Custom Metadata Types Implementation Guide

Salesforce, Winter '16

@salesforcedocs

Last updated: October 1, 2015

CONTENTS

CUSTOM METADATA TYPES

You can create your own declarative developer frameworks for internal teams, partners, and customers. Rather than building apps from data, you can build apps that are defined and driven by their own types of metadata. Metadata is the information that describes the configuration of each customer's organization.

Custom metadata is customizable, deployable, packageable, and upgradeable application metadata. First, you create a *custom metadata type*, which defines the form of the application metadata. Then you build reusable functionality that determines the behavior based on metadata of that type. Similar to a custom object or custom setting, a custom metadata type has a list of custom fields that represent aspects of the metadata. After you create a public custom metadata type, you or others can declaratively create *custom metadata records* that are defined by that type. When you package a public custom metadata type, customers who install the package can add their own records to the metadata type. Your reusable functionality reads your custom metadata and uses it to produce customized application behavior.

EDITIONS

Available in: Salesforce Classic

Available in:
- Enterprise
- Performance
- Unlimited
- Developer
- Database.com

Custom metadata rows resemble custom object rows in structure. You create, edit, and delete custom metadata rows in the Metadata API or in Setup. Because the records are metadata, you can migrate them using using packages or Metadata API tools. Custom metadata records are read-only in Apex and in the Enterprise and Partner APIs.

With custom metadata types, you can issue unlimited Salesforce Object Query Language (SOQL) queries for each Apex transaction.

Custom metadata types support the following custom field types.

- Checkbox
- Date
- Date and Time
- Email
- Number
- Percent
- Phone
- Text
- Text Area
- URL

A subscriber to a managed package containing a custom metadata type can't add their own fields to that type. Only the org that develops the type can add custom fields to it.

Custom metadata fields are *manageable*, which means that the developer of a type can decide who can change field values after they are deployed to a subscriber organization.

- Locked after release—For any record of the type, the value of the field is immutable after deployment, even on the developer organization where the record was created.
- Subscriber editable—Anyone with the correct permissions can change the value of the field at will. Any changes the developer deploys do not overwrite values in the subscriber's organization.
- Upgradable—The developer of a record can change the value of the field by releasing a new version of the custom metadata package. The subscriber can't change the value of the field.

Custom metadata types and records have names and labels. Type names must be unique within their namespace. Record names must be unique within their custom metadata type and namespace.

Custom metadata records can be protected. If a developer releases protected records in a managed package, access to them is limited in specific ways.

- Code that's in the same managed package as custom metadata records can read the records.
- Code that's in the same managed package as custom metadata types can read the records that belong to that type.
- Code that's in a managed package that doesn't contain either the type or the protected record can't read the protected records.
- Code that the subscriber creates and code that's in an unmanaged package can't read the protected records.
- The developer can modify protected records only with a package upgrade. The subscriber can't read or modify protected records. The developer name of a protected record can't be changed after release.

If you create a protected custom metadata record in your organization, then it's accessible only by your code, code from unmanaged packages, and code from the managed package that defines its type.

Custom metadata types can also be protected, providing the same access protection as protected records. If you change a type from protected to public, its protected records remain protected and all other records become public. If you use Setup to create a new record on a protected type, the Protected Component checkbox is checked by default. Once a type is public, you can't convert it to protected. The subscriber can't create records of a protected type.

The custom metadata types documentation refers to a sample application and three fictional companies.

- Picklists R Us develops reusable enhancements to the Salesforce App Cloud that involve picklist-related functionality.
- TravelApp, Inc. develops an interplanetary travel application that uses picklist features from Picklists R Us.
- Galactic Tours is a customer of these organizations. Galactic Tours installs Picklists R Us's package and TravelApp, Inc.'s extension into its organization.

Visit the Custom Metadata Types community group at `success.salesforce.com` to get your own copy of the sample application and discuss this functionality.

CUSTOM METADATA TYPES LIMITATIONS

EDITIONS

Available in: Salesforce Classic

Available in:
- Enterprise
- Performance
- Unlimited
- Developer
- Database.com

When using custom metadata types, be aware of these special behaviors and limitations.

No upsert()

The `upsert()` function isn't available for custom metadata.

