces, you can install them in the same org.</p>	<p>Pass.</p> <p>If the 1GP and 2GP use a unique namespace, you can install them in the same org.</p>	<p>Pass.</p> <p>If each 1GP uses a unique namespace, you can install multiple 1GP packages in the same org.</p>
<b>First-generation Managed Package (1GP) with namespace Y</b>	<p>Fail.</p> <p>If the 1GP and unlocked package share a namespace, you can't install them in the same org.</p>	<p>Fail.</p> <p>If the 1GP and 2GP share a namespace, you can't install them in the same org.</p>	<p>Fail.</p> <p>If the 1GP packages share a namespace, you can't install them in the same org.</p>
<b>Second-generation Managed Package (2GP) with namespace X</b>	<p>Pass.</p> <p>You can install a 2GP and a namespaced unlocked package in the same org. The packages can share a namespace or use unique namespaces.</p>	<p>Pass.</p> <p>You can install multiple 2GP packages with unique namespaces, or the same namespace.</p>	<p>Pass.</p> <p>If the 1GP packages use unique namespaces, you can install multiple 1GP packages in the same org.</p>

Namespace and Package Type	Unlocked Package with Namespace Y	Second-generation Managed Package (2GP) with Namespace Y	First-generation Managed Package (1GP) with Namespace Y
<b>Second-generation Managed Package (2GP) with namespace Y</b>	Pass.  You can install a 2GP and a namespaced unlocked package in the same org. The packages can share a namespace or use unique namespaces.	Pass.  You can install multiple 2GP packages with the same namespace in the same org.	Fail.  If the 1GP and 2GP share a namespace, you can't install them in the same org.

## Namespace-Based Visibility for Apex Classes in Unlocked Packages

The `@namespaceAccessible` makes public Apex in a package available to other packages that use the same namespace. Without this annotation, Apex classes, methods, interfaces, and properties defined in an unlocked package aren't accessible to the other packages with which they share a namespace. Apex that is declared global is always available across all namespaces, and needs no annotation.

### Considerations for Apex Accessibility Across Packages

- A Lightning component outside the package can access a public Apex method installed from a no-namespace unlocked package. The component can be installed from another package or created in the org. For accessing Apex methods, a no-namespace unlocked package is treated the same as an unmanaged package.
- You can't use the `@namespaceAccessible` annotation for an `@AuraEnabled` Apex method.
- You can add or remove the `@namespaceAccessible` annotation at any time, even on managed and released Apex code. Make sure that you don't have dependent packages relying on the functionality of the annotation before adding or removing it.
- When adding or removing `@namespaceAccessible` Apex from a package, consider the impact to users with installed versions of other packages that reference this package's annotation. Before pushing a package upgrade, ensure that no user is running a package version that would fail to fully compile when the upgrade is pushed.

This example shows an Apex class marked with the `@namespaceAccessible` annotation. The class is accessible to other packages within the same namespace. The first constructor is also visible within the namespace, but the second constructor isn't.

```
// A namespace-visible Apex class
@namespaceAccessible
public class MyClass {
 private Boolean bypassFLS;

 // A namespace-visible constructor that only allows secure use
 @namespaceAccessible
 public MyClass() {
 bypassFLS = false;
 }

 // A package private constructor that allows use in trusted contexts,
 // but only internal to the package
 public MyClass (Boolean bypassFLS) {
 this.bypassFLS = bypassFLS;
 }
 @namespaceAccessible
 protected Boolean getBypassFLS() {
 return bypassFLS;
 }
}
```

```
 }
}
```

## Share Release Notes and Post-Install Instructions

Share details about what's new and changed in a released unlocked package with your users.

Share details about what's new and changed in an unlocked package with your users. You can specify a release notes URL to display on the package detail page in the user's org. And you can share instructions about using your package by specifying a post install URL. The release notes and post install URLs display on the Installed Packages page in Setup, after a successful package installation. For users who install packages using an installation URL, the package installer page displays a link to release notes. And users are redirected to your post install URL following a successful package installation or upgrade.

Specify the `postInstallUrl` and `releaseNotesUrl` attributes in the `packageDirectories` section for the package.

```
"packageDirectories": [
 {
 "path": "expenser-schema",
 "default": true,
 "package": "Expense Schema",
 "versionName": "'ver 0.3.2'",
 "versionNumber": "0.3.2.NEXT",
 "postInstallUrl": "https://expenser.com/post-install-instructions.html",
 "releaseNotesUrl": "https://expenser.com/winter-2020-release-notes.html"
 },
 {
 "namespace": "",
 "sfdcLoginUrl": "https://login.salesforce.com",
 "sourceApiVersion": "47.0",
 "packageAliases": {
 "Expenser Schema": "0HoB00000004CzHKAU",
 "Expenser Schema@0.1.0-1": "04tB0000000719qIAA"
 }
 }]
```

You can also use the `--postinstallurl` and the `--releasenotesurl` Salesforce CLI parameters with the `force:package:version:create` command. The CLI parameters override the URLs specified in the `sfdx-project.json` file.

## Specify Unpackaged Metadata or Apex Access for Apex Tests (Unlocked Packages)

### Specify Unpackaged Metadata for Package Version Creation Tests

Specify the path to the unpackaged metadata in your `sfdx-project.json` file.

In this example, metadata in the `my-unpackaged-directory` is available for test runs during the package version creation of the `TV_unl` package.

```
"packageDirectories": [
 {
```

```
"path": "force-app",
"package": "TV_unl",
"versionName": "ver 0.1",
"versionNumber": "0.1.0.NEXT",
"default": true,
"unpackagedMetadata": {
 "path": "my-unpackaged-directory"
},
],
]
```

The `unpackagedMetadata` attribute is intended for metadata that isn't part of your package. You can't include the same metadata in both an unpackaged directory and a packaged directory.

## Manage Apex Access for Package Version Creation Tests

Sometimes the Apex tests that you write require a user to have certain permission sets or permission set licenses. Use the `apexTestAccess` setting to assign permission sets and permission set licenses to the user in whose context your Apex tests get run at package version creation.

```
"packageDirectories": [
 {
 "path": "force-app",
 "package": "TV_unl",
 "versionName": "ver 0.1",
 "versionNumber": "0.1.0.NEXT",
 "default": true,
 "unpackagedMetadata": {
 "path": "my-unpackaged-directory"
 },
 "apexTestAccess": {
 "permissionSets": [
 "Permission_Set_1",
 "Permission_Set_2"
],
 "permissionSetLicenses": [
 "SalesConsoleUser"
]
 }
 },
]
```

 **Note:** To assign user licenses, use the [rusAs Method](#). User licenses can't be assigned in the `sfdx-project.json` file.

## Best Practices for Unlocked Packages

We suggest that you follow these best practices when working with unlocked packages.

- We recommend that you work with only one Dev Hub, and enable Dev Hub in a production org.
- The Dev Hub org against which you run the `force:package:create` command becomes the owner of the package. If the Dev Hub org associated with a package expires or is deleted, its packages no longer work.

- Use care in deciding how to utilize namespaces. For most customers, we recommend working with no namespace or a single namespace to avoid unnecessary complexity in managing components. If you're test-driving unlocked packages, use a test namespace. Use real namespaces only when you're ready to embark on a development path headed for release in a production org.



**Note:** You can't install a no-namespace, unlocked package into any org with a namespace (for example, a scratch org with a namespace).

- Include the `--tag` option when you use the `package:version:create` and `package:version:update` commands. This option helps you keep your version control system tags in sync with specific package versions.
- Create user-friendly aliases for packaging IDs, and include those aliases in your Salesforce DX project file and when running CLI packaging commands. See: [Package IDs and Aliases for Unlocked Packages](#).

## Package IDs and Aliases for Unlocked Packages

During the package lifecycle, packages and package versions are identified by an ID or package alias. When you create a package or package version, Salesforce CLI creates a package alias based on the package name, and stores that name in the `sfdx-project.json` file. When you run CLI commands or write scripts to automate packaging workflows, it's often easier to reference the package alias, instead of the package ID or package version ID.

Package aliases are stored in the `sfdx-project.json` file as name-value pairs, in which the name is the alias and the value is the ID. You can modify package aliases for existing packages and package versions in the project file.

At the command line, you also see IDs for things like package members (a component in a package) and requests (like a `package:version:create` request).



**Note:** As a shortcut, the documentation sometimes refers to an ID by its three-character prefix. For example, a package version ID always starts with `04t`.

Here are the most commonly used IDs.

ID Example	Short ID Name	Description
033J0000dAb27uxVRE	Subscriber Package ID	Use this ID when contacting Salesforce for packaging or security review support. To locate this ID for your package, run <code>force:package:list --verbose</code> against the Dev Hub that owns the package.
04t6A0000004eytQAA	Subscriber Package Version ID	Use this ID to install a package version. Returned by <code>force:package:version:create</code> .
0Hoxx00000000CqCAI	Package ID	Use this ID on the command line to create a package version. Or enter it into the <code>sfdx-project.json</code> file and use the directory name. Generated by <code>force:package:create</code> .
08cxx00000000BEAYY	Version Creation Request ID	Use this ID to view the status and monitor progress for a specific request to create a package version such as <code>force:package:version:create:report</code>

## Frequently Used Packaging Operations

For a complete list of Salesforce CLI packaging commands, see: [Salesforce Command Line Reference Guide](#).

Salesforce CLI command	What it Does
<code>force:package:create</code>	Creates a package. When you create a package, you specify its package type and name, among other things.
<code>force:package:version:create</code>	Creates a package version.
<code>force:package:install</code>	Installs a package version in a scratch, sandbox, or production org.
<code>force:package:uninstall</code>	Removes a package that has been installed in an org. This process deletes the metadata and data associated with the package.
<code>force:package:version:promote</code>	Changes the state of the package version from beta to the managed-released state.
<code>force:org:create</code>	Creates a scratch org.
<code>force:org:open</code>	Opens an org in the browser.

## How We Handle Profile Settings in Unlocked Packages

When you package a profile in an unlocked or second-generation managed package, the build system inspects the contents of the profile during package creation and preserves only the profile settings that are directly related to the metadata in the package. The profile itself, and any profile settings unrelated to the package's metadata are discarded from the package.

During package installation, the preserved profile settings are applied only to existing profiles in the subscriber org. The profile itself is not installed in the subscriber org.

 **Note:** Packages that contain only profiles and no additional metadata aren't allowed and fail during package version creation.

When you select...	The packaged profile settings are applied to...	This installation option is available via...
Install for Admins Only	<p>The System Administrator profile in the subscriber org.</p> <p>CRUD access to custom objects is granted automatically to the System Administration profile.</p>	<ul style="list-style-type: none"> <li>The package installer page</li> <li>Salesforce CLI <code>sfdx force:package:install</code> command</li> </ul> <p>The default behavior for CLI-based package installs is to install for admins only.</p>
Install for All Users	<p>The System Administrator profile and all cloned profiles in the subscriber org.</p> <p>CRUD access to custom objects is granted automatically to the System Administration profile, and all cloned profiles.</p> <p><a href="#">Standard profiles</a> can't be modified.</p>	<ul style="list-style-type: none"> <li>The package installer page</li> <li>Salesforce CLI <code>sfdx force:package:install</code> command</li> </ul> <p>To install for all users via the CLI, include the security type parameter.</p>

When you select...	The packaged profile settings are applied to...	This installation option is available via...
		<code>sfdx force:package:install --securitytype AllUsers</code>
Install for Specific Profiles	Specific profiles in the subscriber org. This selection lets the person installing your package determine how to map the profile settings you packaged to specific profiles in their org.	<ul style="list-style-type: none"> <li>• The package installer page</li> </ul> <p>Not available for CLI-based package installations.</p>

To test the behavior of your packaged profile, install your package in a scratch org.

1. From Setup, enter *Profile* in the Quick Find box, and then locate and inspect the profiles you selected during package installation.
2. Check whether your profile settings have been applied to that profile.

Repeat this step for any other profile you expect to contain your profile settings. Don't look for the profile name you created; we apply profile settings to existing profiles in the subscriber org.

Whenever possible, use package permission sets instead of profile settings. Subscribers who install your package can easily assign your permission set to their users.



**Note:** During a push upgrade, some profile settings related to Apex classes and field-level security aren't automatically assigned to the System Admin profile. Communicate with your customer to ensure that user access is set up correctly after a push upgrade. Make sure you review and update your profile settings after push upgrade.

