Ant Migration Tool Guide

Version 63.0, Spring '25

Last updated: December 27, 2024

CONTENTS

CHAPTER 1 Ant Migration Tool

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The Ant Migration Tool is a Java/Ant-based command-line utility for moving metadata between a local directory and a Salesforce org. The Ant Migration Tool is especially useful in the following scenarios.

- Development projects for which you need to populate a test environment with a lot of setup changes—Making these changes using a web interface can take a long time.
- Multistage release processes—A typical development process requires iterative building, testing, and staging before releasing to a production environment. Scripted retrieval and deployment of components can make this process much more efficient.
- Repetitive deployment using the same parameters—You can retrieve all the metadata in your organization, make changes, and deploy a subset of components. If you need to repeat this process, it's as simple as calling the same deployment target again.
- When migrating from stage to production is done by IT—Anyone that prefers deploying in a scripting environment will find the Ant Migration Tool a familiar process.
- Scheduling batch deployments—You can schedule a deployment for midnight to not disrupt users. Or you can pull down changes to your Developer Edition org every day.

Understanding Metadata API

Metadata API contains a set of objects that manage setup and customization information (metadata) for your organizations, and the SOAP calls that manipulate those objects. With Metadata API you can:

- Work with setup configuration as XML metadata files
- Migrate configuration changes between organizations
- Create your own tools for managing organization and application metadata

Although you can write your own client applications for using Metadata API SOAP calls, Salesforce provides the Ant Migration Tool to retrieve and deploy Apex and metadata.

Understanding Package and Directory Structure

Metadata API functions in a package-centric manner. Components can be in one or more packages, or in no package. Packages can be local (created in your Salesforce org) or installed from Salesforce AppExchange. Whenever the Ant Migration Tool retrieves a set of components, that set is limited to what's in a single package or what's in no package at all. There are three kinds of packages.

- Unpackaged—Components that live natively in your organization, such as standard objects, go in the *unpackaged* package.
- Unmanaged package—Unmanaged packages are typically used to distribute open-source projects or application templates to provide developers with the basic building blocks for an application. After the components are installed from an unmanaged package, the components can be edited in the organization they're installed in. The developer who created and uploaded the unmanaged package has no control over the installed components, and can't change or upgrade them. Don't use unmanaged packages to migrate components from a sandbox to production org. Instead, use Change Sets.

- Managed package—A collection of application components that is posted as a unit on the AppExchange and associated with a namespace and possibly a License Management Organization. To support upgrades, a package must be managed. An organization can create a single managed package that can be downloaded and installed by many different organizations. Managed packages differ from unmanaged packages by having some locked components, allowing the managed package to be upgraded later. Unmanaged packages do not include locked components and cannot be upgraded. In addition, managed packages obfuscate certain components (like Apex) on subscribing organizations to protect the intellectual property of the developer.

CHAPTER 2 Installing the Ant Migration Tool

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

Before you install the Ant Migration Tool you will need Java and Ant installed on your local machine. Then you can download the Ant Migration Tool from a Salesforce organization.

1. Install Java and Ant, as described in Prerequisites for Using the Ant Migration Tool.
2. Download the Ant Migration Tool, as described in Install the Ant Migration Tool.

Prerequisites for Using the Ant Migration Tool

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

Before you can use the Ant Migration Tool, Java and Ant must be installed and configured correctly. If you already have Java and Ant on your computer, you don't need to install them, so first verify the installation from a command prompt.

Java

Java version 11 or later is recommended for better security and for the latest TLS security protocols.

To check the version of Java that's installed on your system:

1. Open a command prompt.
2. At the prompt, type `java -version` and press Enter.

If you have Java version 11, the output looks something like the following.

```
openjdk version "11.0.8" 2020-07-14
OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.8+10)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.8+10, mixed mode)
```

Note: Ant Migration Tool version 51.0 and later requires Java version 11 or later.

If working with Ant Migration Tool version 36.0 to 50.0, for enhanced security, we recommend Java 7 or later and a recent version of the Ant Migration Tool (version 36.0 or later). Starting with version 36.0, the Ant Migration Tool uses TLS 1.2 for secure communications with Salesforce when it detects Java version 7 (1.7). The tool explicitly enables TLS 1.1 and 1.2 for Java 7. If you're using Java 8 (1.8), TLS 1.2 is used. For Java version 6, TLS 1.0 is used, which is no longer supported by Salesforce.

Alternatively, if you're using Java 7, instead of upgrading your Ant Migration Tool to version 36.0 or later, you can add the following to your `ANT_OPTS` environment variable:

```
-Dhttps.protocols=TLSv1.1,TLSv1.2
```

This setting also enforces TLS 1.1 and 1.2 for any other Ant tools on your local system.

To install Java, go to http://www.oracle.com/technetwork/java/javase/downloads/index.html and get the latest version of the Java JDK. When you're finished with the installation, verify by typing `java -version` at a command prompt.

Ant

1. Open a command prompt.
2. At the prompt, type `ant -version` and press Enter.

The output looks something like the following:

```
Apache Ant version 1.7.0 compiled on December 13 2006
```

If the Ant version is 1.5.x or earlier, download the latest version of Ant.

Note: Even if you have Ant installed, you sometimes still need to put the `bin` directory on your path. On a Windows operating system, you sometimes also need to set the `ANT_HOME` and `JAVA_HOME` environment variables as follows.

To install and configure Ant:

1. Download Apache Ant version 1.6 or later to a directory of your choice: `http://ant.apache.org/bindownload.cgi`. This directory known as ANT_HOME. When the files are on your computer, no further installation is required.
2. Add the `bin` directory to your path. (Only the `bin` and `lib` directories are required to run Ant.)
3. If you are using a Windows operation system, create an `ANT_HOME` environment variable and set the value to where you have installed Ant. Also create a `JAVA_HOME` environment variable and set the value to the location of your JDK.

For more information, see `http://ant.apache.org/manual/install.html`.

Install the Ant Migration Tool

Follow these steps to download and install the Ant Migration Tool.

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

If you don't have Ant installed, see Prerequisites for Using the Ant Migration Tool.

1. Download the .zip file of the Winter '24 Ant Migration Tool. The download link doesn't require authentication to Salesforce. If you're logged in to Salesforce, we recommend that you log out before accessing the link in your browser.
2. Save the .zip file locally, and extract the contents to the directory of your choice.

Note: The Ant Migration Tool uses the `ant-salesforce.jar` file that's in the distribution .zip file. If you installed a previous version of the tool and copied `ant-salesforce.jar` to the Ant `lib` directory, delete the previous jar file. The `lib` directory is located in the root folder of your Ant installation. It's not necessary to copy the new jar file to the Ant `lib` directory.

If you plan to run the tool from a directory other than its installation directory, modify the `build.xml` file to indicate the location of the `ant-salesforce.jar` file. Update the `location` attribute on `<pathelement>` in `build.xml` to point to `ant-salesforce.jar` in the installation directory.

When you extract the Ant Migration Tool .zip files, these folders and files are written to the location that you specified.