Updating Types and Records

You can't update protected types and records in an installed managed package programmatically. You can modify protected types and records only by performing a package upgrade.

You can't update public types and records by using Apex directly. To modify records from Apex, you must make calls to the Metadata API.

Metadata relationships

Metadata relationships aren't supported. However, you can simulate them using text fields that contain the API name of the target object.

Application lifecycle management tools

Custom metadata types don't support these application lifecycle management tools:

- Version control
- Tooling API
- Developer Console

Licenses

Licenses that are defined for an extension package aren't enforced on custom metadata records in that package unless the types are also in the package.

SOQL

Custom metadata types support the following SOQL query syntax.

```
SELECT fieldList [...]
FROM objectType
    [USING SCOPE filterScope]
[WHERE conditionExpression]
[ORDER BY field {ASC|DESC} [NULLS {FIRST|LAST}] ]
```

- The *`fieldList`* can include only non-relationship fields.
- `FROM` can include only one object.
- You can't use `COUNT` with custom metadata types.
- You can use the following operators.
  - `IN` and `NOT IN`
  - =, >, >=, <, <=, and `!=`
  - `LIKE`, including wild cards
  - `AND`
- You can use `ORDER BY`, `ASC`, and `DESC` with multiple fields.

Protected custom metadata types

Subscribers can't add custom metadata records to installed custom metadata types that are protected. To allow subscribers to create custom metadata records that are defined by a custom metadata type, the type must be public.

Metadata API returns protected custom entity definitions (but not custom metadata records) in subscriber organizations.

Caching

Custom metadata records are cached at the type level after the first read request. This enhances performance on subsequent requests. Requests that are in flight when metadata is updated won’t get the most recent metadata.

CUSTOM METADATA LIMITS

Be aware of these requirements for custom metadata types and records.

EDITIONS

Available in: Salesforce Classic

Available in:
- Enterprise
- Performance
- Unlimited
- Developer
- Database.com

Description	Maximum amount
SOQL queries per Apex transaction	Unlimited
Custom metadata per organization *	10 MB
Custom metadata per certified managed package *	10 MB Note: Custom metadata records in certified managed packages that you've installed don't count toward your organization's limit. However, custom metadata records that you create do count toward the limit. This rule applies regardless of whether you create records in your own custom metadata type or in a type from a certified managed package.
Fields per custom metadata type or record	100
Custom metadata types per organization	100. This number includes all types developed in the organization and installed from managed and unmanaged packages.
Characters per description field	1,000
Records returned per transaction	50,000
Custom metadata types in one call	200

* Record size is based on the maximum field size of each field type, not the actual storage that's used in each field. When adding fields to a custom metadata record, use the appropriate type and specify a length that doesn't exceed what's needed for your data. This action helps you avoid reaching the cached data limit. For example, if you create a US social security number (SSN) field, select the `Text` data type and specify a length of 9. If instead you selected `Text Area`, the field would add 255 characters to the usage count for each record, regardless of the number of characters entered.

GET STARTED WITH THE SAMPLE APPLICATION

Sample Application

The sample application is based on a collaborative effort between two fictional organizations. Picklists R Us develops reusable enhancements to the Salesforce App Cloud that involve picklist-related functionality. TravelApp, Inc. develops an interplanetary travel application that uses picklist features from Picklists R Us. Galactic Tours is a customer of these organizations. Galactic Tours installs Picklists R Us's package and TravelApp, Inc.'s extension into its organization.

Sample Application Objects and Fields

The sample application is based on three imaginary companies.

- Picklists R Us creates reusable picklists by using custom metadata types.
- TravelApp uses picklists from Picklists R Us to build an application for travel agencies.
- Galactic Tours uses TravelApp to book tours in outer space.

Picklists R Us creates three custom metadata types.

- `ReusablePicklist__mdt` defines the picklists.
- `ReusablePicklistOption__mdt` defines items and associates them with picklists.
- `PicklistUsage__mdt` associates picklists with objects.

TravelApp populates the custom metadata types with custom metadata records that define picklists and how they are used in the application. The following tables show how TravelApp uses the custom metadata types created by Picklists R Us.