## Retain License Settings in Unlocked Packages

By default, license settings in profiles are removed during package creation. To retain these settings, specify the `includeProfileUserLicenses` parameter in your `sfdx-project.json` file. In this scenario, the license settings are retained and applied to the profiles in the subscriber org that are selected during package installation.

```
"packageDirectories": [
 {
 "package": "PackageA",
 "path": "common",
 "versionName": "ver 0.1",
 "versionNumber": "0.1.0.NEXT",
 "default": false,
 includeProfileUserLicenses: true
 }
]
```

## Create a Package

A package is a top-level container that holds important details about the app or package: the package name, description, and associated namespace.

You supply the package details in the package descriptor section of your `sfdx-project.json` project configuration file.

### [Generate the Package](#)

When you're ready to test or share your package, use the `force:package:create` command to create a package.

### [Generate a Package Version](#)

A package version is a fixed snapshot of the package contents and related metadata. The package version lets you manage changes and track what's different each time you release or deploy a specific set of changes.

### [Code Coverage for Unlocked Packages](#)

Before you can promote and release an unlocked package, the Apex code must meet a minimum 75% code coverage requirement. You can install package versions that don't meet code coverage requirements only in scratch orgs and sandboxes.

### [Release an Unlocked Package](#)

Each new package version is marked as beta when it's created. As you develop your package, you may create several package versions before you create a version that is ready to be released and installed in production orgs.

### [Update a Package Version](#)

You can update most properties of a package version from the command line. For example, you can change the package version name or description. One important exception is that you can't change the release status.

### [Hard-Deleted Components in Unlocked Packages](#)

When these components are removed from an unlocked package, they're hard deleted from the target install org during the package upgrade.

### [Delete an Unlocked Package or Package Version](#)

Use the `force:package:version:delete` and `force:package:delete` commands to delete packages and package versions that you no longer need.

### [View Package Details](#)

View the details of previously created packages and package versions from the command line.

## Generate the Package

When you're ready to test or share your package, use the `force:package:create` command to create a package.

If you are using a namespace, specify the package namespace in the `sfdx-project.json` file. To learn more, see [Understanding Namespaces](#).

To create the package, change to the project directory. The name becomes the package alias, which is automatically added to the project file. You can choose to designate an active Dev Hub org user to receive email notifications for Apex gacks, and install, upgrade, or uninstall failures associated with your packages.

```
sfdx force:package:create --name "Expenser App" --packagetype Unlocked --path \
"expenser-main" --targetdevhubusername my-hub --errornotificationusername me@devhub.org
```

The output is similar to this example.

```
sfdx-project.json has been updated.
Successfully created a package. 0HoB00000004CzHKAU
==== Ids
NAME VALUE

Package Id 0HoB00000004CzHKAU
```

## Update the Package

To update the name, description, or the user to receive error notifications of an existing package, use this command.

```
sfdx force:package:update --package "Expense App" --name "Expense Manager App" \
--description "New Description" --errornotificationusername me2@devhub.org
```

 **Note:** You can't change the package namespace or package type after you create the package.

## Generate a Package Version

A package version is a fixed snapshot of the package contents and related metadata. The package version lets you manage changes and track what's different each time you release or deploy a specific set of changes.

Before you create a package version, first verify package details, such as the package name, dependencies, and major, minor, and patch version numbers, in the `sfdx-project.json` file. Verify that the metadata you want to change or add in the new package version is located in the package's main directory.

## How Many Package Versions Can I Create Per Day?

Run this command to see how many package versions you can create per day and how many you have remaining.

```
sfdx force:limits:api:display
```

Look for the `Package2VersionCreates` entry.

NAME	REMAINING	MAXIMUM
Package2VersionCreates	23	50

## Create a Package Version

Create the package version with this command. Specify the package alias or ID (0Ho). You can also include a scratch definition file that contains a list of features and setting that the metadata of the package version depends on.

```
sfdx force:package:version:create --package "Expenser App" --installationkey "HIF83ks8ks7C" \
--definitionfile config/project-scratch-def.json --wait 10
```

 **Note:** When creating a package version, specify a `--wait` time to run the command in non-asynchronous mode. If the package version is created within that time, the `sfdx-project.json` file is automatically updated with the package version information. If not, you must manually edit the project file.

It can be a long-running process to create a package version, depending on the package size and other variables. You can easily view the status and monitor progress.

```
sfdx force:package:version:create:report --packagecreaterequestid 08cxx00000000YDAAY
```

The output shows details about the request.

==== Package Version Create Request	
NAME	VALUE

Version Create Request Id	08cB00000004CBxIAM
Status	InProgress
Package Id	0HoB00000004C9hKAE
Package Version Id	05iB0000000CaaNIAS
Subscriber Package Version Id	04tb0000000NOimIAG
Tag	git commit id 08dcfsdf
Branch	
CreatedDate	2018-05-08 09:48
Installation URL	<a href="https://login.salesforce.com/packaging/installPackage.apexp?p0=04tb0000000NOimIAG">https://login.salesforce.com/packaging/installPackage.apexp?p0=04tb0000000NOimIAG</a>

You can find the request ID (08c) in the initial output of `force:package:version:create`.

Depending on the size of the package and other variables, the create request can take several minutes. When you've more than one pending request to create package versions, you can view a list of all requests with this command.

```
sfdx force:package:version:create:list --createdlastdays 0
```

Details for each request display as shown here (IDs and labels truncated).

== Package Version Create Requests [3]								
ID	STATUS	PACKAGE2 ID	PKG2 VERSION ID	SUB PKG2 VER ID	TAG	BRANCH	CREATED DATE	==
08c...	Error	0Ho...						
08c...	Success	0Ho...	05i...	04t...			2017-06-22 12:07	
08c...	Success	0Ho...	05i...	04t...			2017-06-23 14:55	

### [Simplify Package Development by Creating and Specifying an Org Shape](#)

If your package's metadata depends on a complex set of features, settings, or licenses, it can be difficult to declaratively specify these dependencies in a scratch org definition file. Instead, create an org shape of your production org, or another development org, and specify that source org's ID in your scratch org definition file. During package creation, we mimic the source org's environment when we build and validate your package's metadata.

### [Use Branches in Unlocked Packaging](#)

Development teams who use branches in their source control system (SCS), often build package versions based on the metadata in a particular branch of code.

### [Skip Validation to Quickly Iterate Package Development](#)

Iterate package development more efficiently by skipping validation of dependencies, package ancestors, and metadata during package version creation. Skipping validation reduces the time it takes to create a new package version, but you can promote only validated package versions to the released state.

### [Target a Specific Release for Your Unlocked Packages During Salesforce Release Transitions](#)

During major Salesforce release transitions, you can specify `preview` or `previous` when creating a package version. Specifying the release version for a package allows you to test upcoming features, run regression tests, and support customers regardless of which Salesforce release their org is on. Previously, you could only create package versions that matched the Salesforce release your Dev Hub org was on.

## Simplify Package Development by Creating and Specifying an Org Shape

If your package's metadata depends on a complex set of features, settings, or licenses, it can be difficult to declaratively specify these dependencies in a scratch org definition file. Instead, create an org shape of your production org, or another development org, and specify that source org's ID in your scratch org definition file. During package creation, we mimic the source org's environment when we build and validate your package's metadata.

Before using this feature, get familiar with how [Org Shape for Scratch Orgs](#) works.

Then [enable the scratch org setting](#) in your source org, [generate the org shape](#), and edit your scratch org definition file to include the org name and 15-character source org ID.

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

## Use Branches in Unlocked Packaging

Development teams who use branches in their source control system (SCS), often build package versions based on the metadata in a particular branch of code.

To identify which branch in your SCS a package version is based on, tag your package version with a branch name using `--branch` attribute in this Salesforce CLI command.

```
sfdx force:package:version:create --branch featureA
```

You can specify any alphanumeric value up to 240 characters as the branch name.

You can also specify the branch name in the package directories section of the `sfdx-project.json` file.

```
"packageDirectories": [
 {
 "path": "util",
 "default": true,
 "package": "pkgA",
 "versionName": "Spring '21",
 "versionNumber": "4.7.0.NEXT",
 "branch": "featureA"
 }
]
```

When you specify a branch, the package alias for that package version is automatically appended with the branch name. You can view the package alias in the `sfdx.project.json` file.

```
"packageAliases": {
 "pkgA@1.0.0.4-featureA":"04tB0000000IB1EIAW"}
```

Keep in mind that version numbers increment within each branch, and not across branches. For example, you could have two or more beta package versions with the version number 1.3.0.1.

Branch Name	Package Version Alias
featureA	pkgA@1.3.0-1-featureA
featureB	pkgA@1.3.0-1-featureB
Not specified	pkgA@1.3.0-1

Although more than one beta package version can have the same version number, there can be only one promoted and released package version for a given major.minor.patch package version.

## Package Dependencies and Branches

By default, your package can have dependencies on other packages in the same branch. For package dependencies based on packages in other branches, explicitly set the branch attribute in the `sfdx.project.json` file.

To specify a package dependency	Use this format
Using the branch attribute	<pre>"dependencies": [   {     "package": "pkgB",     "versionNumber": "1.3.0.LATEST",     "branch": "featureC"   } ]</pre>
Using the most recent promoted and released version of package	<pre>"dependencies": [   {     "package": "pkgB",     "versionNumber": "2.1.0.RELEASED"   } ]</pre>
If your package has an associated branch, but the dependent package doesn't have a branch	<pre>"dependencies": [   {     "package": "pkgB",     "versionNumber": "1.3.0.LATEST",     "branch": ""   } ]</pre>
Using the package alias	<pre>"dependencies": [   {     "package": "pkgB@2.1.0-1-featureC"   } ]</pre>

## Skip Validation to Quickly Iterate Package Development

Iterate package development more efficiently by skipping validation of dependencies, package ancestors, and metadata during package version creation. Skipping validation reduces the time it takes to create a new package version, but you can promote only validated package versions to the released state.

```
sfdx force:package:version:create --skipvalidation
```

In Tooling API, use the `SkipValidation` field on the [Package2VersionCreateRequest](#) object.

 **Note:** You can't specify both skip validation and code coverage, because code coverage is calculated during validation.

## Target a Specific Release for Your Unlocked Packages During Salesforce Release Transitions

During major Salesforce release transitions, you can specify `preview` or `previous` when creating a package version. Specifying the release version for a package allows you to test upcoming features, run regression tests, and support customers regardless of which Salesforce release their org is on. Previously, you could only create package versions that matched the Salesforce release your Dev Hub org was on.

To create a package version based on a preview or previous Salesforce release version, create a scratch org definition file that includes either:

```
{
 "release": "previous"
}
```

or

```
{
 "release": "preview"
}
```

In the `sfdx-project.json` file, set the `sourceApiVersion` to correspond with the release version of the package version you're creating. If you are targeting a previous release, any `sourceApiVersion` value below the current release is accepted.

Then when you create your package version, specify the scratch org definition file.

```
sfdx force:package:version:create --package pkgA --definitionfile
config/project-scratch-def.json
```

Preview start date is when sandbox instances are upgraded. Preview end date is when all instances are on the GA release.

Release Version	Preview Start Date	Preview End Date
Spring '23	January 8, 2023	February 11, 2023
Summer '23	May 7, 2023	June 10, 2023
Winter '24	September 10, 2023	October 14, 2023

## Code Coverage for Unlocked Packages

Before you can promote and release an unlocked package, the Apex code must meet a minimum 75% code coverage requirement. You can install package versions that don't meet code coverage requirements only in scratch orgs and sandboxes.

**!** **Important:** Unlocked package versions that were promoted to the released state before Winter '21 aren't subject to code coverage requirements.

To compute code coverage using Salesforce CLI, use the `--codecoverage` parameter when you run the `force:package:version:create` command.

Package version creation can take longer to complete when code coverage is being computed, so consider when in the development cycle to include the code coverage parameter. You can choose to skip code coverage, and you can skip all validation by specifying the `--skipvalidation` parameter. You can promote package versions only if they're validated and meet code coverage requirements.

View code coverage information for a package version using `force:package:version:list` with the `--verbose` parameter, or the `force:package:version:report` command in Salesforce CLI.

We don't calculate code coverage for org-dependent unlocked packages.

## Release an Unlocked Package

Each new package version is marked as beta when it's created. As you develop your package, you may create several package versions before you create a version that is ready to be released and installed in production orgs.