- A `Readme.html` file that explains how to use the tools
- A Jar file containing the ant task: `ant-salesforce.jar`
- A sample folder containing:

- A `codepkg\classes` folder that contains `SampleDeployClass.cls` and `SampleFailingTestClass.cls`
- A `codepkg\triggers` folder that contains `SampleAccountTrigger.trigger`
- A `mypkg\objects` folder that contains the custom objects used in the examples
- A `removecodepkg` folder that contains XML files for removing the examples from your organization
- A sample `build.properties` file that you must edit, specifying your credentials, in order to run the sample ant tasks in `build.xml`
- A sample `build.xml` file that exercises the `deploy` and `retrieve` API calls

CHAPTER 3 Using the Ant Migration Tool

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The Ant Migration Tool is a Java/Ant-based command-line utility for moving metadata between a local directory and a Salesforce organization. You can use the Ant Migration Tool to retrieve components, create scripted deployment, and repeat deployment patterns.

The general procedure you will follow when using the Ant Migration Tool to copy metadata from one Salesforce organization to another is:

1. Enter credentials and connection information for source Salesforce organization in `build.properties`
2. Create retrieve targets in `build.xml`
3. Construct a project manifest in `package.xml`
4. Run the Ant Migration Tool to retrieve metadata files from Salesforce
5. Enter credentials and connection information for destination Salesforce organization in `build.properties`
6. Run the Ant Migration Tool to deploy metadata files or deletions to Salesforce

Entering Salesforce Connection Information

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

To retrieve or deploy metadata components, you need to edit `build.properties` to point to a Salesforce org.

1. Go to the location where you extracted the Ant Migration Tool files and open the `sample` subdirectory.
2. Open `build.properties` in a text editor and do either of the following.
   - To use a username and password for login, substitute a valid Salesforce username and password. If you're using a security token, paste the 25-digit token value at the end of your password.
   - To use an active Salesforce session for login, uncomment the `sf.sessionId` property and substitute a valid session ID. Also, make sure to comment out the `sf.username` and `sf.password` properties.
   - To use an OAuth access token for login, uncomment the `sf.sessionId` property and supply the access token. Also, make sure to comment out the `sf.username` and `sf.password` properties.

Parameter	Value
`sf.username`	The Salesforce username for login. The username associated with this connection must have the Modify Metadata through Metadata API Functions permission. When connecting to a sandbox instance, your sandbox name is appended to your username. For example, if your production username is `foo@salesforce.com`, and one of your sandboxes is called `bar`, your sandbox username is `foo@salesforce.com.bar`.

Parameter	Value
`sf.password`	The password you use to log in to the org associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`sf.serverurl`	The salesforce server URL. Use `https://login.salesforce.com` to connect to a production or Developer Edition org. To connect to a sandbox instance, change this to `https://test.salesforce.com`.
`sf.sessionId`	The ID of an active Salesforce session or the OAuth access token. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging in to an existing session instead of creating a new session. Alternatively, use an access token for OAuth authentication. For more information, see Authenticating Apps with OAuth in the Salesforce Help.

Note: In the `build.properties` file, you can specify values for either the `sf.username` and `sf.password` property pair, or the `sf.sessionId` property, but not both. In the `build.xml` file, your targets can contain all three parameters (`username`, `password`, and `sessionId`). Either the username and password or the session ID will be used for authentication.

Constructing a Project Manifest

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The `package.xml` file is a project manifest that lists all the components you want to retrieve or deploy in a single request. You can retrieve or deploy only a single package at a time.

The following elements may be defined in `package.xml`:

Name	Description
`<fullName>`	The name of the server-side package to deploy into. If the `<fullName>` field is omitted, components will not be assigned to a package when deployed, and will be in the `unpackaged` package. This field is not used for retrieve.
`<types>`	This element contains one or more `<members>` tags and one `<name>` tag, and is used to list the metadata components of a certain type to retrieve or deploy.
`<members>`	The full name of a component. There is one `<members>` element defined for each component in the directory. You can replace the value in this member with the wildcard character `*` (asterisk) instead of listing each member separately. This is a child element of `<types>`.
`<name>`	Contains the type of the component, for example `CustomObject` or `Profile`. There is one name defined for each component type in the directory. This is a child element of `<types>`.
`<version>`	The Metadata API version number of the files being retrieved or deployed. When deploying, all the files must conform to the same version of the Metadata API.

Component Types

For a complete list of the component types that can be defined by the `<name>` element in `package.xml`, see Metadata Types in the *Metadata API Developer Guide*.

Specifying Standard Objects

To retrieve standard objects and/or custom fields on standard objects, you must name the component in `package.xml`. The following `package.xml` file will retrieve a single field `EngineeringReqNumber__c`, on the Case object, as well as the entire Account object.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>Case.EngineeringReqNumber__c</members>
        <name>CustomField</name>
    </types>
    <types>
        <members>Account</members>
        <name>CustomObject</name>
    </types>
    <version>63.0</version>
</Package>
```

Note: Custom objects and standard objects should be specified in the same `<types>` section, the one containing `<name>CustomObject</name>`.

Specifying Named Components

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

To retrieve a component, specify the type of component in the `<name>` element and declare each component to be retrieved or deployed in the `<members>` element. The following is a sample `package.xml` project manifest that names two custom objects to be retrieved or deployed:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>MyCustomObject__c</members>
        <members>MyHelloWorldObject__c</members>
        <name>CustomObject</name>
    </types>
    <version>63.0</version>
</Package>
```

Some metadata components are sub-components of another component. This means you must dot-qualify the sub-component with the parent component name.

The following metadata components are defined as part of an object:

- `CustomField`
- `Picklist`

- `RecordType`
- `Weblink`
- `ValidationRule`

For example, the following code retrieves a validation rule called `ValidationRuleName` on the Opportunity object:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>Opportunity.ValidationRuleName</members>
        <name>ValidationRule</name>
    </types>
    <version>63.0</version>
</Package>
```

Specifying all Components of a Type

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

To retrieve all components of a particular type, use the wildcard symbol (*). For example, to retrieve all custom objects:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
    <version>63.0</version>
</Package>
```

The wildcard symbol does not apply to all metadata types. For example, using the wildcard with the `CustomObject` type name will not retrieve standard objects. To retrieve a standard object, you must explicitly name the object in `package.xml`. Likewise, if you want to retrieve custom fields defined on standard objects, you must name the object and field.

Specifying Standard Objects

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

To retrieve standard objects and/or custom fields on standard objects, you must name the component in `package.xml`. The following `package.xml` file will retrieve a single field `EngineeringReqNumber__c`, on the Case object, as well as the entire Account object.

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

```
        <members>Account</members>
        <name>CustomObject</name>
    </types>
    <version>63.0</version>
</Package>
```

Note: Custom objects and standard objects should be specified in the same `<types>` section, the one containing `<name>CustomObject</name>`.