Table 1: Reusable_Picklist__mdt

Developer Name	Label	AlphaSort__c	Comments
TestPicklistAlpha	Alpha Sorted Test Picklist	True	Picklists R Us framework
TestPicklistNonAlpha	Non-Alpha Sorted Test Picklist	False	Picklists R Us framework
Planets	Planets	False	TravelApp application
Hotels	Hotels	True	TravelApp application

Table 2: Reusable_PicklistOption__mdt

Developer Name	Label	Picklist__c	SortOrder__c	Comments
AlphaTestValue1	Test Value 1	TestPicklistAlpha		Picklists R Us framework
AlphaTestValue2	Test Value 2	TestPicklistAlpha		Picklists R Us framework
NonAlphaTestValue1	B Test Value 1	TestPicklistNonAlpha	1	Picklists R Us framework
NonAlphaTestValue2	A Test Value 2	TestPicklistNonAlpha	2	Picklists R Us framework
Mercury	Mercury	Planets	1	TravelApp application

Developer Name	Label	Picklist__c	SortOrder__c	Comments
Venus	Venus	Planets	2	TravelApp application
Motel6	Motel 6	Hotels		TravelApp application
Bellagio	Bellagio	Hotels		TravelApp application

Table 3: PicklistUsage__mdt

Developer Name	Label	Picklist__c	sObjectType__c	Field__c	Comments
AlphaTestUsage	Alpha Sorted Test Picklist Usage	TestPicklistAlpha	PicklistTestData__c	AlphaTestField__c	Picklists R Us framework
NonAlphaTestUsage	Non-Alpha Sorted Test Picklist Usage	TestPicklistAlpha	PicklistTestData__c	NonAlphaTestField__c	Picklists R Us framework
BookingDestination	Interplanetary Booking: Destination	Planets	InterplanetaryBooking__c	Destination__c	TravelApp application
BookingHotel	Interplanetary Booking: Hotel	Hotels	InterplanetaryBooking__c	Hotel__c	TravelApp application
GreetingPlanetVisited	Interplanetary Greeting: Planet Visited	Planets	InterplanetaryGreeting__c	PlanetVisited__c	TravelApp application

TravelApp creates the object `InterplanetaryGreeting__c` to contain greetings entered by travel agents. Galactic Tours uses this object to greet visitors to different planets. The field `Greeting__c` is a formula that adds the word `Hello` to the name of the planet the guest has visited. The following table shows how Galactic Tours populates the `InterplanetaryGreeting__c` object. The `InterplanetaryGreeting__c` object uses the `Planets` picklist to populate the `PlanetVisited__c` field.

Table 4: InterplanetaryGreeting__c

Name	Guest__c (lookup)	PlanetVisited__c	Greeting__c (formula)	Comments
John's Earth Greeting	John Many Jars	Earth	Hello Earth!	TravelApp user data
Skip's Jupiter Greeting	Skip Orbit	Jupiter	Hello Jupiter!	TravelApp user data
Luna's Neptune Greeting	Luna Darkside	Neptune	Hello Neptune!	TravelApp user data

TravelApp creates the object `InterplanetaryBooking__c` to contain trips entered by travel agents. Galactic Tours uses this object to store its bookings.

Table 5: InterplanetaryBooking__c

Name	Traveller__c (lookup)	Destination__c	Hotel__c	Departure__c	Return__c	Comments
John's Trip to Earth	John Many Jars	Earth	Motel6	6/15/2015	6/25/2015	TravelApp user data

Name	Traveller__c (lookup)	Destination__c	Hotel__c	Departure__c	Return__c	Comments
Luna's Trip to Neptune	Luna Darkside	Neptune	Bellagio	7/21/2015	8/21/2015	TravelApp user data

Installing the Sample Application

Visit the Custom Metadata Types community group at `success.salesforce.com` to get your own copy of the sample application and discuss this functionality. To explore the sample application's functionality, you need separate test organizations for Picklists R Us TravelApp, and Galactic Tours. The Picklists R Us and TravelApp organizations must be Developer Edition organizations.

The sample application is distributed as two metadata packages, `picklistsRUs.zip` and `travelApp.zip`, and a Perl script, `updateSampleAppWithNs.pl`. All three are in the `sampleApp.zip` file. The Perl script updates the files in `travelApp.zip` so that they correctly refer to the namespace of your Picklists R Us organization. The Perl script works in a UNIX-type shell, such as the BASH shell in Linux or the Terminal utility in OS X. If you are installing the sample application under a Microsoft Windows operating system, we recommend a UNIX-type shell such as Cygwin or Git BASH.