Before you promote the package version, ensure that the user permission, **Promote a package version to released**, is enabled in the Dev Hub org associated with the package. Consider creating a permission set with this user permission, and then assign the permission set to the appropriate user profiles.

When you're ready to release, use `force:package:version:promote`.

```
sfdx force:package:version:promote --package "Expense Manager@1.3.0-7"
```

If the command is successful, a confirmation message appears.

```
Successfully promoted the package version, ID: 04tB0000000719qIAA to released.
```

After the update succeeds, view the package details.

```
sfdx force:package:version:report --package "Expense Manager@1.3.0.7"
```

Confirm that the value of the Released property is `true`.

==== Package Version	
NAME	VALUE
Name	ver 1.0
Alias	Expense Manager-1.0.0.5
Package Version Id	05iB0000000CaahIAC
Package Id	0HoB0000000CabmKAC
Subscriber Package Version Id	04tB0000000NPbBIW
Version	1.0.0.5
Description	update version
Branch	
Tag	git commit id 08dcfsdf
Released	true
Created Date	2018-05-08 09:48
Installation URL	<a href="https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB0000000NPbBIW">https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB0000000NPbBIW</a>

You can promote and release only once for each package version number, and you can't undo this change.

## Update a Package Version

You can update most properties of a package version from the command line. For example, you can change the package version name or description. One important exception is that you can't change the release status.

If the most recent package version has been released, increment either the major, minor, or patch version number for the next package version you create.

Package version numbers use the format major.minor.patch.build. For example, if you released package 1.0.0.2, you could use 1.1.0.0, 2.0.0.0, or 1.0.1.0 for the next package version.

## Hard-Deleted Components in Unlocked Packages

When these components are removed from an unlocked package, they're hard deleted from the target install org during the package upgrade.

- AccountForecastSettings
- AcctMgrTargetSettings
- ActionableListDefinition
- ActionPlanTemplate
- AccountingFieldMapping
- AccountingModelConfig
- AdvAccountForecastSet
- AdvAcctForecastDimSource
- AdvAcctForecastPeriodGroup
- AIApplicationConfig
- AIUsecaseDefinition
- AnalyticSnapshot
- ApexClass
- ApexComponent
- ApexPage
- ApexTrigger
- ApplicationRecordTypeConfig
- ApplicationSubtypeDefinition
- AppointmentAssignmentPolicy
- AssessmentQuestion
- AssessmentQuestionSet
- AssistantContextItem
- AssistantSkillQuickAction
- AssistantSkillSobjectAction
- AssistantVersion
- AuraDefinitionBundle
- BatchCalcJobDefinition
- BatchProcessJobDefinition
- BenefitAction
- BldgEnergyIntensityCnfg
- BrandingSet
- BriefcaseDefinition
- BusinessProcessGroup
- BusinessProcessTypeDefinition
- CareBenefitVerifySettings
- CareLimitType
- CareProviderSearchConfig

- CareRequestConfiguration
- ChannelObjectLinkingRule
- ClaimFinancialSettings
- ClauseCatgConfiguration
- CompactLayout
- ContractType
- ConversationVendorInfo
- CustomApplication
- CustomPageWebLink
- CustomPermission
- CustomTab
- Dashboard
- DecisionMatrixDefinition
- DecisionMatrixDefinitionVersion
- DecisionTable
- DecisionTableDatasetLink
- DisclosureDefinition
- DisclosureDefinitionVersion
- DisclosureType
- DiscoveryAIModel
- DiscoveryGoal
- Document
- DocumentGenerationSetting
- DocumentType
- EmailServicesFunction
- EmailTemplate
- EmbeddedServiceBranding
- EmbeddedServiceConfig
- EmbeddedServiceLiveAgent
- EmbeddedServiceMenuSettings
- ESignatureConfig
- ESignatureEnvelopeConfig
- ExplainabilityActionDefinition
- ExplainabilityActionVersion
- ExplainabilityMsgTemplate
- ExpressionSetDefinition
- ExpressionSetDefinitionVersion
- ExpressionSetObjectAlias
- ExternalAIModel
- ExternalClientApplication

- ExtICIntAppMobileSettings
- ExtICIntAppOauthSettings
- ExternalDataSrcDescriptor
- ExternalServiceRegistration
- FeatureParameterBoolean
- FeatureParameterDate
- FeatureParameterInteger
- FieldRestrictionRule
- FieldServiceMobileExtension
- FlexiPage
- FuelType
- FuelTypeSustnUom
- GatewayProviderPaymentMethodType
- HomePageComponent
- HomePageLayout
- IdentityVerificationProcDef
- InstalledPackage
- IntegrationHubSettings
- IntegrationHubSettingsType
- IntegrationProviderDef
- Layout
- Letterhead
- LicenseDefinition
- LightningComponentBundle
- LightningExperienceTheme
- LightningMessageChannel
- LightningOnboardingConfig
- ListView
- LiveChatAgentConfig
- LiveChatButton
- LiveChatSensitiveDataRule
- LocationUse
- LoyaltyProgramSetup
- MarketingAppExtActivity
- MarketingAppExtension
- MatchingRule
- MfgProgramTemplate
- MLDataDefinition
- MLPredictionDefinition
- NamedCredential

- NetworkBranding
- ObjectHierarchyRelationship
- OcrSampleDocument
- OcrTemplate
- OmniDataTransform
- OmniIntegrationProcedure
- OmniScript
- OmniUiCard
- PaymentGatewayProvider
- PermissionSet
- PermissionSetGroup
- PermissionSetLicense
- PipelineInspMetricConfig
- PlatformEventSubscriberConfig
- ProductAttributeSet
- ProductSpecificationTypeDefinition
- Profile
- QuickAction
- RecordAlertCategory
- RecordAlertDataSource
- RegisteredExternalService
- RelatedRecordAssocCriteria
- RelationshipGraphDefinition
- RemoteSiteSetting
- Report
- ReportType
- RestrictionRule
- SalesAgreementSettings
- SchedulingRule
- SchedulingObjective
- ScoreCategory
- ServiceAISetupDefinition
- ServiceAISetupField
- ServiceProcess
- SharingReason
- SharingRecalculation
- SlackApp
- StaticResource
- StnryAssetEnvSrcCnfg
- SustainabilityUom

- SustnUomConversion
- SvcCatalogCategory
- SvcCatalogFulfillmentFlow
- SvcCatalogItemDef
- TimelineObjectDefinition
- UIObjectRelationConfig
- UserAccessPolicy
- UserLicense
- UserProfileSearchScope
- ValidationRule
- VehicleAssetEmssnSrcCnfg
- ViewDefinition
- VirtualVisitConfig
- WaveApplication
- WaveComponent
- WaveDashboard
- WaveDataflow
- WaveDataset
- WaveLens
- WaveRecipe
- WaveTemplateBundle
- WaveXmd
- WebLink
- WebStoreTemplate
- WorkflowAlert
- WorkflowFieldUpdate
- WorkflowFlowAction
- WorkflowOutboundMessage
- WorkflowRule
- WorkflowTask

## Delete an Unlocked Package or Package Version

Use the `force:package:version:delete` and `force:package:delete` commands to delete packages and package versions that you no longer need.

To delete a package or package version, users need the Delete Second-Generation Packages user permission. Before you delete a package, first delete all associated package versions.

Package Type	Can I delete beta packages and package versions?	Can I delete released packages and package versions?
Second-Generation Managed Packages	Yes	No

Package Type	Can I delete beta packages and package versions?	Can I delete released packages and package versions?
Unlocked Packages	Yes	Yes

### Considerations for Deleting a Package or Package Version

- Deletion is permanent.
- Attempts to install a deleted package version will fail.
- Before deleting, ensure that the package or package version isn't referenced as a dependency.

#### Examples:

```
$ sfdx force:package:delete -p "Your Package Alias"
$ sfdx force:package:delete -p 0Ho...
$ sfdx force:package:version:delete -p "Your Package Version Alias"
$ sfdx force:package:version:delete -p 04t...
```

These CLI commands can't be used with first-generation managed packages or package versions. To delete a first-generation managed package, see [View Package Details](#) in the *ISVforce Guide*.

## View Package Details

View the details of previously created packages and package versions from the command line.

To display a list of all packages in the Dev Hub org, use this command.

```
sfdx force:package:list --targetdevhubusername my-hub
```

You can view the namespace, package name, ID, and other details in the output.

Name	ID	Alias	Description	Type
Expenser App	0HoB00000004CzRKAU	Expenser App		Unlocked
Expenser Logic	0HoB00000004CzMKAU	Expenser Logic		Unlocked
Expenser Schema	0HoB00000004CzHKAU	Expenser Schema		Unlocked

Include optional parameters to filter the list results based on the modification date, creation date, and to order by specific fields or package IDs. To limit the details, use `--concise`. To show expanded details, use `--verbose`.

To display a list of all package versions in the Dev Hub org, use this command.

```
sfdx force:package:version:list --targetdevhubusername my-hub
```

You can view the namespace, version name, and other details in the output.

Package Name Installation Key	Namespace Released	Version	Sub Pkg Ver	Id	Alias
Expenser Schema		0.1.0.1	04tB0000000719qIAA	Expenser Schema@0.1.0-1	false

Expenser Schema	true	0.2.0.1	04tB000000071AjIAI	Expenser Schema@0.2.0-1	false
Expenser Schema	true	0.3.0.1	04tB000000071AtIAI	Expenser Schema@0.3.0-1	false
Expenser Schema	false	0.3.0.2	04tB000000071AyIAI	Expenser Schema@0.3.0-2	false
Expenser Schema	true	0.3.1.1	04tB0000000KGU6IAO	Expenser Schema@0.3.1-1	false
Expenser Schema	false	0.3.1.2	04tB0000000KGUBIA4	Expenser Schema@0.3.1-2	false
Expenser Schema	true	0.3.2.1	04tB0000000KGUQIA4	Expenser Schema@0.3.2-1	false
Expenser Logic	true	0.1.0.1	04tB000000071vIAA	Expenser Logic@0.1.0-1	false
Expenser App	true	0.1.0.1	04tB000000071A0IAI	Expenser App@0.1.0-1	false

## Push a Package Upgrade

Push upgrades enable you to upgrade packages installed in subscriber orgs, without asking customers to install the upgrade themselves. You can choose which orgs receive a push upgrade, what version the package is upgraded to, and when you want the upgrade to occur. Push upgrades are particularly helpful if you need to push a change for a hot bug fix.

For more information, see [Push a Package Upgrade](#) in the Second-Generation Managed Packaging section.

## Install a Package

Install unlocked packages using the CLI or the browser. You can install package versions in a scratch org, sandbox org, DE org, or production org.

### [Install Packages with the CLI](#)

If you're working with the Salesforce CLI, you can use the `force:package:install` command to install packages in a scratch org or target subscriber org.

### [Install Packages from a URL](#)

Install unlocked packages from the CLI or from a browser, similar to how you install managed packages.

### [Upgrade a Package Version](#)

Are you introducing metadata changes to an existing package? You can use the CLI to upgrade one package version to another.

### [Sample Script for Installing Packages with Dependencies](#)

Use this sample script as a basis to create your own script to install packages with dependencies. This script contains a query that finds dependent packages and installs them in the correct dependency order.

## Install Packages with the CLI

If you're working with the Salesforce CLI, you can use the `force:package:install` command to install packages in a scratch org or target subscriber org.

Before you install a package to a scratch org, run this command to list all the packages and locate the ID or package alias.

```
sfdx force:package:version:list
```

Identify the version you want to install. Enter this command, supplying the package alias or package ID (starts with 04t).

```
sfdx force:package:install --package "Expense Manager@1.2.0-12" --targetusername jdoe@example.com
```

If you've already set the scratch org with a default username, enter just the package version ID.

```
sfdx force:package:install --package "Expense Manager@1.2.0-12"
```

-  **Note:** If you've defined an alias (with the `-a` parameter), you can specify the alias instead of the username for `--targetusername`.

The CLI displays status messages regarding the installation.

```
Waiting for the subscriber package version install request to get processed. Status = InProgress Successfully installed the subscriber package version: 04txx0000000FIuAAM.
```

## Control Package Installation Timeouts

When you issue a `force:package:install` command, it takes a few minutes for a package version to become available in the target org and for installation to complete. To allow sufficient time for a successful install, use these parameters that represent mutually exclusive timers.