Getting Information About Metadata Types

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

You sometimes need to experiment with the composition of your `package.xml` manifest file before you settle on the final version that retrieves or deploys the metadata that you want. There are a couple of helper targets, `<sf:describeMetadata>` and `<sf:listMetadata>`, that are useful for gathering the relevant information during this experimentation period. The `build.xml` file specifies a series of commands to be executed by Ant. Within the `build.xml` file are named targets that process a series of commands when you run Ant with a target name.

Describing Metadata Types

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The `describeMetadata` target returns a list of metadata types that are enabled for your organization. This target is useful when you want to identify the syntax needed for a metadata type in a `<name>` element in `package.xml`; for example, `CustomObject` for custom objects or `Layout` for page layouts. The following parameters may be set for each `<sf:describeMetadata>` target:

Field	Description
`username`	Required if `sessionId` isn't specified. The Salesforce username used for login. The username associated with this connection must have the Modify Metadata through Metadata API Functions permission.
`password`	Required if `sessionId` isn't specified. The password you use to log in to the organization associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`sessionId`	Required if `username` and `password` aren't specified. The ID of an active Salesforce session or the OAuth access token. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging in to an existing session instead of creating a new session. Alternatively, use an access token for OAuth authentication. For more information, see Authenticating Apps with OAuth in the Salesforce Help.
`serverurl`	Optional. The Salesforce server URL (if blank, defaults to `login.salesforce.com`). To connect to a sandbox instance, change this value to `test.salesforce.com`.
`apiVersion`	Optional. The API version to use for the metadata. The default is 63.0.

Field	Description
`resultFilePath`	Optional. The path of the output file where results are stored. The default output is the console. Directing the output to a file makes it easier to extract the relevant information for your `package.xml` manifest file.
`trace`	Optional. Defaults to `false`. Prints the SOAP requests and responses to the console. This option shows the user's password in plain text during login.

To get the list of metadata types enabled for your organization, specify a target in the `build.xml` file using `<sf:describeMetadata>`.

```
<target name="describeMetadata">
  <sf:describeMetadata
    username="${sf.username}"
    password="${sf.password}"
    sessionId="${sf.sessionId}"
    serverurl="${sf.serverurl}"
    resultFilePath="describeMetadata/describe.log"/>
</target>
```

Listing Components for a Metadata Type

The `listMetadata` target retrieves property information about your metadata components.This target is useful to identify individual components in `package.xml` for a retrieval or a high-level view of particular metadata types. For example, you can use this target to return a list of names of all your `CustomObject` or `Layout` components. Then use the information retrieve a subset of the components.

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

You can set parameters for each `<sf:listMetadata>` target.

Field	Description
`username`	Required if `sessionId` isn't specified. The Salesforce username used for login. The username associated with this connection must have the Modify Metadata through Metadata API Functions permission.
`password`	Required if `sessionId` isn't specified. The password you use to log in to the organization associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`sessionId`	Required if `username` and `password` aren't specified. The ID of an active Salesforce session or the OAuth access token. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging in to an existing session instead of creating a new session. Alternatively, use an access token for OAuth authentication. For more information, see Authenticating Apps with OAuth in the Salesforce Help.
`serverurl`	Optional. The Salesforce server URL (if blank, defaults to `login.salesforce.com`). To connect to a sandbox instance, change this value to `test.salesforce.com`.

Field	Description
`metadataType`	Required. The name of the metadata type that you're retrieving property information for, such as `CustomObject` for custom objects or `Report` for custom reports. The StandardValueSet metadata type isn't supported. To review the supported types, see the Metadata Types chapter in the *Metadata API Developer Guide*.
`folder`	The folder associated with the component. This field is required for components that use folders, such as Dashboard, Document, EmailTemplate, or Report.
`apiVersion`	Optional. The API version to use for the metadata. The default is 63.0.
`resultFilePath`	Optional. The path of the output file where results are stored. The default output is the console. Directing the output to a file makes it easier to extract the relevant information for your `package.xml` manifest file.
`trace`	Optional. Defaults to `false`. Prints the SOAP requests and responses to the console. This option shows the user's password in plain text during login.

To get property information for components of one metadata type, such as `CustomObject`, specify a target in the `build.xml` file that uses `<sf:listMetadata>`.

```
<target name="listMetadata">
  <sf:listMetadata
    username="${sf.username}"
    password="${sf.password}"
    sessionId="${sf.sessionId}"
    serverurl="${sf.serverurl}"
    metadataType="CustomObject"
    resultFilePath="listMetadata/list.log"/>
  </target>
```

This example uses a component that resides in a folder.

```
<target name="listMetadata">
  <sf:listMetadata
    username="${sf.username}"
    password="${sf.password}"
    sessionId="${sf.sessionId}"
    serverurl="${sf.serverurl}"
    metadataType="Report"
    folder="Marketing_Reports"
    resultFilePath="listMetadata/list.log"/>
</target>
```

Creating Retrieve Targets

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The `build.xml` file specifies a series of commands to be executed by Ant. Within the `build.xml` file are named targets that process a series of commands when you run Ant with a target name. The following parameters can be set for each `<sf:retrieve>` target:

Field	Description
`username`	Required if `sessionId` isn't specified. The Salesforce username used for login. The username associated with this connection must have the Modify Metadata through Metadata API Functions permission.
`password`	Required if `sessionId` isn't specified. The password you use to log in to the organization associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`sessionId`	Required if `username` and `password` aren't specified. The ID of an active Salesforce session or the OAuth access token. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging in to an existing session instead of creating a new session. Alternatively, use an access token for OAuth authentication. For more information, see Authenticating Apps with OAuth in the Salesforce Help.
`serverurl`	Optional. The Salesforce server URL (if blank, defaults to `login.salesforce.com`). To connect to a sandbox instance, change this value to `test.salesforce.com`.
`retrieveTarget`	Required. The root of the directory structure into which the metadata files are retrieved.
`packageNames`	Required if `unpackaged` is not specified. A comma-separated list of the names of the packages to retrieve. Specify either `packageNames` or `unpackaged`, but not both.
`apiVersion`	Optional. The Metadata API version to use for the retrieved metadata files. The default is 63.0.
`pollWaitMillis`	Optional. Defaults to `10000`. The number of milliseconds to wait between attempts when polling for results of the retrieve request. The client continues to poll the server up to the limit defined by `maxPoll`.
`maxPoll`	Optional. Defaults to `200`. The number of times to poll the server for the results of the retrieve request. The wait time between successive poll attempts is defined by `pollWaitMillis`.
`singlePackage`	Optional. Defaults to `true`. Set this parameter to `false` if you are retrieving multiple packages. If set to `false`, the retrieved zip file includes an extra top-level directory containing a subdirectory for each package.
`trace`	Optional. Defaults to `false`. Prints the SOAP requests and responses to the console. This option shows the user's password in plain text during login.
`unpackaged`	Required if `packageNames` is not specified. The path and name of a file manifest that specifies the components to retrieve. Specify either `unpackaged` or `packageNames`, but not both.
`unzip`	Optional. Defaults to `true`. If set to `true`, the retrieved components are unzipped. If set to `false`, the retrieved components are saved as a zip file in the `retrieveTarget` directory.