These directions are for deploying the application into your organizations and getting it working. The application components are explained in this implementation guide and the *Metadata API Developer's Guide*.

Deploy and Upload the Package for Picklists R Us

The foundation of the sample application is Picklists R Us's package, `picklistsRUs.zip`. Deploy and upload Picklists R Us's package first.

1. Connect to one of your organizations (here referred to as the Picklists R Us organization) via the Workbench tool. (For more information, see developer.salesforce.com/page/Workbench.) Verify that the API version of your connection is 34.0 or later.
2. From the Deploy page in Workbench (**Migration** > **Deploy**), stage and deploy the `picklistsRUs.zip` file. Select the **Single Package** option. This action uploads all picklist package components and adds them to an unmanaged package that's named Picklists R Us.
3. Log in to your Picklists R Us organization.
4. From Setup, enter *`Packages`* in the `Quick Find` box, then select **Packages**.
5. In the Developer Settings section, click **Edit**.
6. Choose a namespace prefix for this organization, and then select the Picklists R Us package to manage. Click **Review My Selections**, and then click **Save**.
7. In the Packages section, click **Picklists R Us**.
8. Click **Upload**, select **Managed - Released**—this option is required to upload extensions of the package—and then click **Upload**.
9. When the upload is complete, make a note of the package installation URL.
10. Log out of the Picklists R Us organization.

You're now ready to install TravelApp's package.

Deploy and Upload an Extension Package for TravelApp, Inc.

TravelApp, Inc. develops an interplanetary travel application that uses picklist features from Picklists R Us.. Install TravelApp's extension package.

Before you deploy TravelApp's extension, be sure to deploy and upload the Picklists R Us package.

1. Log in to your second Developer Edition organization, referred to here as the TravelApp organization.
2. Go to the package installation URL that you noted in Deploy and Upload the Package for Picklists R Us. Follow the steps to install the Picklists R Us package.
3. Verify that you have write permission on the `travelApp.zip` file.
4. In the directory that contains the `travelApp.zip` file, run the Perl script specifying the Picklists R Us package's namespace as an argument. For example, if the Picklists R Us namespace is `picklist1234` and you're in the directory with the Perl script and travelApp.zip file, use the syntax: `perl updateSampleAppWithNs.pl picklist1234`.

   Note: The Perl script works in a UNIX-type shell, such as the BASH shell in Linux or the Terminal utility in OS X. If you are installing the sample application under a Microsoft Windows operating system, we recommend a UNIX-type shell such as Cygwin or Git BASH.

5. Connect to the TravelApp organization via the Workbench tool.
6. From the Deploy page in Workbench (**Migration** > **Deploy**), stage and deploy the `travelApp.zip` file. Use the **Single Package** option. This action uploads all travel application components.
7. Log in to the TravelApp organization.
8. From Setup, enter *`Packages`* in the `Quick Find` box, then select **Packages**.
9. In the Developer Settings section, click **Edit**.
10. Choose a namespace prefix for this organization, and then select the Travel App package to manage.
11. Click **Review My Selections**, and then click **Save**.
12. From Setup, enter *`Tabs`* in the `Quick Find` box, then select **Tabs**.
13. Edit the Interplanetary Bookings tab.
14. Move through the wizard to the Edit Tab page.
15. On the Edit Tab page, in the Button or Link URL field, change the type parameter to the namespace that you set for TravelApp, and save your changes. This action ensures that the URL continues to work in organizations that install the tab. For example, if the Travel App Package namespace is travelApp1234, change this URL to `/apex/picklist1234__GenericTab?type=travelApp1234__InterplanetaryBooking__c`.
16. Repeat steps 11–14 for the Interplanetary Greetings tab.
17. Click **Upload**, and then follow the steps to upload a version of the package. This version can be released or beta. When the upload is complete, make a note of the package installation URL.
18. Log out of the TravelApp organization.

You're now ready to install and use the base package and extension for Galactic Tours.

Install and Use the Base Package and Extension for Galactic Tours

Galactic Tours is a customer of Picklists R Us and TravelApp. Install the base package and extension that was created by Galactic Tours.