- `--publishwait` defines the maximum number of minutes that the command waits for the package version to be available in the target org. The default is 0. If the package is not available in the target org in this time frame, the install is terminated.

Setting `--publishwait` is useful when you create a new package version and then immediately try to install it to target orgs.

-  **Note:** If `--publishwait` is set to 0, the package installation immediately fails, unless the package version is already available in the target org.

- `--wait` defines the maximum number of minutes that the command waits for the installation to complete after the package is available. The default is 0. When the `--waitinterval` ends, the install command completes, but the installation continues until it either fails or succeeds. You can poll the status of the installation using `sfdx force:package:install:report`.

-  **Note:** The `--wait` timer takes effect after the time specified by `--publishwait` has elapsed. If the `--publishwait` interval times out before the package is available in the target org, the `--wait` interval never starts.

For example, consider a package called Expense Manager that takes five minutes to become available on the target org, and 11 minutes to install. The following command has `publishwait` set to three minutes and `wait` set to 10 minutes. Because Expense Manager requires more time than the set `publishwait` interval, the installation is aborted at the end of the three minute `publishwait` interval.

```
sfdx force:package:install --package "Expense Manager@1.2.0-12" --publishwait 3 --wait 10
```

The following command has `publishwait` set to six minutes and `wait` set to 10 minutes. If not already available, Expense Manager takes five minutes to become available on the target org. The clock then starts ticking for the 10 minute `wait` time. At the end of 10 minutes, the command completes because the `wait` time interval has elapsed, although the installation is not yet complete. At this point, `package:install:report` indicates that the installation is in progress. After one more minute, the installation completes and `package:install:report` indicates a successful installation.

```
sfdx force:package:install --package "Expense Manager@1.2.0-12" --publishwait 6 --wait 10
```

## Install Packages from a URL

Install unlocked packages from the CLI or from a browser, similar to how you install managed packages.

If you create packages from the CLI, you can derive an installation URL for the package by adding the subscriber package ID to your Dev Hub URL. You can use this URL to test different deployment or installation scenarios.

For example, if the package version has the subscriber package ID, 04tB00000009oZ3JBl, add the ID as the value of apvId.

`https://MyDomainName.lightning.force.com/packagingSetupUI/ipLanding.app?apvId=04tB00000009oZ3JBl`

Anyone with the URL and a valid login to a Salesforce org can install the package.

To install the package:

1. In a browser, enter the installation URL.
2. Enter your username and password for the Salesforce org in which you want to install the package, and then click **Login**.
3. If the package is protected by an installation key, enter the installation key.
4. For a default installation, click **Install**.

A message describes the progress. You receive a confirmation message when the installation is complete.

## Upgrade a Package Version

Are you introducing metadata changes to an existing package? You can use the CLI to upgrade one package version to another.

When you perform a package upgrade, here's what to expect for metadata changes.

- Metadata introduced in the new version is installed as part of the upgrade.
- If an upgraded component has the same API name as a component already in the target org, the component is overwritten with the changes.
- If a component in the upgrade was deleted from the target org, the component is re-created during the upgrade.
- Metadata that was removed in the new package version is also removed from the target org as part of the upgrade. Removed metadata is metadata not included in the current package version install, but present in the previous package version installed in the target org. If metadata is removed before the upgrade occurs, the upgrade proceeds normally. Some examples where metadata is deprecated and not deleted are:
  - User-entered data in custom objects and fields are deprecated and not deleted. Admins can export such data if necessary.
  - An object such as an Apex class is deprecated and not deleted if it is referenced in a Lightning component that is part of the package.
- In API version 45.0 and later (Salesforce CLI version 45.0.9 or later), you can specify what happens to removed metadata during package upgrade. Use the `force:package:install` command's `-t | --upgradetype` parameter, specifying one of these values:
  - `Delete` specifies to delete all removed components, except for custom objects and custom fields, that don't have dependencies.
  - `DeprecateOnly` specifies that all removed components must be marked deprecated. The removed metadata exists in the target org after package upgrade, but is shown in the UI as deprecated from the package. This option is useful when migrating metadata from one package to another.
  - `Mixed` (the default) specifies that some removed components are deleted, and other components are marked deprecated. For more information on hard-deleted components, see [Hard-Deleted Components in Unlocked Packages](#).

When you upgrade to a new package version, you choose whether to require successful compilation of all Apex in the org and package (`--apexcompile all`), or only the Apex in the package (`--apexcompile package`).

 **Note:** For package installs into production orgs, or any org that has [Apex Compile on Deploy enabled](#), the platform compiles all Apex in the org after the package install or upgrade operation completes. This approach assures that package installs and upgrades don't impact the performance of an org, and is done even if `--apexcompile package` is specified.

## Sample Script for Installing Packages with Dependencies

Use this sample script as a basis to create your own script to install packages with dependencies. This script contains a query that finds dependent packages and installs them in the correct dependency order.

### Sample Script

 **Note:** Be sure to replace the package version ID and scratch org user name with your own specific details.

```
#!/bin/bash

The execution of this script stops if a command or pipeline has an error.
For example, failure to install a dependent package will cause the script
to stop execution.

set -e

Specify a package version id (starts with 04t)
If you know the package alias but not the id, use force:package:version:list to find it.

PACKAGE=04tb000000NmnHIAS

Specify the user name of the subscriber org.

USER_NAME=test-bvdfz3m9tqdf@example.com

Specify the timeout in minutes for package installation.

WAIT_TIME=15

echo "Retrieving dependencies for package Id: \"$PACKAGE"

Execute soql query to retrieve package dependencies in json format.

RESULT_JSON=`sfdx force:data:soql:query -u $USER_NAME -t -q "SELECT Dependencies FROM
SubscriberPackageVersion WHERE Id='$PACKAGE'" --json`

Parse the json string using python to test whether the result json contains a list of
```

```
ids or not.

DEPENDENCIES=`echo $RESULT_JSON | python -c 'import sys, json; print
json.load(sys.stdin) ["result"] ["records"] [0] ["Dependencies"]'`

If the parsed dependencies is None, the package has no dependencies. Otherwise, parse
the result into a list of ids.

Then loop through the ids to install each of the dependent packages.

if [["$DEPENDENCIES" != 'None']]; then

 DEPENDENCIES=`echo $RESULT_JSON | python -c '

import sys, json

ids = json.load(sys.stdin) ["result"] ["records"] [0] ["Dependencies"] ["ids"]

dependencies = []

for id in ids:

 dependencies.append(id["subscriberPackageVersionId"])

print " ".join(dependencies)

`

 echo "The package you are installing depends on these packages (in correct dependency
order): "$DEPENDENCIES

 for id in $DEPENDENCIES

 do

 echo "Installing dependent package: "$id

 sfdx force:package:install --package $id -u $USER_NAME -w $WAIT_TIME --publishwait
10

 done

 else

 echo "The package has no dependencies"

 fi
```

```
After processing the dependencies, proceed to install the specified package.

echo "Installing package: \"$PACKAGE"

sfdx force:package:install --package $PACKAGE -u $USER_NAME -w $WAIT_TIME --publishwait
10

exit 0;
```

## Migrate Deprecated Metadata from Unlocked Packages

You can deprecate metadata in an unlocked package, move that metadata to a new package, and then install the new package in your production org.

As you create more unlocked packages, you can refactor your package and move metadata from one unlocked package to another unlocked package if necessary.

To move production metadata from package A to package B, follow these steps.

1. Identify the metadata to be moved from package A to package B.
2. Remove the metadata from package A, create a version, and release the package.
3. Add the metadata to package B, create a version, and release the package.
4. In your production org, upgrade package A.
5. In your production org, install package B.

Your metadata is now a part of package B in your production org.

## Uninstall a Package

You can uninstall a package from an org using Salesforce CLI or from the Setup UI. When you uninstall unlocked packages, all components in the package are deleted from the org.

To use the CLI to uninstall a package from the target org, authorize the Dev Hub org and run this command.

```
sfdx force:package:uninstall --package "Expense Manager@2.3.0-5"
```

You can also uninstall a package from the web browser. Open the Salesforce org where you installed the package.

```
sfdx force:org:open -u me@my.org
```

Then uninstall the package.

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

## Considerations on Uninstalling Packages

- If you're uninstalling a package that includes a custom object, all components on that custom object are also deleted. Deleted items include custom fields, validation rules, custom buttons, and links, 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.

## Transfer an Unlocked Package to a Different Dev Hub

---

You can transfer the ownership of an unlocked package from one Dev Hub org to another.



**Note:** This package transfer feature is available only to unlocked packages and second-generation managed packages. Dev Hub orgs aren't used with first-generation managed packages or unmanaged packages, so this feature doesn't apply to those package types.

### Request a Package Transfer to a Different Dev Hub

Start by logging a case with Salesforce Customer Support, and provide:

- Dev Hub org ID for the source org.
- Subscriber package ID of the package you're transferring. This ID starts with 033.

To verify the 033 ID of your package, run the `force:package:list` command with the `--verbose` flag on the source Dev Hub org.

- Dev Hub org ID for the destination org.
- (Optional) Namespace of the package being transferred. If the package is a no-namespace unlocked package, skip this step.

If you're transferring more than one package, file a separate case for each package.

After your case has been reviewed and approved, someone from Salesforce Customer Support will contact you to arrange a time to initiate the package transfer.



**Note:** For security reasons, package transfers between a Dev Hub located in Government Cloud and a Dev Hub located outside Government Cloud aren't permitted.

## Prepare to Transfer Your Package

Here's how you can help ensure a smooth package transfer.

- If the package you're transferring has a namespace, keep the namespace linked to the source Dev Hub. Before the package transfer, the namespace must be linked to both the source and destination Dev Hub orgs.
- Before the package transfer process is initiated, ensure all push upgrades or package version creation processes have completed.
- Delete package versions that are no longer needed.
- If specified, clear the package's Error Notification User using the `sfdx force:package:update` command. If you're transferring the package to a Dev Hub org you own, you can set the Error Notification User to a user in the destination Dev Hub after the package transfer is complete.

## During the Package Transfer Process

All push upgrades or package version creation processes must be complete before the package transfer process is initiated. Salesforce Customer Support will alert you about the date the package transfer will occur.

## After the Package Transfer Is Complete

Run `sfdx force:package:list` and verify that the package is no longer associated with your Dev Hub.

## Impact of Package Transfers on Package IDs

ID Type	ID starts with	After package transfer is complete
Subscriber Package ID	033	This ID remains the same.
Subscriber Package Version ID	04t	This ID remains the same.
Package ID	0Ho	The transferred package receives a new and unique package ID.

## Update Your Package Project File

Before you create new packages or package versions on your Dev Hub, update your `sfdx-project.json` file and remove all references to the transferred package from the package directory and package alias sections.

If you have packages in your Dev Hub that depend on the package that you're transferring, update the package dependency section in your `sfdx-project.json` file to explicitly specify the 04t ID of the transferred package that you depend on.

For example, if you transferred pkgA to a different Dev Hub, and your `sfdx-project.json` file lists the package dependency like this.

```
"dependencies": [
 {
```

## Unlocked Packages

## Take Ownership of an Unlocked Package Transferred from a Different Dev Hub

```
 "package": "pkgA"
 "versionNumber": "2.0.0.LATEST"
 }
]
```

Update the dependency to either specify the 04t ID of pkgA.

```
"dependencies": [
 {
 "package": "04tB000000UzH5IAK"
 }
]
```

Or specify the dependency using a package alias.

```
"dependencies": [
 {
 "package": "pkgA2.0.0-1"
 }
]
"packageAliases": {
 "pkgA2.0.0-1": "04tB000000UzH5IAK"
}
]
```

## What Package History Is Transferred?

Regardless of whether the package transfer occurred between two Dev Hub orgs you own, or the package was transferred externally to a Dev Hub you don't own, we transfer the package version history.

We transfer:

- Package name, namespace, type, and IDs. One exception is that the transferred package gets a new 0Ho ID.
- Package version info. This includes all the info that is typically displayed when you run the `force:package:version:list` or `force:package:version:report` command.

We don't transfer:

- Push upgrade history.
- Package version create requests.
- The username of the Dev Hub user who received Apex and other types of error notifications. This optional user is set using `--errornotificationusername`.

### [Take Ownership of an Unlocked Package Transferred from a Different Dev Hub](#)

You can take ownership of an unlocked package that is transferred from another Dev Hug org.