Retrieving Unpackaged Components

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The *unpackaged* package contains all of the standard objects, custom objects, Apex classes and other metadata components that exist natively in your organization, and not within a package. To retrieve unpackaged components, use a `build.xml` target that contains the unpackaged attribute that points to a `package.xml` file. For example:

```
<target name="retrieveUnpackaged">
  <mkdir dir="projectFolder"/>
  <sf:retrieve
      username="${sf.username}"
      password="${sf.password}"
      sessionId="${sf.sessionId}"
      serverurl="${sf.serverurl}"
      retrieveTarget="projectFolder"
      unpackaged="unpackaged/package.xml"/>
</target>
```

The `salesforce-ant.jar` file contains Ant *tasks* for accessing the Metadata API. In the code above, `sf:retrieve` is an Ant task. The full list of metadata Ant tasks are described in the *Metadata API Developer Guide*.

Retrieving Managed or Unmanaged Packages

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

Packages are useful for distributing related bundles of metadata across multiple instances or organizations, via Lightning Platform AppExchange. However, you can use the Ant Migration Tool to freely retrieve and deploy packaged metadata without using AppExchange. You retrieve both managed and unmanaged packages in the same way.

To retrieve a package, specify a `packageNames` parameter in the `build.xml` file. For example:

```
<target name="retrieveNamedPackage">
  <sf:retrieve
      username="${sf.username}"
      password="${sf.password}"
      sessionId="${sf.sessionId}"
      serverurl="${sf.serverurl}"
      retrieveTarget="projectFolder"
      packageNames="mySourcePackage"/>
</target>
```

Retrieving Components in Bulk

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

This target is the optimal way to download a large number of components of a single metadata type, such as custom reports, into a set of local files. The following parameters may be set for each `<sf:bulkRetrieve>` target:

Field	Description
`username`	Required if `sessionId` isn't specified. The Salesforce username used for login. The username associated with this connection must have the Modify Metadata through Metadata API Functions permission.
`password`	Required if `sessionId` isn't specified. The password you use to log in to the organization associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`sessionId`	Required if `username` and `password` aren't specified. The ID of an active Salesforce session or the OAuth access token. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging in to an existing session instead of creating a new session. Alternatively, use an access token for OAuth authentication. For more information, see Authenticating Apps with OAuth in the Salesforce Help.
`serverurl`	Optional. The Salesforce server URL (if blank, defaults to `login.salesforce.com`). To connect to a sandbox instance, change this value to `test.salesforce.com`.
`retrieveTarget`	Required. The root of the directory structure into which the metadata files are retrieved.
`metadataType`	Required. The name of the metadata type to be retrieved; for example, `CustomObject` for custom objects, or `Report` for custom reports. For a full list of allowed values, see Component Types on page 8.
`containingFolder`	Optional. If the metadata is contained in a folder, this parameter should be the name of the folder from which the contents are retrieved.
`batchSize`	Optional, defaults to 10. The number of items to retrieve while doing multi-part retrieve.
`apiVersion`	Optional. The Metadata API version to use for the retrieved metadata files. The default is 63.0.
`maxPoll`	Optional. Defaults to `200`. The number of times to poll the server for the results of the retrieve request. The clients waits for two seconds after the first poll attempt. The wait time is doubled for each successive poll attempt up to maximum of 30 seconds between poll attempts.
`unzip`	Optional. Defaults to `true`. If set to `true`, the retrieved components are unzipped. If set to `false`, the retrieved components are saved as a zip file in the `retrieveTarget` directory.

To retrieve custom report components in bulk, specify a target in the `build.xml` file using `<sf:bulkRetrieve>`.

```
<target name="bulkRetrieve">
  <sf:bulkRetrieve
    username="${sf.username}"
    password="${sf.password}"
    sessionId="${sf.sessionId}"
    serverurl="${sf.serverurl}"
    metadataType="Report"
    retrieveTarget="retrieveUnpackaged"/>
</target>
```

Retrieving Metadata from a Salesforce Organization

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

To retrieve Lightning Platform components:

1. Open a command prompt.
2. Run Ant by specifying a target name in `build.xml`. If it's the first time you're running Ant, use `ant retrieveUnpackaged` to retrieve unpackaged components specified in `package.xml`.

Note:

- The sample `build.xml` contains some useful targets for various `retrieve()` and `deploy()` options that you can modify or use as is. To see a list of all your named targets in `build.xml`, enter `ant -p` at the command line.
- You can deploy or retrieve up to 10,000 files at once. AppExchange packages use different limits: They can contain up to 35,000 files. The maximum size of the deployed or retrieved .zip file is 39 MB. If the files are uncompressed in an unzipped folder, the size limit is 600 MB. If you're working with many components, use the `listMetadata` target to identify the subset of files that you want to retrieve. You can also retrieve batches of components as described in Retrieving Components in Bulk.

Editing Metadata

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

You can use any UTF-8 text editor to make changes to the files you retrieve.

Warning: Text editors that do not natively support UTF-8 may insert a byte order mark (BOM) at the top of the file, which can cause problems in the XML metadata.

Deleting Files from an Organization

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The `package.xml` file is a project manifest that lists all the components to retrieve or deploy. Although you can use `package.xml` to add components, it's not sufficient to delete them. To delete files, create a delete manifest that's called `destructiveChanges.xml`. The format of the delete manifest is the same as `package.xml`, except that wildcards aren't supported.

Deleting Components in a Deployment

To delete components, use the same procedure as with deploying components, but also include a delete manifest file that's named `destructiveChanges.xml` and list the components to delete in this manifest. The format of this manifest is the same as `package.xml` except that wildcards aren't supported.

Note: You can't use `destructiveChanges.xml` to delete items that are associated with an active Lightning page, such as a custom object, a component on the page, or the page itself. First, you must remove the page's action override by deactivating it in the Lightning App Builder.

The following sample `destructiveChanges.xml` file names a single custom object to be deleted:

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

To deploy the destructive changes, you must also have a `package.xml` file that lists no components to deploy, includes the API version, and is in the same directory as `destructiveChanges.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <version>63.0</version>
</Package>
```

Note:

- To bypass the Recycle Bin, set the `purgeOnDelete` option to `true`.
- When you delete a roll-up summary field using Metadata API, the field isn't saved in the Recycle Bin. The field is purged even if you don't set the `purgeOnDelete` deployment option to `true`.
- If you try to delete some components that don't exist in the organization, the rest of the deletions are still attempted.

Adding and Deleting Components in a Single Deployment

You can perform a deployment that specifies components to delete in `destructiveChanges.xml` and components to add or update in `package.xml`. The process is the same as with performing a delete-only deployment except that `package.xml` contains the components to add or update.

By default, deletions are processed before component additions. In API version 33.0 and later, you can specify components to be deleted before and after component additions. The process is the same as with performing a delete-only deployment except that the name of the deletion manifest file is different.

- To delete components *before* adding or updating other components, create a manifest file that's named `destructiveChangesPre.xml` and include the components to delete.
- To delete components *after* adding or updating other components, create a manifest file that's named `destructiveChangesPost.xml` and include the components to delete.

The ability to specify when deletions are processed is useful when you're deleting components with dependencies. For example, if a custom object is referenced in an Apex class, you can't delete it unless you modify the Apex class first to remove the dependency on the custom object. In this example, you can perform a single deployment that updates the Apex class to clear the dependency and then deletes the custom object by using `destructiveChangesPost.xml`. The following are samples of the `package.xml` and `destructiveChangesPost.xml` manifests that would be used in this example.

Sample `package.xml`, which specifies the class to update:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>SampleClass</members>
        <name>ApexClass</name>
    </types>
    <version>63.0</version>
</Package>
```

Sample `destructiveChangesPost.xml`, which specifies the custom object to delete after the class update:

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

Note: The API version that the deployment uses is the API version that's specified in `package.xml`.