Before you install the Galactic Tours package, be sure to deploy and upload the Picklists R Us package and deploy and upload a TravelApp extension package.

1. Verify that you're logged out of your other two organizations. Then log in to your third organization, referred to here as the Galactic Tours organization.
2. Go to the package installation URL that you noted in Deploy and Upload the Package for Picklists R Us, and then follow the steps to install the Picklists R Us package.
3. Go to the package installation URL that you noted in Deploy and Upload an Extension Package for TravelApp, Inc., and then follow the steps to install the TravelApp package.
4. From Setup in the TravelApp organization, enter `Users` in the `Quick Find` box, then select **Users**.
5. Click the link to open your user page.
6. Under Permission Set Assignments, click **Edit Assignments**.
7. Add "Travel App" and "Reusable Picklists End-User" to the Enabled Permission Sets list.
8. From the Force.com App menu, select the application TravelApp, Inc.
9. Click the **Interplanetary Greetings** tab.
10. When the tab opens, click **New**.
11. Fill in all fields, and then click **Save**. The Planet Visited picklist now contains a list of planets.
12. Repeat steps 9–11 for the Interplanetary Bookings tab.

    The Destination picklist is populated with the same list of options as the Planet Visited picklist. The Hotel picklist is populated with another set of options.

CREATE, EDIT, AND DELETE CUSTOM METADATA TYPES AND RECORDS

To create, update, and delete custom metadata types and records, use the Metadata API.

For more information, see "Custom Metadata Types (CustomObject)" in the *Metadata API Developer's Guide*

EDITIONS

Available in: Salesforce Classic

Available in: **Professional**, **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

Professional Edition organizations can create, edit, and delete only custom metadata records from types in installed packages.

IN THIS SECTION:

Define a Custom Metadata Type Declaratively

Use Salesforce UI to create and update custom metadata types declaratively.

EDITIONS

Available in:

USER PERMISSIONS

To create or edit custom metadata types:

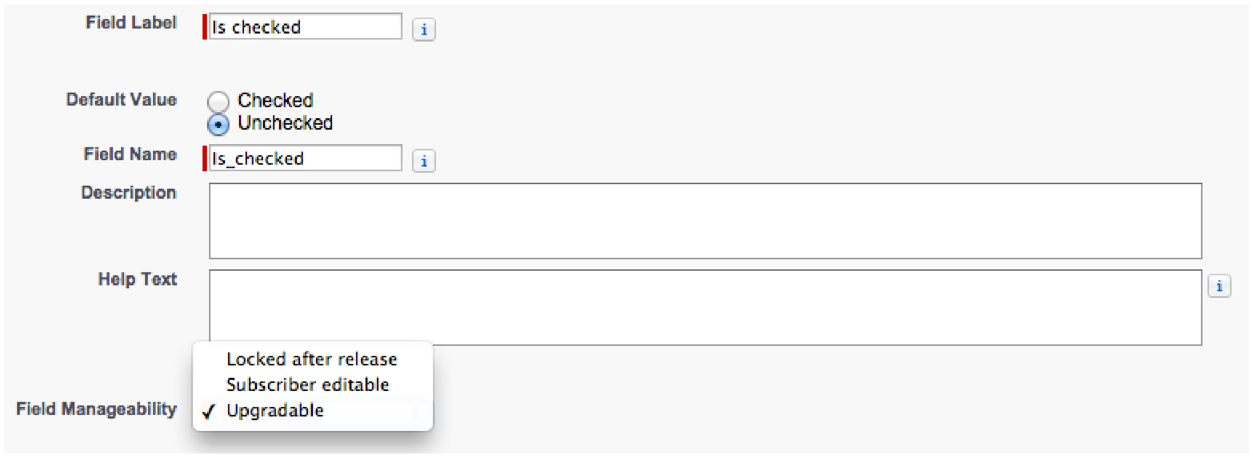
- "Author Apex"

1. Search Setup for `Custom Metadata Types`.
2. On the All Custom Metadata Types page, click **New Custom Metadata Type**, or click **Edit** to modify an existing custom metadata type.
3. Complete these fields.