## Take Ownership of an Unlocked Package Transferred from a Different Dev Hub

You can take ownership of an unlocked package that is transferred from another Dev Hug org.

To initiate a package transfer from your Dev Hub org, see [Transfer an Unlocked Package to a Different Dev Hub](#).



**Note:** For security reasons, package transfers between a Dev Hub located in Government Cloud and a Dev Hub located outside Government Cloud aren't permitted.

## Receive a Package Transfer

Link the namespace of the package you're receiving to your Dev Hub org. See [Link a Namespace to a Dev Hub Org](#) in the *Salesforce DX Developer Guide*. If the package isn't associated with a namespace, skip this step.

## After the Package Transfer Is Complete

After the package transfer is complete, you'll be notified by Salesforce Customer Support.

To verify that the transferred package is associated with your Dev Hub, run `sfdx force:package:list`.

## Impact of Package Transfers on Package IDs

ID Type	ID starts with	After package transfer is complete
Subscriber Package ID	033	This ID remains the same.
Subscriber Package Version ID	04t	This ID remains the same.
Package ID	0Ho	The transferred package receives a new and unique package ID.

## Update Your Package Project File

Open and review the contents of the `sfdx-project.json` file associated with the transferred package.

Open and review the contents of any scratch org definition files associated with the transferred package. Definition files help in setting up your scratch orgs during development. Use the `--definitionfile` parameter to specify a definition file when you create a new package version.

If the package directories section lists additional packages that weren't transferred to you, remove those references from the `sfdx-project.json` file.

Next, review the package alias section of the `sfdx-project.json` file, and remove any references to package aliases that aren't associated with the package that was transferred.

Update the package alias of the transferred package to specify its 0Ho package ID.

## Before You Create a New Package Version

Similar to how you go about creating new package versions, you must update the `sfdx-project.json` file, and update the version number.

To designate a Dev Hub user to receive email notifications for unhandled Apex exceptions, and install, upgrade, or uninstall failures associated with your package, run the `sfdx force:package:update` command, and use the `--errornotificationusername` parameter.

## What Package History Is Transferred?

We transfer:

- Package name, namespace, type, and IDs. One exception is that the transferred package gets a new OHo ID.
- Package version info. This includes all the info that is typically displayed when you run the `force:package:version:list` or `force:package:version:report` command.

We don't transfer:

- Push upgrade history.
- Package version create requests.
- The username of the Dev Hub user who received Apex and other types of error notifications.

# CHAPTER 16 Continuous Integration

## In this chapter ...

- [Continuous Integration Using CircleCI](#)
- [Continuous Integration Using Jenkins](#)
- [Continuous Integration with Travis CI](#)
- [Sample CI Repos for Org Development Model](#)
- [Sample CI Repos for Package Development Model](#)

Continuous integration (CI) is a software development practice in which developers regularly integrate their code changes into a source code repository. To ensure that the new code does not introduce bugs, automated builds and tests run before or after developers check in their changes.

Many third-party CI tools are available for you to choose from. Salesforce DX easily integrates into these tools so that you can set up continuous integration for your Salesforce applications.

# Continuous Integration Using CircleCI

CircleCI is a commonly used integration tool that integrates with your existing version control system to push incremental updates to the environments you specify. CircleCI can be used as a cloud-based or on-premise tool. These instructions demonstrate how to use GitHub, CircleCI, and your Dev Hub org for continuous integration.

## Configure Your Environment for CircleCI

Before integrating your existing CircleCI framework, configure your Dev Hub org and CircleCI project.

### Connect CircleCI to Your DevHub

Authorize CircleCI to push content to your Dev Hub via a connected app.

## SEE ALSO:

[CircleCI](#)

[The sfdx-circleci-package Github Repo](#)

[The sfdx-circleci-org Github Repo](#)

## Configure Your Environment for CircleCI

Before integrating your existing CircleCI framework, configure your Dev Hub org and CircleCI project.

1. Set up your GitHub repository with CircleCI. You can follow the [sign-up steps on the CircleCI website](#) to access your code on GitHub.
2. [Install the Salesforce CLI](#), if you haven't already.
3. Follow [Authorize an Org Using the JWT Bearer Flow](#) for your Dev Hub org, if you haven't already.
4. Encrypt your server key.

a. First, generate a key and initialization vector (iv) to encrypt your `server.key` file locally. CircleCI uses the key and iv to decrypt your server key in the build environment.

Run the following command in the directory containing your `server.key` file. For the `<passphrase>` value, enter a word of your own choosing to create a unique key.

```
openssl enc -aes-256-cbc -k <passphrase> -P -md sha1 -nosalt
```

The key and iv value display in the output.

```
key=*****24B2
iv =*****DA58
```

- b. Note the key and iv values, you need them later.
- c. Encrypt the `server.key` file using the newly generated key and iv values. Run the following command in the directory containing your `server.key` file, replacing `<key>` and `<iv>` with the values from the previous step.

```
openssl enc -nosalt -aes-256-cbc -in server.key -out server.key.enc -base64 -K <key>
-iv <iv>
```

 **Note:** Use the key and iv values only once, and don't use them to encrypt more than the `server.key`. While you can reuse this pair to encrypt other things, it is considered a security violation to do so.

You generate a new key and iv value every time you run the command in step a. In other words, you can't regenerate the same pair. If you lose these values you must generate new ones and encrypt again.

Next, you'll store the key, iv, and contents of `server.key.enc` as protected environment variables in the CircleCI UI. These values are considered secret, so take the appropriate precautions to protect them.

## Connect CircleCI to Your DevHub

Authorize CircleCI to push content to your Dev Hub via a connected app.

1. Make sure that you have Salesforce CLI installed. Check by running `sfdx force --help` and confirm that you see the command output. If you don't have it installed, see [Install Salesforce CLI](#).
2. Confirm you can perform a JWT-based authorization from the directory containing your `server.key` file. Run the following command from the directory containing your `server.key` (replace `<your_consumer_key>` and `<your_username>` values where indicated).

```
sfdx auth:jwt:grant --clientid <your_consumer_key> --jwtkeyfile server.key --username <your_username> --setdefaultdevhubusername
```

3. Fork the [sfdx-circleci](#) repo into your GitHub account using the **Fork** link at the top of the page.
4. Create a local directory for this project and clone your forked repo locally into the new directory. Replace `<git_username>` with your own GitHub username.

```
git clone https://github.com/<git_username>/sfdx-circleci.git
```

5. Retrieve the generated consumer key from your JWT-Based Authorization connected app. From Setup, in the Quick Find box, enter `App`, and then select **App Manager**. Select **View** in the row-menu next to the connected app.
6. In the CircleCI UI, you see a project named `sfdx-circleci`. In the project settings, store the consumer key in a CircleCI environment variable named `HUB_CONSUMER_KEY`. For more information, see the CircleCI documentation [Setting an Environment Variable in a Project](#).
7. Store the username that you use to access your Dev Hub in a CircleCI environment variable named `HUB_SFDX_USER` using the CircleCI UI.
8. Store the key and iv values from Encrypt Your Server Key in CircleCI environment variables named `DECRYPTION_KEY` and `DECRYPTION_IV`, respectively. When you finish setting the environment variables, your project screen looks like the following image.

Name	Value	Remove
DECRIPTION_IV	xxxxDA58	X
DECRIPTION_KEY	xxxx24B2	X
HUB_CONSUMER_KEY	xxxxue7f	X
HUB_SFDFX_USER	xxxx.com	X

 **Note:** In the directory containing your `server.key` file, use the command `rm server.key` to remove the `server.key`. Never store keys or certificates in a public place.

You're ready to go! Now when you commit and push a change, your change kicks off a CircleCI build.

### Contributing to the Repository

If you find any issues or opportunities for improving this repository, fix them! Feel free to contribute to this project, [fork](#) this repository, and then change the content. After you make your changes, share them with the community by sending a pull request. See [How to send pull requests](#) for more information about contributing to GitHub projects.

### Reporting Issues

If you find any issues with this demo that you can't fix, feel free to report them in the [issues](#) section of this repository.

## Continuous Integration Using Jenkins

Jenkins is an open-source, extensible automation server for implementing continuous integration and continuous delivery. You can easily integrate Salesforce DX into the Jenkins framework to automate testing of Salesforce applications against scratch orgs.

To integrate Jenkins, we assume:

- You are familiar with how Jenkins works. You can configure and use Jenkins in many ways. We focus on integrating Salesforce DX into Jenkins multibranch pipelines.
- The computer on which the Jenkins server is running has access to your version control system and to the repository that contains your Salesforce application.

### Configure Your Environment for Jenkins

Before integrating your Dev Hub and scratch orgs into your existing Jenkins framework, configure your Jenkins environment. Our example assumes that you're working in a package development model.

### [Jenkinsfile Walkthrough](#)

The sample Jenkinsfile shows how to integrate your Dev Hub and scratch orgs into a Jenkins job. The sample uses Jenkins Multibranch Pipelines. Every Jenkins setup is different. This walkthrough describes one of the ways to automate testing of your Salesforce applications. The walkthrough highlights Salesforce CLI commands to create a scratch org, upload your code, and run your tests.

### [Sample Jenkinsfile](#)

A `Jenkinsfile` is a text file that contains the definition of a Jenkins Pipeline. This `Jenkinsfile` shows how to integrate Salesforce CLI commands to automate testing of your Salesforce applications using scratch orgs.

#### SEE ALSO:

[Jenkins](#)

[Pipeline-as-code with Multibranch Workflows in Jenkins](#)

## Configure Your Environment for Jenkins

Before integrating your Dev Hub and scratch orgs into your existing Jenkins framework, configure your Jenkins environment. Our example assumes that you’re working in a package development model.

1. In your Dev Hub org, [create a connected app](#) as described by the JWT-based authorization flow. This step includes obtaining or [creating a private key and digital certificate](#).

Make note of your consumer key (sometimes called a client ID) when you save the connected app. You need the consumer key to set up your Jenkins environment. Also have available the private key file used to sign the digital certificate.

2. On the computer that’s running the Jenkins server, do the following.

a. Download and install Salesforce CLI.

b. Store the private key file as a Jenkins Secret File using the [Jenkins Admin Credentials interface](#). Make note of the new entry’s ID.

You later reference this Credentials entry in your `Jenkinsfile`.

c. Set the following variables in your Jenkins environment.

- SF\_USERNAME—The username for the Dev Hub org, such as juliet.capulet@myenvhub.com.
- SF\_INSTANCE\_URL—The login URL of the Salesforce instance that hosts the Dev Hub org. The default is <https://login.salesforce.com>. We recommend that you update this value to the My Domain login URL for the Dev Hub org. You can find an org’s My Domain login URL on the My Domain page in Setup.
- SF\_CONSUMER\_KEY—The consumer key that was returned after you created a connected app in your Dev Hub org.
- SERVER\_KEY\_CREDENTIALS\_ID—The credentials ID for the private key file that you stored in the Jenkins Admin Credentials interface.
- PACKAGE\_NAME—The name of your package, such as My Package.
- PACKAGE\_VERSION—The version of your package, which starts with 04t.
- TEST\_LEVEL—The test level for your package, such as RunLocalTests.

The names for these environment variables are just suggestions. You can use any name as long as you specify it in the `Jenkinsfile`.

You can also optionally set the SFDX\_AUTOUPDATE\_DISABLE variable to `true` to disable auto-update of Salesforce CLI. CLI auto-update can interfere with the execution of a Jenkins job.

3. Set up your Salesforce DX project so that you can create a scratch org.

4. (Optional) Install the Custom Tools Plugin into your Jenkins console, and create a custom tool that references Salesforce CLI. The Jenkins walkthrough assumes that you created a custom tool named toolbelt in the `/usr/local/bin` directory, which is the directory in which Salesforce CLI is installed.

#### SEE ALSO:

[Authorize an Org Using the JWT Bearer Flow](#)  
[Salesforce CLI Setup Guide](#)  
[Jenkins: Credentials Binding Plugin](#)  
[Project Setup](#)

## Jenkinsfile Walkthrough

The sample Jenkinsfile shows how to integrate your Dev Hub and scratch orgs into a Jenkins job. The sample uses Jenkins Multibranch Pipelines. Every Jenkins setup is different. This walkthrough describes one of the ways to automate testing of your Salesforce applications. The walkthrough highlights Salesforce CLI commands to create a scratch org, upload your code, and run your tests.