Deploying Changes to a Salesforce Org

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The `build.xml` file specifies targets to retrieve and deploy. You can set the following parameters for each deploy target.

Field	Description
`username`	Required if `sessionId` isn't specified. The Salesforce username for login. The username associated with this connection must have the Modify Metadata through Metadata API Functions permission.
`password`	Required if `sessionId` isn't specified. The password you use to log in to the org associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`sessionId`	Required if `username` and `password` aren't specified. The ID of an active Salesforce session or the OAuth access token. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging in to an existing session instead of creating a new session. Alternatively, use an access token for OAuth authentication. For more information, see Authenticating Apps with OAuth in the Salesforce Help.
`serverurl`	Optional. The Salesforce server URL (if blank, defaults to `login.salesforce.com`). To connect to a sandbox instance, change this URL to `test.salesforce.com`.

Field	Description
`pollWaitMillis`	Optional. Defaults to `10000`. The number of milliseconds to wait when polling for results of the deployment. Deployment can succeed even if you stop waiting.
`checkOnly`	Optional. Defaults to `false`. Set to `true` to perform a test deployment (validation) of components without saving the components in the target org. A validation enables you to verify the results of tests that would be generated in a deployment, but doesn't commit any changes. After the validation finishes with passing tests, it might qualify for deployment without rerunning tests. See Deploying a Recent Validation. Note: If you change a field type from Master-Detail to Lookup or vice versa, the change isn't supported when using the `checkOnly` parameter to test a deployment. This change isn't supported for test deployments to avoid data loss or corruption. If a change that isn't supported for test deployments is included in the deployment package, the test deployment fails and issues an error. If your deployment package changes a field type from Master-Detail to Lookup or vice versa, you can still validate the changes before you deploy to production. Perform a full deployment to another test sandbox. A full deployment includes a validation of the changes as part of the deployment process.
`maxPoll`	Optional. Defaults to `200`. The number of times to poll the server for the results of the deploy request. Deployment can succeed even if you stop waiting.
`deployRoot`	Required if `zipFile` isn't specified. Specifies the root of the directory tree of files to deploy. You must define a value for either `zipFile` or `deployRoot`.
`zipFile`	Required if `deployRoot` isn't specified. Specifies the path of the metadata zip file to be deployed. You must define a value for either `zipFile` or `deployRoot`.
`singlePackage`	Optional. Defaults to `false`. Declares that the `zipFile` or `deployRoot` parameter points to a directory structure with a single package, as opposed to a set of packages.
`allowMissingFiles`	Optional. Defaults to `false`. Specifies whether a deploy succeeds even if files that are specified in `package.xml` are not in the zip file. Do not use this parameter for deployment to production orgs.
`autoUpdatePackage`	Optional. Defaults to `false`. Specifies whether a deploy continues even if files present in the zip file are not specified in `package.xml`. Do not use this parameter for deployment to production orgs.
`ignoreWarnings`	Optional. Defaults to `false`. This setting indicates that a deployment succeeds even if there are warnings (`true`) or that one or more warnings causes the deployment to fail and roll back (`false`). If there are errors, as opposed to warnings, the deployment always fails and rolls back.
`logType`	Optional. The debug logging level for running tests. The default is `None`. Valid options are: • `None` • `Debugonly` • `Db` • `Profiling` • `Callout`

Field	Description
	• `Detail`
`purgeOnDelete`	If `true`, the deleted components in the `destructiveChanges.xml` manifest file aren't stored in the Recycle Bin. Instead, they become immediately eligible for deletion.This option only works in Developer Edition or sandbox orgs. It doesn't work in production orgs.
`rollbackOnError`	Optional. Defaults to `true`. Indicates whether any failure causes a complete rollback (`true`) or not (`false`). If `false`, whatever set of actions can be performed without errors are performed, and errors are returned for the remaining actions. This parameter must be set to `true` if you are deploying to a production org. Note: In earlier versions of the Ant Migration Tool (Spring '14 and earlier), this parameter is ignored in `build.xml`, and Salesforce behaves as if this parameter is set to `true`.
`runAllTests`	(Deprecated and available only in API version 33.0 and earlier.) This parameter is optional and defaults to `false`. Set to `true` to run all Apex tests after deployment, including tests that originate from installed managed packages. Note: Apex tests that run as part of a deployment always run synchronously and serially.
`runTest`	Optional child elements. A list of Apex classes containing tests run after deployment. For more information, see Running a Subset of Tests in a Deployment. To use this option, set `testLevel` to `RunSpecifiedTests`.
`testLevel`	Optional. Specifies which tests are run as part of a deployment. The test level is enforced regardless of the types of components that are present in the deployment package. Valid values are: • `NoTestRun`—No tests are run. This test level applies only to deployments to development environments, such as sandbox, Developer Edition, or trial organizations. This test level is the default for development environments. • `RunSpecifiedTests`—Only the tests that you specify in the `runTests` option are run. Code coverage requirements differ from the default coverage requirements when using this test level. Each class and trigger in the deployment package must be covered by the executed tests for a minimum of 75% code coverage. This coverage is computed for each class and triggers individually and is different than the overall coverage percentage. • `RunLocalTests`—All tests in your org are run, except the ones that originate from installed managed and unlocked packages. This test level is the default for production deployments that include Apex classes or triggers. • `RunAllTestsInOrg`—All tests are run. The tests include all tests in your org, including tests of managed packages. If you don't specify a test level, the default test execution behavior is used. See Running Tests in a Deployment. Note: Apex tests that run as part of a deployment always run synchronously and serially. This field is available in API version 34.0 and later.

Field	Description
`trace`	Optional. Defaults to `false`. Prints the SOAP requests and responses to the console. This option shows the user's password in plain text during login.

Note: The Ant Migration Tool ignores files or folders with a name starting with a period (.) or ending with a tilde (~) when deploying files. Some source control systems, such as Subversion, create files or folders with names starting with a period. These files can cause issues during deployment to Salesforce, so the Ant Migration Tool ignores them.

The Ant Migration Tool comes with a sample `build.xml` file that lists several deployment targets. You want to create your own custom targets using the sample targets as starting points.