Field	Description
`Label`	This name is used to refer to the type in a user interface page.
`Plural Label`	The plural name of the type. If you create a tab for this type, this name is used for the tab.
`Starts with a vowel sound`	If it is appropriate for your organization's default language, indicate whether the label is preceded by "an" instead of "a."
`Object Name`	A unique name used to refer to the object when using the API. In managed packages, this name prevents naming conflicts with package installations. Use only alphanumeric characters and underscores. The name must begin with a letter and have no spaces. It cannot end with an underscore nor have two consecutive underscores.
`Description`	An optional description of the object. A meaningful description helps you remember the differences between your custom objects when you are viewing them in a list.

Field	Description
`Context-Sensitive Help Setting`	Defines what displays when a user clicks **Help for this Page** from the custom object record home (overview), edit, and detail pages, list views, and related lists. To display the standard Salesforce help available for any custom object record, select **Open the standard Salesforce Help & Training window**. To display custom object-level help for your custom object, select **Open a window using a Visualforce page** and then select the Visualforce page to use as the target of the context-sensitive help link from that custom object's pages. Note: This setting doesn't affect the Help & Training link at the top of a page. That link always opens the Salesforce Help & Training window.
`Content Name`	The name used in page layouts, list views, related lists, and search results.
`Visibility`	Who should see the type: • Public—anyone can see it. • Protected—if the type is installed as part of a managed package, only Apex code in that managed package can use it.

4. Click **Save**.
5. Under `Custom Fields`, click **New** to start adding fields to the custom metadata type. For each field, remember to choose a **Field Manageability** value to determine who can change the field later.


Field Label: Is checked

Default Value: Checked [✓ Unchecked]

Field Name: Is_checked

Description:

Help Text:

Field Manageability: Locked after release / Subscriber editable / [✓ Upgradable]

Note: Custom metadata types that were created before the Winter '15 release don't automatically get layouts. Before adding records to this kind of custom metadata type using the UI, you must add a layout that contains all the fields that you want to make editable for the custom metadata type. In the All Custom Metadata Types page, click the custom metadata type. Then click **New** under Page Layouts. If you plan to release a custom metadata type as a managed package, make sure you add all the fields you want to add first. After a customer downloads the managed package, any changes to the layout must be done manually because you can't add fields to a layout via an upgrade.

Add or Edit Custom Metadata Records Declaratively

You can add, delete, or modify custom metadata declaratively from Setup.

1. Search Setup for `Custom Metadata Types`.
2. Click **Manage Records** next to the type of custom metadata that you want to add or modify.
3. On the list of custom metadata records, click **New**, or click **Edit** to modify an existing custom metadata record.
4. Fill out the fields. The **Protected Component** checkbox determines whether the record is *protected*. A protected record is only accessible to code in the same namespace as either the record or its associated custom metadata type: code you create, code in an unmanaged package, and code in the same managed package as either the protected record or its custom metadata type.
5. Click **Save**.

EDITIONS

Available in:
- Professional
- Enterprise
- Performance
- Unlimited
- Developer
- Database.com

USER PERMISSIONS

To create or modify custom metadata records:
- "CustomizeApplication"

ACCESS CUSTOM METADATA TYPES, RECORDS, AND FIELDS

Access Custom Metadata Types and Records

Use SOQL to access your custom metadata types and to retrieve the API names of the records on those types. DML operations aren't allowed on custom metadata in Apex, the Partner APIs, and Enterprise APIs.

For information about the `Custom Metadata Type__mdt` sObject, see *`Custom Metadata Type__mdt`* in the *Object Reference for Salesforce and Force.com*..

For example, declare an Apex variable *`custMeta`* of the custom metadata type `MyCustomMetadataType__mdt`, which is in your namespace, as follows.

EDITIONS

Available in: Salesforce Classic

Available in: **Professional**, **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

Professional Edition organizations can access only custom metadata records from installed custom metadata types.

```
MyCustomMetadataType__mdt custMeta;
```

Declare the *`custMeta`* variable of the custom metadata type `TheirCustomMetadataType__mdt`, which isn't in your namespace but is in the `their_ns` namespace, as follows.

```
their_ns__TheirCustomMetadataType__mdt custMeta;
```

To get the names of all objects of the `MyMdt__mdt` custom metadata type:

```
MyMdt__mdt[] allEntityNames = [select QualifiedApiName from MyMdt__mdt]
```

You can't use `queryMore()` with custom metadata, but you can use the SOQL keywords `LIMIT` and `OFFSET` to page through large numbers of records. For more information, see Paginating Data for Force.com Applications.