This walkthrough relies on the `sfdx-jenkins-package` Jenkinsfile. We assume that you're familiar with the structure of the [Jenkinsfile](#), Jenkins Pipeline DSL, and the Groovy programming language. This walkthrough demonstrates implementing a Jenkins pipeline using Salesforce CLI and scratch orgs. See the [CLI Command Reference](#) regarding the commands used.

This workflow most closely corresponds to `Jenkinsfile` stages.

- [Define Variables](#)
- [Check Out the Source Code](#)
- [Wrap All Stages in a `withCredentials` Command](#)
- [Wrap All Stages in a `withEnv` Command](#)
- [Authorize Your Dev Hub Org and Create a Scratch Org](#)
- [Push Source and Assign a Permission Set](#)
- [Run Apex Tests](#)
- [Delete the Scratch Org](#)
- [Create a Package](#)
- [Create a Scratch Org and Display Info](#)
- [Install Package, Run Unit Tests, and Delete Scratch Org](#)

## Define Variables

Use the `def` keyword to define the variables required by Salesforce CLI commands. Assign each variable the corresponding environment variable that you previously set in your Jenkins environment.

```
def SF_CONSUMER_KEY=env.SF_CONSUMER_KEY
def SERVER_KEY_CREDENTIALS_ID=env.SERVER_KEY_CREDENTIALS_ID
def TEST_LEVEL='RunLocalTests'
def PACKAGE_NAME='0Ho1U00000CaUzSAK'
def PACKAGE_VERSION
def SF_INSTANCE_URL = env.SF_INSTANCE_URL ?: "https://MyDomainName.my.salesforce.com"
```

Define the `SF_USERNAME` variable, but don't set its value. You do that later.

```
def SF_USERNAME
```

Although not required, we assume that you used the Jenkins Global Tool Configuration to create the `toolbelt` custom tool that points to the CLI installation directory. In your `Jenkinsfile`, use the `tool` command to set the value of the `toolbelt` variable to this custom tool.

```
def toolbelt = tool 'toolbelt'
```

You can now reference the Salesforce CLI executable in the `Jenkinsfile` using  `${toolbelt} /sfdx`.

## Check Out the Source Code

Before testing your code, get the appropriate version or branch from your version control system (VCS) repository. In this example, we use the `checkout scm` Jenkins command. We assume that the Jenkins administrator has already configured the environment to access the correct VCS repository and check out the correct branch.

```
stage('checkout source') {
 // when running in multi-branch job, one must issue this command
 checkout scm
}
```

## Wrap All Stages in a `withCredentials` Command

You previously stored the JWT private key file as a Jenkins Secret File using the Credentials interface. Therefore, you must use the `withCredentials` command in the body of the `Jenkinsfile` to access the secret file. The `withCredentials` command lets you name a credential entry, which is then extracted from the credential store and provided to the enclosed code through a variable. When using `withCredentials`, put all stages within its code block.

This example stores the credential ID for the JWT key file in the variable `SERVER_KEY_CREDENTIALS_ID`. You defined the `SERVER_KEY_CREDENTIALS_ID` earlier and set it to its corresponding environment variable. The `withCredentials` command fetches the contents of the secret file from the credential store and places the contents in a temporary location. The location is stored in the variable `server_key_file`. You use the `server_key_file` variable with the `auth:jwt` command to specify the private key securely.

```
withCredentials([file(credentialsId: SERVER_KEY_CREDENTIALS_ID, variable: 'server_key_file')]

 # all stages will go here
}
```

## Wrap All Stages in a `withEnv` Command

When running Jenkins jobs, it's helpful to understand where files are being stored. There are two main directories to be mindful of: the workspace directory and the home directory. The workspace directory is unique to each job while the home directory is the same for all jobs.

The `withCredentials` command stores the JWT key file in the Jenkins workspace during the job. However, Salesforce CLI `auth` commands store authentication files in the home directory; these authentication files persist outside of the duration of the job.

This setup isn't a problem when you run a single job but can cause problems when you run multiple jobs. So, what happens if you run multiple jobs using the same Dev Hub or other Salesforce user? When the CLI tries to connect to the Dev Hub as the user you authenticated, it fails to refresh the token. Why? The CLI tries to use a JWT key file that no longer exists in the other workspace, regardless of the `withCredentials` for the current job.

If you set the home directory to match the workspace directory using `withEnv`, the authentication files are unique for each job. Creating unique auth files per job is also more secure because each job has access only to the auth files it creates.

When using `withEnv`, put all stages within its code block,

```
withEnv(["HOME=${env.WORKSPACE}"]) {
 # all stages will go here
}
```

 **Note:** If you don't use a pipeline or you run commands outside of a pipeline stage, add a home environment specification to your script: `export HOME=$WORKSPACE`.

## Authorize Your Dev Hub Org and Create a Scratch Org

This `sfdx-jenkins-package` example uses two stages: one stage to authorize the Dev Hub org and another stage to create a scratch org.

```
// -----
// Authorize the Dev Hub org with JWT key and give it an alias.
// -----

stage('Authorize DevHub') {
 rc = command "${toolbelt}/sfdx auth:jwt:grant --instanceurl ${SF_INSTANCE_URL} --clientid
${SF_CONSUMER_KEY} --username ${SF_USERNAME} --jwtkeyfile ${server_key_file}
--setdefaultdevhubusername --setalias HubOrg"
 if (rc != 0) {
 error 'Salesforce dev hub org authorization failed.'
 }
}

// -----
// Create new scratch org to test your code.
// -----

stage('Create Test Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:create --targetdevhubusername HubOrg
--setdefaultusername --definitionfile config/project-scratch-def.json --setalias ciorg
--wait 10 --durationdays 1"
 if (rc != 0) {
 error 'Salesforce test scratch org creation failed.'
 }
}
```

Use `auth:jwt:grant` to authorize your Dev Hub org.

You're required to run this step only one time, but we suggest you add it to your `Jenkinsfile` and authorize each time you run the Jenkins job. This way you're always sure that the Jenkins job isn't aborted due to lack of authorization. There's typically little harm in authorizing multiple times, but keep in mind that the API call limit for your scratch org's edition still applies.

Use the parameters of the `auth:jwt:grant` command to provide information about the Dev Hub org that you're authorizing. The values for the `--clientid`, `--username`, and `--instanceurl` parameters are the `SF_CONSUMER_KEY`, `HubOrg`, and `SF_INSTANCE_URL` environment variables you previously defined, respectively. The value of the `--jwtkeyfile` parameter is the `server_key_file` variable that you set in the previous section using the `withCredentials` command. The `--setdefaultdevhubusername` parameter specifies that this `HubOrg` is the default Dev Hub org for creating scratch orgs.

 **Note:** It's a best practice to have a unique authentication file for each Jenkins job using the `withEnv` wrapper. But it's possible to authorize a Dev Hub on your Jenkins machine instead. The advantage is that your authentication is set centrally on your machine.

for any Jenkins job you run. The disadvantage is security: Every job has access to all authenticated users whether you want them to or not.

If you do want to auth to your Dev Hub on your Jenkins machine, follow these steps:

- On the Jenkins machine as the Jenkins user, authorize to your Dev Hub using any of the `auth` commands.
- In your Jenkinsfile, remove the `withCredentials`, `withEnv`, and `auth:jwt:grant` statements.

Use the `force:org:create` CLI command to create a scratch org. In the example, the CLI command uses the `config/project-scratch-def.json` file (relative to the project directory) to create the scratch org. The `--json` parameter specifies the output as JSON format. The `--setDefaultUsername` parameter sets the new scratch org as the default.

The Groovy code that parses the JSON output of the `force:org:create` command extracts the username that was auto-generated as part of the org creation. This username, stored in the `SF_USERNAME` variable, is used with the CLI commands that push source, assign a permission set, and so on.

## Push Source and Assign a Permission Set

Let's populate your new scratch org with metadata. This example uses the `force:source:push` command to upload your source to the org. The source includes all the pieces that make up your Salesforce application: Apex classes and test classes, permission sets, layouts, triggers, custom objects, and so on.

```
// -----
// Push source to test scratch org.
// -----
```

```
stage('Push To Test Scratch Org') {
 rc = command "${toolbelt}/sfdx force:source:push --targetusername ciorg"
 if (rc != 0) {
 error 'Salesforce push to test scratch org failed.'
 }
}
```

Recall the `SF_USERNAME` variable that contains the auto-generated username that was output by the `force:org:create` command in an earlier stage. The code uses this variable as the argument to the `--targetusername` parameter to specify the username for the new scratch org.

The `force:source:push` command pushes all the Salesforce-related files that it finds in your project. Add a `.forceignore` file to your repository to list the files that you don't want pushed to the org.

## Run Apex Tests

Now that your source code and test source are pushed to the scratch org, run the `force:apex:test:run` command to run Apex tests.

```
// -----
// Run unit tests in test scratch org.
// -----
```

```
stage('Run Tests In Test Scratch Org') {
 rc = command "${toolbelt}/sfdx force:apex:test:run --targetusername ciorg --wait 10
--resultformat tap --codecoverage --testlevel ${TEST_LEVEL}"
 if (rc != 0) {
 error 'Salesforce unit test run in test scratch org failed.'
 }
}
```

```
 }
}
```

You can specify various parameters to the `force:apex:test:run` CLI command. In the example:

- The `--testlevel ${TEST_LEVEL}` option runs all tests in the scratch org, except tests that originate from installed managed packages. You can also specify `RunLocalTests` to run only local tests, `RunSpecifiedTests` to run only certain Apex tests or suites or `RunAllTestsInOrg` to run all tests in the org.
- The `--resultformat tap` option specifies that the command output is in Test Anything Protocol (TAP) format. The test results that are written to a file are still in JUnit and JSON formats.
- The `--targetusername ciorg` option specifies the username for accessing the scratch org (the value in `SF_USERNAME`).

The `force:apex:test:run` command writes its test results in JUnit format.

## Delete the Scratch Org

Salesforce reserves the right to delete a scratch org a specified number of days after it was created. You can also create a stage in your pipeline that uses `force:org:delete` to explicitly delete your scratch org when the tests complete. This cleanup ensures better management of your resources.

```
// -----
// Delete package install scratch org.
// -----

stage('Delete Package Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:delete --targetusername installorg --noprompt"

 if (rc != 0) {
 error 'Salesforce package install scratch org deletion failed.'
 }
}
```

## Create a Package

Now, let's create a package. If you're new to packaging, you can think about a package as a container that you fill with metadata. It contains a set of related features, customizations, and schema. You use packages to move metadata from one Salesforce org to another. After you create a package, add metadata and create a new package version.

```
// -----
// Create package version.
// -----

stage('Create Package Version') {
 if (isUnix()) {
 output = sh returnStdout: true, script: "${toolbelt}/sfdx
force:package:version:create --package ${PACKAGE_NAME} --installationkeybypass --wait 10
--json --targetdevhubusername HubOrg"
 } else {
 output = bat(returnStdout: true, script: "${toolbelt}/sfdx
force:package:version:create --package ${PACKAGE_NAME} --installationkeybypass --wait 10
--json --targetdevhubusername HubOrg").trim()
 output = output.readLines().drop(1).join(" ")
 }
}
```

```
// Wait 5 minutes for package replication.
sleep 300

def jsonSlurper = new JsonSlurperClassic()
def response = jsonSlurper.parseText(output)

PACKAGE_VERSION = response.result.SubscriberPackageVersionId

response = null

echo ${PACKAGE_VERSION}
}
```

## Create a Scratch Org and Display Info

Remember when you created a scratch org earlier? Now let's create a scratch org to install your package into, and display info about that scratch org.

```
// -----
// Create new scratch org to install package to.
// -----

stage('Create Package Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:create --targetdevhubusername HubOrg
--setdefaultusername --definitionfile config/project-scratch-def.json --setalias installorg
--wait 10 --durationdays 1"
 if (rc != 0) {
 error 'Salesforce package install scratch org creation failed.'
 }
}

// -----
// Display install scratch org info.
// -----

stage('Display Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:display --targetusername installorg"
 if (rc != 0) {
 error 'Salesforce install scratch org display failed.'
 }
}
```

## Install Package, Run Unit Tests, and Delete Scratch Org

To finish up, install your package in your scratch org, run unit tests, then delete the scratch org. That's it!

```
// -----
// Install package in scratch org.
// -----
```

```
stage('Install Package In Scratch Org') {
 rc = command "${toolbelt}/sfdx force:package:install --package ${PACKAGE_VERSION}
--targetusername installorg --wait 10"
 if (rc != 0) {
 error 'Salesforce package install failed.'
 }
}

// -----
// Run unit tests in package install scratch org.
// -----

stage('Run Tests In Package Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:apex:test:run --targetusername installorg
--resultformat tap --codecoverage --testlevel ${TEST_LEVEL} --wait 10"
 if (rc != 0) {
 error 'Salesforce unit test run in pacakge install scratch org failed.'
 }
}

// -----
// Delete package install scratch org.
// -----

stage('Delete Package Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:delete --targetusername installorg --noprompt"

 if (rc != 0) {
 error 'Salesforce package install scratch org deletion failed.'
 }
}
```

#### SEE ALSO:

[Sample Jenkinsfile](#)

[Pipeline-as-code with Multibranch Workflows in Jenkins](#)

[TAP: Test Anything Protocol](#)

[Configure Your Environment for Jenkins](#)

[Salesforce CLI Command Reference](#)

## Sample Jenkinsfile

A `Jenkinsfile` is a text file that contains the definition of a Jenkins Pipeline. This `Jenkinsfile` shows how to integrate Salesforce CLI commands to automate testing of your Salesforce applications using scratch orgs.