- `deployUnpackaged` — Deploys unpackaged components specified in the target.
- `deployCode` — Deploys the contents of the `codepkg` package specified in the target.
- `undeployCode` — Deletes classes and triggers in the `removecodepkg` directory specified by the `destructiveChanges.xml` manifest. This file is similar to `package.xml`, but lists components to be deleted. For more information, see Deleting Files from an Organization on page 16.
- `deployCodeFailingTest` — Deploys code that fails testing requirements, strictly for demonstration purposes.
- `deployCodeCheckOnly` — Verifies that the deployment works, but doesn't deploy any components.

Note: You can deploy or retrieve up to 10,000 files at once. AppExchange packages use different limits. In API version 43.0 and 44.0, AppExchange packages can contain up to 12,500 files. In API version 45.0, AppExchange packages can contain up to 17,500 files. In API version 46.0, AppExchange packages can contain up to 22,000 files. In API version 47.0 through 50.0, AppExchange packages can contain up to 30,000 files. In API version 51.0 and later, AppExchange packages can contain up to 31,000 files.The maximum size of the deployed or retrieved .zip file is 39 MB. If the files are uncompressed in an unzipped folder, the size limit is 400 MB.

- If using the Ant Migration Tool to deploy an unzipped folder, all files in the folder are compressed first. The maximum size of uncompressed components in an unzipped folder is 400 MB or less depending on the compression ratio. If the files have a high compression ratio, you can migrate a total of approximately 400 MB because the compressed size would be under 39 MB. However, if the components can't be compressed much, like binary static resources, you can migrate less than 400 MB.
- Metadata API base-64 encodes components after they're compressed. The resulting .zip file can't exceed 50 MB, which is the limit for SOAP messages. Base-64 encoding increases the size of the payload, so your compressed payload can't exceed approximately 39 MB before encoding.
- You can perform a `retrieve()` call for a big object only if its index is defined. If a big object is created in Setup and doesn't yet have an index defined, you can't retrieve it.

Deploying Components

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

You can deploy any set of components as a package or into your organization directly in the unpackaged package. The package used is not determined by the `build.xml` target, but by the project manifest (`package.xml`). A sample deployment target follows:

```
<target name="deployUnpackaged">
  <sf:deploy
      username="${sf.username}"
      password="${sf.password}"
      sessionId="${sf.sessionId}"
      serverurl="${sf.serverurl}"
      deployroot="projectFolder"/>
</target>
```

Deploying Code

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

You can deploy metadata components and Apex at the same time, but you may find it useful to create separate targets for deploying Apex, so that you can run tests as part of the deployment. A portion of a `build.xml` file is listed below, with a target named `deployCode` that deploys the contents of the `codepkg` package and runs the tests for one class.

```
<target name="deployCode">
  <sf:deploy
      username="${sf.username}"
      password="${sf.password}"
      sessionId="${sf.sessionId}"
      serverurl="${sf.serverurl}"
      deployroot="codepkg">
    <runTest>SampleDeployClass</runTest>
  </sf:deploy>
</target>
```

SEE ALSO:

Running a Subset of Tests in a Deployment

Deploying a Recent Validation

Deploying a validation helps you shorten your deployment time because tests aren't rerun. If you have a recent successful validation, you can deploy the validated components without running tests. You can deploy a recent validation with the `<sf:deployRecentValidation>` task.

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

A validation doesn't save any components in the organization. You use a validation only to check the success or failure messages that you would receive with an actual deployment. To validate your components, add the `checkOnly="true"` parameter in your deploy target (`<sf:deploy>`).

Before deploying a recent validation, ensure that the following requirements are met.

- The components have been validated successfully for the target environment within the last 10 days.
- As part of the validation, Apex tests in the target org have passed.
- Code coverage requirements are met.
  - If all tests in the org or all local tests are run, overall code coverage is at least 75%, and Apex triggers have some coverage.

- If specific tests are run with the `RunSpecifiedTests` test level, each class and trigger that was deployed is covered by at least 75% individually.

The `<sf:deployRecentValidation>` task supports these parameters.

Field	Description
`username`	Required if `sessionId` isn't specified. The Salesforce username for login. The username associated with this connection must have the Modify Metadata through Metadata API Functions permission.
`password`	Required if `sessionId` isn't specified. The password you use to log in to the org associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`recentValidationId`	Required. Specifies the ID of a recent validation.
`sessionId`	Required if `username` and `password` aren't specified. The ID of an active Salesforce session or the OAuth access token. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging in to an existing session instead of creating a new session. Alternatively, use an access token for OAuth authentication. For more information, see Authenticating Apps with OAuth in the Salesforce Help.
`serverurl`	Optional. The Salesforce server URL (if blank, defaults to `login.salesforce.com`). To connect to a sandbox instance, change this URL to `test.salesforce.com`.
`rollbackOnError`	Optional. Defaults to `true`. Indicates whether any failure causes a complete rollback (`true`) or not (`false`). If `false`, whatever set of actions can be performed without errors are performed, and errors are returned for the remaining actions. This parameter must be set to `true` if you are deploying to a production org.
`maxPoll`	Optional. Defaults to `200`. The number of times to poll the server for the results of the deploy request. Deployment can succeed even if you stop waiting.
`pollWaitMillis`	Optional. Defaults to `10000`. The number of milliseconds to wait when polling for results of the deployment. Deployment can succeed even if you stop waiting.
`trace`	Optional. Defaults to `false`. Prints the SOAP requests and responses to the console. This option shows the user's password in plain text during login.

This example shows a target for deploying a recent validation. The value of `recentValidationId` (`sf.recentValidationId`) is defined in the `build.properties` file.

```
<target name="quickDeploy">
  <sf:deployRecentValidation username="${sf.username}" password="${sf.password}"
                             sessionId="${sf.sessionId}" serverurl="${sf.serverurl}"
                             maxPoll="${sf.maxPoll}"
                             recentValidationId="${sf.recentValidationId}"/>
</target>
```

Support for `<sf:deployRecentValidation>` starts with version 34.0 of the Ant Migration Tool.

Running Tests in a Deployment

Default Test Execution in Production

When no test level is specified in the deployment options, the default test execution behavior depends on the contents of your deployment package. When deploying to production, all tests, except those that originate from managed packages, are executed if your deployment package contains Apex classes or triggers. If your package doesn't contain Apex components, no tests are run by default.

In API version 33.0 and earlier, tests were run for components that required tests, such as custom objects, and not only for Apex components. For example, if your package contains a custom object, all tests are run in API version 33.0 and earlier. In contrast, starting with API version 34.0, no tests are run for this package. The API version corresponds to the version of your API client or the version of the tool you're using (Ant Migration Tool).

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 sandbox, before deploying to production. Running tests in your development environment reduces the number of tests needed to run in a production deployment.

Default Test Execution in Production for API Version 33.0 and Earlier

For deployment to a production organization, all local tests in your organization are run by default. Tests that originate from installed managed packages aren't run by default. If any test fails, the entire deployment is rolled back.