Alternatively, to provide an entity that looks more like a `Schema.SObjectDescribeResult` than SOQL, make the Apex class `Acme.MyMdtDescribeResult` encapsulate the information queried from `Acme__MyMdt`. Then create the class `Acme.Acme` with methods such as:

```
Acme.MyMdtDescribeResult describeMyMdt(String qualifiedApiName) {
   ///perform queries and create object
}
```

Access Custom Metadata Fields

Read-only access to the fields on your custom metadata types and records is available through SOQL.

Custom fields on custom metadata types in SOQL are referred to in the same way as they are in the Metadata API. For example, the following SOQL statement retrieves all `Field__c` and `Picklist__c` values of any `PicklistUsage__mdt` related to any custom object named `InterplanetaryGreeting__c`.

EDITIONS

Available in: Salesforce Classic

Available in: **Professional**, **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

Professional Edition organizations can access only custom metadata fields from installed custom metadata types.

```
SELECT Field__c, Picklist__c
   FROM PicklistUsage__mdt
   WHERE SObjectType__c = 'InterplanetaryGreeting__c'
```

The information that's common to all custom metadata is represented as standard fields. For more information, see "*`Custom Metadata Type`*__mdt" in the *Object Reference for Salesforce and Force.com*.

The following Apex statement in the `picklist1234` namespace retrieves the label and namespace for the custom metadata that's represented in the file-based Metadata API as `picklist1234__ReusablePicklistOption.travelApp1234__Motel6`. This statement assigns the object to the variable `motelEx`.

```
ReusablePicklistOption__mdt motelEx = [SELECT MasterLabel, NamespacePrefix
                                       FROM ReusablePicklistOption__mdt
                                       WHERE NamespacePrefix = 'travelApp1234'
                                       AND DeveloperName='Motel6'];
```

Note: Subscribers can run packaged Apex code that queries protected custom metadata types in the same package. However, subscribers can't query protected types in an installed package by using Apex code that they have written.

PACKAGE CUSTOM METADATA TYPES AND RECORDS

EDITIONS

Available in: Salesforce Classic

Available in:
- Enterprise
- Performance
- Unlimited
- Developer
- Database.com

You can package custom metadata types and records in unmanaged packages, managed packages, or managed package extensions. Your packages can then be installed in Professional, Developer, Enterprise, Performance, Unlimited, and Database.com Edition organizations. Use change sets to deploy custom metadata types and records from a sandbox.

You can add custom metadata types and records to packages using the Force.com user interface. From Setup, enter *`Packages`* in the `Quick Find` box, then select **Packages**, click your package name, and then click **Add**.

Then, to add custom metadata types:

1. Select the **Custom Metadata Type** component type.
2. Select the custom metadata type to add to your package.
3. Click **Add to Package**.

To add custom metadata records:

1. Select the custom metadata type's label from the available component types—for example, `ReusablePicklist__mdt`, or if the type is from a package that you're extending, `ReusablePicklist__mdt [picklist1234]`.
2. Select the records to add.
3. Click **Add to Package**.

If you add a record to your package, its corresponding type is added. If you add a record to a change set, its corresponding type is included in the list of dependent components.

For information on change sets and deploying your package, see the *Development Lifecycle Guide*.

Note: You can't uninstall a package with a custom metadata type if you've created your own records of that custom metadata type.

As with all packageable metadata components, you can also add custom metadata types and records to a package by specifying the package's full name in your `package.xml` file. For example, we specify the package in this fragment from Picklists R Us's `package.xml` file.

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

IN THIS SECTION:

Considerations for Custom Metadata Type Packages
Be aware of the following behaviors for packages that contain custom metadata types.

Considerations for Custom Metadata Type Packages

Be aware of the following behaviors for packages that contain custom metadata types.

Once you upload a Managed - Released package that contains a custom metadata type, you can't:

- Add required fields to the custom metadata type
- Set any non-required fields to required
- Delete custom fields

EDITIONS

Available in: Salesforce Classic

Available in:

- Enterprise
- Performance
- Unlimited
- Developer
- Database.com