The [Jenkinsfile Walkthrough](#) topic uses this `sfdx-jenkins-package` `Jenkinsfile` as an example.

```
#!groovy

import groovy.json.JsonSlurperClassic
```

```
node {

 def SF_CONSUMER_KEY=env.SF_CONSUMER_KEY
 def SF_USERNAME=env.SF_USERNAME
 def SERVER_KEY_CREDENTIALS_ID=env.SERVER_KEY_CREDENTIALS_ID
 def TEST_LEVEL='RunLocalTests'
 def PACKAGE_NAME='0Ho1U000000CaUzSAK'
 def PACKAGE_VERSION
 def SF_INSTANCE_URL = env.SF_INSTANCE_URL ?: "https://login.salesforce.com"

 def toolbelt = tool 'toolbelt'

 // -----
 // Check out code from source control.
 // -----

 stage('checkout source') {
 checkout scm
 }

 // -----
 // Run all the enclosed stages with access to the Salesforce
 // JWT key credentials.
 // -----

 withEnv(["HOME=${env.WORKSPACE}"]) {

 withCredentials([file(credentialsId: SERVER_KEY_CREDENTIALS_ID, variable: 'server_key_file')]) {

 // -----
 // Authorize the Dev Hub org with JWT key and give it an alias.
 // -----

 stage('Authorize DevHub') {
 rc = command "${toolbelt}/sfdx auth:jwt:grant --instanceurl ${SF_INSTANCE_URL} --clientid ${SF_CONSUMER_KEY} --username ${SF_USERNAME} --jwtkeyfile ${server_key_file} --setdefaultdevhubusername --setalias HubOrg"
 if (rc != 0) {
 error 'Salesforce dev hub org authorization failed.'
 }
 }

 // -----
 // Create new scratch org to test your code.
 // -----

 stage('Create Test Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:create --targetdevhubusername HubOrg --setdefaultusername --definitionfile config/project-scratch-def.json --setalias
 }
 }
 }
}
```

```
ciorg --wait 10 --durationdays 1"
 if (rc != 0) {
 error 'Salesforce test scratch org creation failed.'
 }
}

// -----
// Display test scratch org info.
// -----

stage('Display Test Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:display --targetusername ciorg"
 if (rc != 0) {
 error 'Salesforce test scratch org display failed.'
 }
}

// -----
// Push source to test scratch org.
// -----

stage('Push To Test Scratch Org') {
 rc = command "${toolbelt}/sfdx force:source:push --targetusername ciorg"
 if (rc != 0) {
 error 'Salesforce push to test scratch org failed.'
 }
}

// -----
// Run unit tests in test scratch org.
// -----

stage('Run Tests In Test Scratch Org') {
 rc = command "${toolbelt}/sfdx force:apex:test:run --targetusername ciorg
--wait 10 --resultformat tap --codecoverage --testlevel ${TEST_LEVEL}"
 if (rc != 0) {
 error 'Salesforce unit test run in test scratch org failed.'
 }
}

// -----
// Delete test scratch org.
// -----

stage('Delete Test Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:delete --targetusername ciorg
--noprompt"
 if (rc != 0) {
 error 'Salesforce test scratch org deletion failed.'
 }
}
```

```
}

// -----
// Create package version.
// -----

stage('Create Package Version') {
 if (isUnix()) {
 output = sh returnStdout: true, script: "${toolbelt}/sfdx
force:package:version:create --package ${PACKAGE_NAME} --installationkeybypass --wait 10
--json --targetdevhubusername HubOrg"
 } else {
 output = bat(returnStdout: true, script: "${toolbelt}/sfdx
force:package:version:create --package ${PACKAGE_NAME} --installationkeybypass --wait 10
--json --targetdevhubusername HubOrg").trim()
 output = output.readLines().drop(1).join(" ")
 }

 // Wait 5 minutes for package replication.
 sleep 300

 def jsonSlurper = new JsonSlurperClassic()
 def response = jsonSlurper.parseText(output)

 PACKAGE_VERSION = response.result.SubscriberPackageVersionId

 response = null

 echo ${PACKAGE_VERSION}
}
```

```
// -----
// Create new scratch org to install package to.
// -----

stage('Create Package Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:create --targetdevhubusername
HubOrg --setdefaultusername --definitionfile config/project-scratch-def.json --setalias
installorg --wait 10 --durationdays 1"
 if (rc != 0) {
 error 'Salesforce package install scratch org creation failed.'
 }
}

// -----
// Display install scratch org info.
// -----

stage('Display Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:display --targetusername installorg"
```

```

 if (rc != 0) {
 error 'Salesforce install scratch org display failed.'
 }
 }

 // -----
 // Install package in scratch org.
 // -----

 stage('Install Package In Scratch Org') {
 rc = command "${toolbelt}/sfdx force:package:install --package
${PACKAGE_VERSION} --targetusername installorg --wait 10"
 if (rc != 0) {
 error 'Salesforce package install failed.'
 }
 }

 // -----
 // Run unit tests in package install scratch org.
 // -----

 stage('Run Tests In Package Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:apex:test:run --targetusername
installorg --resultformat tap --codecoverage --testlevel ${TEST_LEVEL} --wait 10"
 if (rc != 0) {
 error 'Salesforce unit test run in pacakge install scratch org failed.'
 }
 }

 // -----
 // Delete package install scratch org.
 // -----

 stage('Delete Package Install Scratch Org') {
 rc = command "${toolbelt}/sfdx force:org:delete --targetusername installorg
--noprompt"
 if (rc != 0) {
 error 'Salesforce package install scratch org deletion failed.'
 }
 }
}

def command(script) {
 if (isUnix()) {
 return sh(returnStatus: true, script: script);
 } else {
 return bat(returnStatus: true, script: script);
 }
}

```

```
 }
}
```

SEE ALSO:

[Jenkinsfile Walkthrough](#)

## Continuous Integration with Travis CI

Travis CI is a cloud-based continuous integration (CI) service for building and testing software projects hosted on GitHub.

For help with setting up Travis CI, see:

- Sample [Travis CI repo](#) for Org Development model
- Sample [Travis CI repo](#) for Package Development model

SEE ALSO:

[sfdx-travisci Sample GitHub Repo](#)

[Travis CI](#)

## Sample CI Repos for Org Development Model

Get started quickly with CI by cloning a sample repository from your vendor of choice. Each repo has a sample configuration file and a comprehensive `README.md` with step-by-step information.

These sample repositories support the org development model. This model uses Salesforce CLI, a source control system, and sandboxes during the application life cycle. To determine if this model is right for you, head over and earn your badge by completing the [Org Development Model](#) module.

Vendor	Link to GitHub Repository
AppVeyor	<a href="https://github.com/forcedotcom/sfdx-appveyor-org">https://github.com/forcedotcom/sfdx-appveyor-org</a>
Bamboo	<a href="https://github.com/forcedotcom/sfdx-bamboo-org">https://github.com/forcedotcom/sfdx-bamboo-org</a>
Bitbucket	<a href="https://github.com/forcedotcom/sfdx-bitbucket-org">https://github.com/forcedotcom/sfdx-bitbucket-org</a>
CircleCI	<a href="https://github.com/forcedotcom/sfdx-circleci-org">https://github.com/forcedotcom/sfdx-circleci-org</a>
GitLab	<a href="https://github.com/forcedotcom/sfdx-gitlab-org">https://github.com/forcedotcom/sfdx-gitlab-org</a>
Jenkins	<a href="https://github.com/forcedotcom/sfdx-jenkins-org">https://github.com/forcedotcom/sfdx-jenkins-org</a>
TravisCI	<a href="https://github.com/forcedotcom/sfdx-travisci-org">https://github.com/forcedotcom/sfdx-travisci-org</a>

## Sample CI Repos for Package Development Model

Get started quickly with CI by cloning a sample repository from your vendor of choice. Each repo has a sample configuration file and a comprehensive `README.md` with step-by-step information.

These sample repositories support the package development model. This model uses Salesforce CLI, a source control system, scratch orgs for development, and sandboxes for testing and staging. To determine if this model is right for you, head over and earn your badge by completing the [Package Development Model](#) module.

Vendor	Link to GitHub Repository
AppVeyor	<a href="https://github.com/forcedotcom/sfdx-appveyor-package">https://github.com/forcedotcom/sfdx-appveyor-package</a>
Bamboo	<a href="https://github.com/forcedotcom/sfdx-bamboo-package">https://github.com/forcedotcom/sfdx-bamboo-package</a>
Bitbucket	<a href="https://github.com/forcedotcom/sfdx-bitbucket-package">https://github.com/forcedotcom/sfdx-bitbucket-package</a>
CircleCI	<a href="https://github.com/forcedotcom/sfdx-circleci-package">https://github.com/forcedotcom/sfdx-circleci-package</a>
GitLab	<a href="https://github.com/forcedotcom/sfdx-gitlab-package">https://github.com/forcedotcom/sfdx-gitlab-package</a> CI/CD template for Salesforce/Apex apps: <a href="https://gitlab.com/sfdx/sfdx-cicd-template">https://gitlab.com/sfdx/sfdx-cicd-template</a>
Jenkins	<a href="https://github.com/forcedotcom/sfdx-jenkins-package">https://github.com/forcedotcom/sfdx-jenkins-package</a>
TravisCI	<a href="https://github.com/forcedotcom/sfdx-travisci-package">https://github.com/forcedotcom/sfdx-travisci-package</a>

# CHAPTER 17 Troubleshoot Salesforce DX

## In this chapter ...

- [Error: API Version Mismatch](#)
- [CLI Version Information](#)
- [Run CLI Commands on macOS Sierra \(Version 10.12\)](#)
- [Error: No defaultdevhubusername org found](#)
- [Unable to Work After Failed Org Authorization](#)
- [Error: Lightning Experience-Enabled Custom Domain Is Unavailable](#)

This guide is a work in progress. Log in to the Salesforce Trailblazer Community and let us know if you find a solution that would help other users so that we can incorporate it.

## SEE ALSO:

[Salesforce Trailblazer Community](#)

## Error: API Version Mismatch

If you update Salesforce CLI and try to push source from your local DX project to a scratch org, you see an API version error.

```
sfdx force:source:push
ERROR running force:source:push: The configured apiVersion 51.0 is not supported for this
org. The max apiVersion is 50.0
```

What happened?

**Answer:** Your locally configured `apiVersion` is greater than your org's supported max `apiVersion`. To troubleshoot, first run this command to determine if your local `apiVersion` is overridden:

```
sfdx config:list
```

To resolve the error for a specific project, set the `apiVersion` locally:

```
sfdx config:set apiVersion=50.0
```

To resolve the error for all Salesforce DX projects, set the `apiVersion` globally:

```
sfdx config:set apiVersion=50.0 -g
```

To override the API version for a single CLI command execution, use the `--apiversion` parameter:

```
sfdx force:source:push --apiversion=50.0
```

 **Note:** Not all commands support the `--apiversion` parameter.

## CLI Version Information

Use these commands to view version information about Salesforce CLI.

```
sfdx plugins --core // Version of the CLI and all installed plug-ins
sfdx --version // CLI version
```

## Run CLI Commands on macOS Sierra (Version 10.12)

Some users who upgrade to macOS Sierra can't execute CLI commands. This is a general problem and not isolated to Salesforce DX. To resolve the issue, reinstall your Xcode developer tools.