If the deployment includes components for the following metadata types, all local tests are run.

- ApexClass
- ApexComponent
- ApexPage
- ApexTrigger
- ArticleType
- BaseSharingRule
- CriteriaBasedSharingRule
- CustomField
- CustomObject
- DataCategoryGroup
- Flow
- InstalledPackage
- NamedFilter
- OwnerSharingRule
- PermissionSet
- Profile
- Queue
- RecordType
- RemoteSiteSetting
- Role
- SharingReason

- Territory
- Validation Rules
- Workflow

For example, no tests are run for the following deployments:

- 1 CustomApplication component
- 100 Report components and 40 Dashboard components

But all local tests are run for any of the following example deployments, because they include at least one component from the list above:

- 1 CustomField component
- 1 ApexComponent component and 1 ApexClass component
- 5 CustomField components and 1 ApexPage component
- 100 Report components, 40 Dashboard components, and 1 CustomField component

Running a Subset of Tests in a Deployment

Test levels enable you to have more control over which tests are run in a deployment. To shorten deployment time to production, run a subset of tests when deploying Apex components. The default test execution behavior in production has also changed. By default, if no test level is specified, no tests are executed, unless your deployment package contains Apex classes or triggers.

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

If the code coverage of an Apex component in the deployment is less than 75%, the deployment fails. If one of the specified tests fails, the deployment also fails. We recommend that you test your deployment in sandbox first to ensure that the specified tests cover each component sufficiently. Even if your organization's overall code coverage is 75% or more, the individual coverage of the Apex components being deployed can be less. If the code coverage requirement isn't met, write more tests and include them in the deployment.

To run a subset of tests, add the `testLevel="RunSpecifiedTests"` parameter to the deploy target. Specify each test class to run for a deploy target in a `<runTest> </runTest>` child element within the `sf:deploy` element. Add the test class name within the `<runTest> </runTest>` tags. Add as many `runTest` tags as you need, one for each test class.

This deploy target example shows three test classes. Salesforce runs these test classes when deploying this package.

```
<target name="deployCode">
    <sf:deploy username="${sf.username}" password="${sf.password}"
           sessionId="${sf.sessionId}" serverurl="${sf.serverurl}"
           deployroot="codepkg" testLevel="RunSpecifiedTests">
        <runTest>TestClass1</runTest>
        <runTest>TestClass2</runTest>
        <runTest>TestClass3</runTest>
    </sf:deploy>
</target>
```

The test class name can include a namespace prefix. Add a namespace prefix if your organization has a namespace defined or if the test class belongs to a managed package. For example, if the namespace is `MyNamespace`, specify the test class as `MyNamespace.TestClass1`.

If you don't specify a test class to run in the target, the default deployment behavior applies when deploying to production. The default is all tests in your organization run on deployment except the tests that originate from installed managed packages. The default code coverage requirements are also enforced. The requirements are a minimum overall percentage of 75% for all classes and triggers, and no trigger can have 0% coverage.

Notes About Running Specific Tests

- You can specify only test classes. You can't specify individual test methods.
- We recommend that you refactor test classes to include the minimum number of tests that meet code coverage requirements. Refactoring your test classes can contribute to shorter test execution times, and as a result, shorter deployment times.
- You can deactivate a trigger in the target organization by deploying it with an inactive state. However, the trigger must have been previously deployed with an active state.

Run the Same Tests in Sandbox and Production Deployments

Starting in API version 34.0, you can choose which tests to run in your development environment, such as only local tests, to match the tests run in production. In earlier versions, if you enabled tests in your sandbox deployment, you couldn't exclude managed package tests.

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

By default, no tests are run in a deployment to a non-production organization, such as a sandbox or a Developer Edition organization. To specify tests to run in your development environment, set a `testLevel` deployment option. For example, to run local tests in a deployment and to exclude managed package tests, add the `testLevel="RunLocalTests"` parameter to the deploy target as shown in this example.

```
<target name="deployCode">
    <sf:deploy username="${sf.username}" password="${sf.password}"
           sessionId="${sf.sessionId}" serverurl="${sf.serverurl}"
           deployroot="codepkg" testLevel="RunLocalTests">
    </sf:deploy>
</target>
```

Note: The `RunLocalTests` test level is enforced regardless of the contents of the deployment package. In contrast, tests are executed by default in production only if your deployment package contains Apex classes or triggers. You can use `RunLocalTests` for sandbox and production deployments.

Canceling a Deployment

You can cancel a deployment that's in progress or queued with the `<sf:cancelDeploy>` task.

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

The `<sf:cancelDeploy>` task supports these parameters.

Field	Description
`username`	Required if `sessionId` isn't specified. The Salesforce username for login. The username associated with this connection must have the Modify Metadata through Metadata API Functions permission.
`password`	Required if `sessionId` isn't specified. The password you use to log in to the org associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`sessionId`	Required if `username` and `password` aren't specified. The ID of an active Salesforce session or the OAuth access token. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging in to an existing session instead of creating a new session. Alternatively, use an access token for OAuth authentication. For more information, see Authenticating Apps with OAuth in the Salesforce Help.
`requestId`	Required. Specifies the ID of an in-progress or queued deployment to cancel.
`serverurl`	Optional. The Salesforce server URL (if blank, defaults to `login.salesforce.com`). To connect to a sandbox instance, change this URL to `test.salesforce.com`.
`maxPoll`	Optional. Defaults to `200`. The number of times to poll the server for the results of the deploy request. Deployment can succeed even if you stop waiting.
`pollWaitMillis`	Optional. Defaults to `10000`. The number of milliseconds to wait when polling for results of the deployment. Deployment can succeed even if you stop waiting.
`trace`	Optional. Defaults to `false`. Prints the SOAP requests and responses to the console. This option shows the user's password in plain text during login.

This example shows a target for canceling a deployment. The value of `requestId` (`sf.deployRequestId`) is defined in the `build.properties` file.

```
<target name="cancel">
  <sf:cancelDeploy username="${sf.username}" password="${sf.password}"
      sessionId="${sf.sessionId}" serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}
      requestId="${sf.deployRequestId}"/>
</target>
```

Support for `sf:cancelDeploy` starts with version 34.0 of the Ant Migration Tool.

Checking the Status of a Task

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

When you run a target, the Ant Migration Tool outputs the request ID for each deploy or retrieve task included in the target. You can use the request ID to check the status of a deploy or retrieve task. This is particularly useful if a client is running for a long time and there is a network issue that breaks the connectivity between the Ant Migration Tool on your machine and Salesforce.

To check the status of a run target, use the following command:

```
ant -Dsf.asyncRequestId=requestID targetName
```

Use the *`requestID`* that was printed out when you ran the target. For example, if you run the `deployZip` target, you can run the following command to check the status of the retrieval:

```
ant -Dsf.asyncRequestId=04sx00000002aGuAAI deployZip
```

Note: You should not use this command with targets that contain multiple retrieve or deploy tasks. You should not use this command with the `bulkRetrieve` task also as this task batches retrievals in multiple requests.

