Class

ModelContainer

An object that manages an app’s schema and model storage configuration.

iOS 17.0+ | iPadOS 17.0+ | Mac Catalyst 17.0+ | macOS 14.0+ | tvOS 17.0+ | visionOS 1.0+ | watchOS 10.0+ | Swift 5.9+

```
class ModelContainer
```

Mentioned in

- Reverting data changes using the undo manager
- Syncing model data across a person’s devices

Overview

A model container mediates between its associated model contexts and your app’s underlying persistent storage. The container manages all aspects of that storage and ensures it remains in a consistent and usable state. Whenever you run a fetch or call a context’s `save()` method, the container performs the actual read or write of the underlying data using information from the schema you provide. This helps safeguard an app’s resources and ensures those operations happen only in an efficient and coordinated manner. Additionally, if your app’s entitlements include CloudKit, the container automatically handles syncing the persisted storage across devices. For more information about syncing model data, see Syncing model data across a person’s devices.

As your app’s schema evolves, the container performs automatic migrations of the persisted model data so it remains consistent with the app’s model classes. If the aggregate changes between two versions of your schema exceed the capabilities of automatic migrations, provide the container with a `SchemaMigrationPlan` to participate in those migrations and help ensure they complete successfully.

By default, a model container makes a number of assumptions about how it configures an app's persistent storage. If you need to customize this behavior, provide the container with one or more instances of `ModelConfiguration`. For example, you may want use a particular app group container or specify that the storage is ephemeral and should exist only in memory.

An app that uses SwiftData requires at least one model container. You create a container using one of the class's initializers or the corresponding SwiftUI view modifier. Using the view modifier ensures all windows in the modified window group, or all child views of the modified view, access the same model container. Additionally, the view modifier makes an associated model context available in the SwiftUI environment, which the `Query()` macro depends on.

```
@main
struct RecipesApp: App {
    var body: some Scene {
        WindowGroup {
            RecipesList()
        }
        .modelContainer(for: Recipe.self)
    }
}

struct RecipesList: View {
    @Query private var recipes: [Recipe]

    var body: some View {
        List(recipes) { RecipeRowView($0) }
    }
}
```

Topics

Creating a model container

`init(for: Schema, migrationPlan: (any SchemaMigrationPlan.Type)?, configurations: [ModelConfiguration]) throws`

Creates a model container using the specified schema, migration plan, and configurations.

`convenience init(for: any PersistentModel.Type..., migrationPlan: (any SchemaMigrationPlan.Type)?, configurations: ModelConfiguration...) throws`

Creates a model container using the specified model types, migration plan, and zero or more configurations.

`convenience init(for: Schema, migrationPlan: (any SchemaMigrationPlan.Type)?, configurations: ModelConfiguration...) throws`

Creates a model container using the specified schema, migration plan, and zero or more configurations.

`protocol PersistentModel`

An interface that enables SwiftData to manage a Swift class as a stored model.

`struct ModelConfiguration`

A type that describes the configuration of an app's schema or specific group of models.

`class Schema`

An object that maps model classes to data in the model store, and helps with the migration of that data between releases.

`protocol SchemaMigrationPlan`

An interface for describing the evolution of a schema and how to migrate between specific versions.

Managing schema and configuration details

`let schema: Schema`

The schema that maps your app's model classes to the associated data in the app's persistent storage.

`var configurations: Set<ModelConfiguration>`

The configurations that describe how to manage the persisted data for specific groups of models.

`let migrationPlan: (any SchemaMigrationPlan.Type)?`

The plan that describes the evolution of your app's schema and how to migrate between specific versions.

Accessing the context

`var mainContext: ModelContext`

A model context that's bound to app's main actor.

Deleting the container

`func deleteAllData()`

Removes all persisted model data from the app's persistent storage.

Initializers

`convenience init(for: any PersistentModel.Type..., configurations: any DataStoreConfiguration...) throws`

`init(for: Schema, configurations: [any DataStoreConfiguration]) throws`

Instance Methods

`func erase() throws`

Relationships

Conforms To

`Equatable`, `Sendable`, `SendableMetatype`

See Also

Model life cycle

`class ModelContext`

An object that enables you to fetch, insert, and delete models, and save any changes to disk.

Fetching and filtering time-based model changes

Track all inserts, updates, and deletes that occur in a data store and process them as a series of chronological transactions.

`struct HistoryDescriptor`

A type that describes the criteria, and, optionally, sort order, to use when fetching history data

{} Deleting persistent data from your app

Explore different ways to use SwiftData to delete persistent data.

Reverting data changes using the undo manager

Automatically record data change operations that people perform in your SwiftUI app, and let them undo and redo those changes.

Syncing model data across a person’s devices

Add the required capabilities and define a compatible schema to enable SwiftData to automatically sync your app’s model data using iCloud.

Concurrency support

Types you use to access model attributes and perform storage-related tasks in a safe and isolated way.