Execute this command in Terminal:

```
xcode-select --install
```

If you still can't execute CLI commands, download the **Command Line Tools (macOS Sierra) for Xcode 8** package from the Apple Developer website.

SEE ALSO:

[Apple Developer Downloads](#)

[Stack Overflow: Command Line Tools bash \(git\) not working - macOS Sierra Final Release Candidate](#)

## Error: No defaultdevhubusername org found

Let's say you successfully authorize a Dev Hub org using the `--setdefaultdevhubusername` parameter. The username associated with the org is your default Dev Hub username. You then successfully create a scratch org without using the `--targetdevhubusername` parameter.

But when you try to create a scratch org another time using the same CLI command, you get this error:

```
Unable to invoke command. name: NoOrgFound message: No defaultdevhubusername org found
```

What happened?

**Answer:** You are no longer in the directory where you ran the authorization command. The directory from which you use the `--setdefaultdevhubusername` parameter matters.

If you run the authorization command from the root of your project directory, the `defaultdevhubusername` config value is set locally. The value applies only when you run the command from the same project directory. If you change to a different directory and run `force:org:create`, the local setting of the default Dev Hub org no longer applies and you get an error.

Solve the problem by doing one of the following.

- Set `defaultdevhubusername` globally so that you can run `force:org:create` from any directory.

```
sfdx config:set defaultdevhubusername=<devhubusername> --global
```

- Run `force:org:create` from the same project directory where you authorized your Dev Hub org.
- Use the `--targetdevhubusername` parameter with `force:org:create` to run it from any directory.

```
sfdx force:org:create --definitionfile <file> --targetdevhubusername <devhubusername> --setalias my-scratch-org
```

- To check whether you've set configuration values globally or locally, use this command.

```
sfdx config:list
```

### SEE ALSO:

[How Salesforce Developer Experience Changes the Way You Work](#)

## Unable to Work After Failed Org Authorization

Sometimes you try to authorize a Dev Hub org or a scratch org using the Salesforce CLI or an IDE, but you don't successfully log in to the org. The port remains open for the stray authorization process, and you can't use the CLI or IDE. To proceed, end the process manually.

### macOS or Linux

To recover from a failed org authorization on macOS or Linux, use a terminal to kill the process running on port 1717.

1. From a terminal, run:

```
lsof -i tcp:1717
```

2. In the results, find the ID for the process that's using the port.

3. Run:

```
kill -9 <the process ID>
```

## Windows

To recover from a failed org authorization on Windows, use the Task Manager to end the Node process.

1. Press Ctrl+Alt+Delete, then click **Task Manager**.
2. Select the **Process** tab.
3. Find the process named **Node**.



**Note:** If you're a Node.js developer, you might have several running processes with this name.

4. Select the process that you want to end, and then click **End Process**.

## Error: Lightning Experience-Enabled Custom Domain Is Unavailable

If you create a scratch org with `force:org:create`, and then immediately try to use it, you sometimes get an error after waiting a few minutes for the command to finish.

For example, if you try to open the new scratch org in a browser with `force:org:open`, you might get this error:

```
Waiting to resolve the Lightning Experience-enabled custom domain...
ERROR running force:org:open: The Lightning Experience-enabled custom domain is unavailable.
```

The error occurs because it takes a few minutes for the Lightning Experience-enabled custom domain to internally resolve.

When using the CLI interactively, wait a few more minutes and run the command again. In a CI environment, however, you can avoid the error altogether by changing how long the CLI itself waits.

By default, the CLI waits 240 seconds (4 minutes) for the custom domain to become available. You can configure the CLI to wait longer by setting the `SFDX_DOMAIN_RETRY` environment variable to the number of seconds you want it to wait. For example, to wait 5 minutes (300 seconds):

```
export SFDX_DOMAIN_RETRY=300
```

If you want the CLI to bypass the custom domain check entirely, set `SFDX_DOMAIN_RETRY` to 0.

## CHAPTER 18 Limitations for Salesforce DX

Here are some known issues you could run into while using Salesforce DX.

For the latest known issues, visit the Trailblazer Community's [Known Issues](#) page.

### Salesforce CLI

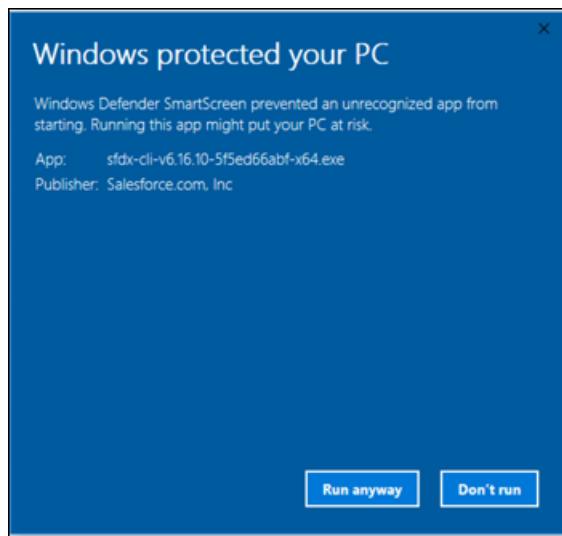
#### Authorization Fails If Using `auth:web:login` with Client Secret

**Description:** If you run `auth:web:login` with a client ID and client secret, you can't use Salesforce CLI to issue commands to the scratch org because the authorization file isn't properly created.

**Workaround:** Use the web-based flow without client ID and client secret, or use the JWT-based flow to authorize to the org. See [Authorization](#) in the *Salesforce DX Developer Guide* for instructions on Dev Hub and scratch org authorization methods.

#### Windows Defender Suspends CLI Installation

**Description:** When you are installing the Salesforce CLI on Windows, you see a Windows Defender warning. This message is expected because we updated the installer's code signing certificate.



**Workaround:** To ignore this message, click **Run anyway**.

#### Can't Import Record Types Using the Salesforce CLI

**Description:** We don't support RecordType when running the `data:tree:import` command.

**Workaround:** None.

### Limited Support for Shell Environments on Windows

**Description:** Salesforce CLI is tested on the Command Prompt (cmd.exe) and Powershell. There are known issues in the Cygwin and Min-GW environments, and with Windows Subsystem for Linux (WSL). These environments might be tested and supported in a future release. For now, use a supported shell instead.

**Workaround:** None.

### The `force:apex:test:run` Command Doesn't Finish Executing

**Description:** In certain situations, the `force:apex:test:run` command doesn't finish executing. Examples of these situations include a compile error in the Apex test or an Apex test triggering a pre-compile when another is in progress.

**Workaround:** Stop the command execution by typing control-C. If the command is part of a continuous integration (CI) job, try setting the environment variable `SFDX_PRECOMPILE_DISABLE=true`.

## Dev Hub and Scratch Orgs

---

### Salesforce CLI Sometimes Doesn't Recognize Scratch Orgs with Communities

**Description:** Sometimes, but not in all cases, the Salesforce CLI doesn't acknowledge the creation of scratch orgs with the Communities feature. You can't open the scratch org using the CLI, even though the scratch org is listed in Dev Hub.

**Workaround:** You can try this workaround, although it doesn't fix the issue in all cases. Delete the scratch org in Dev Hub, then create a new scratch org using the CLI. Deleting and recreating scratch orgs counts against your daily scratch org limits.

### Error Occurs If You Pull a Community and Deploy It

**Description:** The error occurs because the scratch org doesn't have the required guest license.

**Workaround:** In your scratch org definition file, if you specify the Communities feature, also specify the Sites feature.

## Source Management

---

### ERROR: No Results Found for `force:source:status` After Deleting a Custom Label

**Description:** The `force:source:status` command returns a `No Results Found` error after you delete a custom label in a scratch org.

**Workaround:** Option #1: If you have only one or two scratch orgs and you can easily identify the affected scratch org by its generated username, use this workaround. In the `Your DX project/.sfdx/org` directory, delete only the folder of the affected scratch org.

Option #2: If you have several scratch orgs associated with your DX project and you don't know which scratch org's local data to delete, use this workaround. Delete the `Your DX project/.sfdx/org` directory. This directory contains source tracking information for all scratch orgs related to the project. When you run the next source-tracking command for this or another scratch org (`source:push`, `source:pull`, or `source:status`), the CLI reconstructs the source tracking information for that org.

After you delete the directory (after option #1 or option #2), run `force:source:status` again.

**ERROR: Entity of type 'RecordType' named 'Account.PersonAccount' cannot be found**

**Description:** Although you can turn on Person Accounts in your scratch org by adding the feature to your scratch org definition, running `source:push` or `source:pull` results in an error,

**Workaround:** None.

**force:source:convert Doesn't Add Post-Install Scripts to package.xml**

**Description:** If you run `force:source:convert`, `package.xml` does not include the post install script.

**Workaround:** To fix this issue, choose one of these methods:

- Manually add the `<postInstallClass>` element to the `package.xml` in the metadata directory that `force:source:convert` produces
- Manually add the element to the package in the release org or org to which you are deploying the package.

**Must Manually Enable Feed Tracking in an Object's Metadata File**

**Description:** If you enable feed tracking on a standard or custom object, then run `force:source:pull`, feed tracking doesn't get enabled.

**Workaround:** In your Salesforce DX project, manually enable feed tracking on the standard or custom object in its metadata file (`-meta.xml`) by adding `<enableFeeds>true</enableFeeds>`.

**Unable to Push Lookup Filters to a Scratch Org**

**Description:** When you execute the `force:source:push` command to push the source of a relationship field that has a lookup filter, you sometimes get the following error:

`duplicate value found: <unknown> duplicates value on record with id: <unknown> at line num, col num.`

**Workaround:** None.

## Deployment

---

**Compile on Deploy Can Increase Deployment Times in Scratch Orgs**

**Description:** If your deployment times for Apex code are slow, your scratch org might have the `enableCompileOnDeploy` setting set to `true`.

**Workaround:** To turn it off, set it to `false` (the default) or delete the setting from the scratch org definition.

```
{
 "orgName": "ekapner Company",
 "edition": "Developer",
 "features": [],
 "settings": {
 "lightningExperienceSettings": {
 "enableS1DesktopEnabled": true
 },
 "apexSettings": {
 "enableCompileOnDeploy": false
 }
 }
}
```

## Managed First-Generation Packages

---

### When You Install a Package in a Scratch Org, No Tests Are Performed

**Description:** If you include tests as part of your continuous integration process, those tests don't run when you install a package in a scratch org.

**Workaround:** You can manually execute tests after the package is installed.

### New Terminology in CLI for Managed Package Password

**Description:** When you use the CLI to add an installation key to a package version or to install a key-protected package version, the parameter name of the key is `--installationkey`. When you view a managed package version in the Salesforce user interface, the same package attribute is called "Password". In the API, the corresponding field name, "password", is unchanged.

**Workaround:** None.

## Managed Second-Generation Packages

---

### Unable to Specify a Patch Version for Managed Packages

**Description:** The four-part package version number includes a patch segment, defined as major.minor.patch.build. However, you can't create a patch for a second-generation managed package. Package creation fails if you set a patch number in the package descriptor. We plan to provide this functionality for managed packages in the Winter '20 release.

**Workaround:** Always set the patch segment of the version number, to 0. For example, 1.2.0.1 is valid but 1.2.1.1 is not.

### Protected Custom Metadata and Custom Settings are Visible to Developers in a Scratch Org If Installed Packages Share a Namespace

**Description:** Use caution when you store secrets in your second-generation packages using protected custom metadata or protected custom settings. You can create multiple second-generation packages with the same namespace. However, when you install these packages in a scratch org, these secrets are visible to any of your developers that are working in a scratch org with a shared namespace. In the future, we might add a "package-protected" keyword to prevent access to package secrets in these situations.

**Workaround:** None.

## Unlocked Packages

---

### Protected Custom Metadata and Custom Settings are Visible to Developers in a Scratch Org If Installed Packages Share a Namespace

**Description:** Use caution when you store secrets in your unlocked packages using protected custom metadata or protected custom settings. You can create multiple unlocked packages with the same namespace. However, when you install these packages in a scratch org, these secrets are visible to any of your developers that are working in a scratch org with a shared namespace. In the future, we might add a "package-protected" keyword to prevent access to package secrets in these situations.

**Workaround:** None.