To track the status of deployments from within Salesforce, from Setup, enter *`Deployment`* in the `Quick Find` box, then select **Deployment Status**.

CHAPTER 4 Common Migration Issues

Note: The Ant Migration Tool is retired with Spring '24. The tool continues to function for future API versions but isn't updated with new functionality and isn't supported. To manage metadata changes, switch to Salesforce CLI for a modern, supported developer experience.

There are a number of common issues you may run into when migrating metadata and deploying changes. These issues can be grouped into three categories:

- Metadata — Special considerations for dealing with certain metadata components
- Connectivity — Problems connecting to an organization or polling for results
- Testing and Code Coverage — Testing Apex before deployment

Common Metadata Issues

The most common metadata issues are detailed below:

- Retrieving custom fields on standard objects — When you use the wildcard symbol in `package.xml`, to retrieve all objects, you will not retrieve standard objects, or custom fields on standard objects. To retrieve custom fields on standard objects, see Constructing a Project Manifest on page 7.
- Profiles or permission sets and field-level security — The contents of a retrieved profile or permission set depend on the other contents of the retrieve request. For example, field-level security for fields included in custom objects are returned at the same time as profiles or permission sets. For more information, see Profile and PermissionSet in the *Metadata API Developer Guide*.
- Understanding packages — Packages are used to bundle related components so they can be shared with multiple organizations, or distributed on Lightning Platform AppExchange. Managed packages are packages that can be upgraded in the installer's organization. They differ from unmanaged packages in that some components are locked, in order to permit upgrades. Metadata components that are not in any package can be accessed with the `unpackaged` attribute of `sf:retrieve` and `sf:deploy`.
- Workflow — A `.workflow` file is a container for the individual workflow components associated with an object, including WorkflowAlert, WorkflowFieldUpdate, WorkflowOutboundMessage, WorkflowRule, and WorkflowTask. To retrieve all workflows, include the following XML in `package.xml`:

  ```
  <types>
      <members>*</members>
      <name>Workflow</name>
  </types>
  ```

- Retrieving or deploying components that depend on an object definition — The following metadata components are dependent on a particular object for their definition: `CustomField`, `Picklist`, `RecordType`, `Weblink`, and `ValidationRule`. This means you must dot-qualify the component name with the object name in `package.xml`, and may not use the wildcard symbol. For more information, see Constructing a Project Manifest on page 7.
- Personal folders — Users' personal folders, for both reports and documents, are not exposed in the Metadata API. To migrate reports or documents you must move them to a public folder.

Connection Problems

The most common connection problems are detailed below:

- Request timed out — When you retrieve or deploy metadata, the request is sent asynchronously, meaning that the response is not returned immediately. Because these calls are asynchronous, the server will process a `deploy()` to completion even if the Ant Migration Tool times out. If the deploy succeeds, the changes will be committed to the server. If the deploy fails after timing out, there is no way to retrieve the error message from the server. For this reason, it is important to tune your `pollWaitMillis` and `maxPoll` parameters if you receive time-out errors:
  - `pollWaitMillis` — The number of milliseconds to wait between polls for retrieve/deploy results. The default value is `10000` milliseconds (ten seconds). For long-running processes, increase this number to reduce the total number of polling requests, which count against your daily API limits.
  - `maxPoll` — The number of polling attempts to be performed before aborting. The default value is `200`. When combined with the default value of `pollWaitMillis (10000)`, this means the Ant Migration Tool will give the server process a total of 2,000 seconds (33 minutes) to complete before timing out. The total time is computed as 200 poll attempts, waiting 10 seconds between each.

  Note: Since these parameters have default values, they do not have to be specified, and may not exist on your named targets. To add these parameters, include them in the target definition. For example:

  ```
  <sf:retrieve
    username="${sf.username}"
    password="${sf.password}"
    sessionId="${sf.sessionId}"
    serverurl="${sf.serverurl}"
    retrieveTarget="retrieveUnpackaged"
    unpackaged="unpackaged/package.xml"
    pollWaitMillis="10000"
    maxPoll="100"/>
  ```

- Invalid username, password, security token; or user locked out - This error indicates a problem with the credentials in `build.properties`:
  - Username — Verify that your username is correct on this account. When connecting to a sandbox instance your sandbox name is appended to your username. For example, if your production username is `foo@salesforce.com`, and one of your sandboxes is called `bar`, then your sandbox username is `foo@salesforce.com.bar`.
  - Password — Verify that your password is correct for this account. Note that you security token is appended to the end of your pasword.
  - Security token — The security token is a 25-digit string appended to your password. To reset your security token, go to the Reset My Security Token page in your personal settings.
  - Locked out — If you unsuccessfully attempt to log into an organization too many times, you may be temporarily locked out. The number of times you may fail to connect and the lockout duration depend on your organization settings. Either contact your administrator to have the lock removed, or wait until the lockout period expires.
  - Connection mismatch between sandbox and production — If all of your connection credentials in `build.properties` are correct, you may have a URL mismatch. The server URL is different for sandbox and production environments. In `build.properties`, the `sf.serverurl` value for production is `https://login.salesforce.com`. For sandbox environments, the value is `https://test.salesforce.com`.
- Proxy settings — If you connect through a proxy, you will need to follow the Ant Proxy Configuration instructions.

Testing in Apex

When you deploy to a production organization and don't specify the tests to run, every unit test in your organization namespace is executed. Before you deploy Apex, the following must be true:

- Unit tests must cover at least 75% of your Apex code, and all of those tests must complete successfully.

  Note the following.

  - When deploying Apex to a production organization, each unit test in your organization namespace is executed by default.
  - Calls to `System.debug` aren't counted as part of Apex code coverage.
  - Test methods and test classes aren't counted as part of Apex code coverage.
  - While only 75% of your Apex code must be covered by tests, don't focus on the percentage of code that is covered. Instead, make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This approach ensures that 75% or more of your code is covered by unit tests.
- Every trigger must have some test coverage.
- All classes and triggers must compile successfully.

If you specify the tests to run, the code coverage calculation for the deployment is slightly different See Running a Subset of Tests in a Deployment.

Salesforce recommends that you write tests for the following:

Single action
Test to verify that a single record produces the correct, expected result.

Bulk actions
Any Apex code, whether a trigger, a class or an extension, may be invoked for 1 to 200 records. You must test not only the single record case, but the bulk cases as well.

Positive behavior
Test to verify that the expected behavior occurs through every expected permutation, that is, that the user filled out everything correctly and did not go past the limits.

Negative behavior
There are likely limits to your applications, such as not being able to add a future date, not being able to specify a negative amount, and so on. You must test for the negative case and verify that the error messages are correctly produced as well as for the positive, within the limits cases.

Restricted user
Test whether a user with restricted access to the sObjects used in your code sees the expected behavior. That is, whether they can run the code or receive error messages.

Note: Conditional and ternary operators are not considered executed unless both the positive and negative branches are executed.

For more information, see "Understanding Testing in Apex" in the *Apex Developer Guide*.

INDEX