Framework

SwiftData

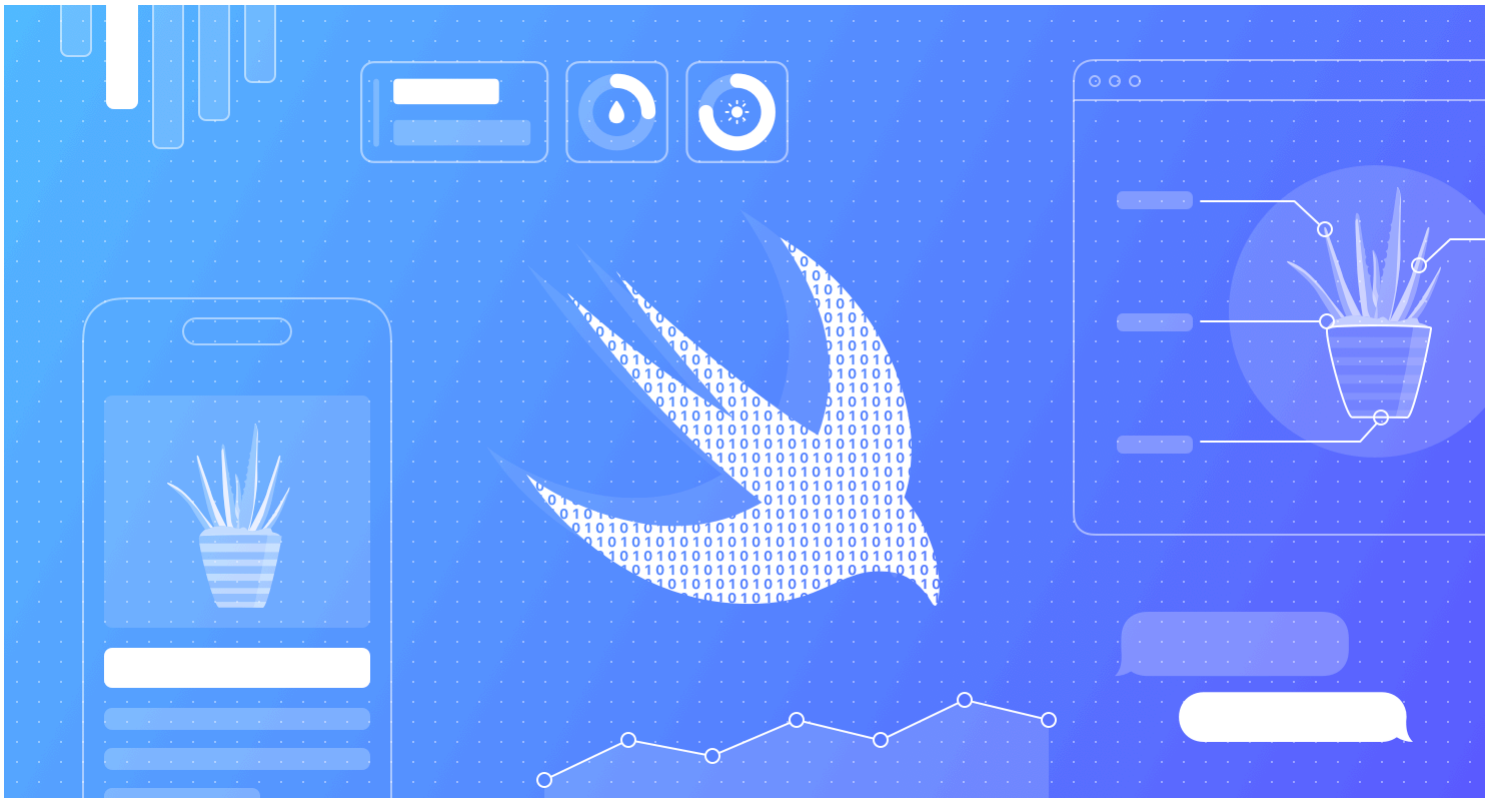
Write your model code declaratively to add managed persistence and efficient model fetching.

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

Overview

Combining Core Data's proven persistence technology and Swift's modern concurrency features, SwiftData enables you to add persistence to your app quickly, with minimal code and no external dependencies. Using modern language features like macros, SwiftData enables you to write code that is fast, efficient, and safe, enabling you to describe the entire model layer (or object graph) for your app. The framework handles storing the underlying model data, and optionally, syncing that data across multiple devices.

SwiftData has uses beyond persisting locally created content. For example, an app that fetches data from a remote web service might use SwiftData to implement a lightweight caching mechanism and provide limited offline functionality.



SwiftData is unintrusive by design and supplements your app's existing model classes. Attach the `Model()` macro to any model class to make it persistable. Customize the behavior of that model's properties with the `Attribute(_:originalName:hashModifier:)` and `Relationship(_:deleteRule:minimumModelCount:maximumModelCount:originalName:inverse:hashModifier:)` macros. Use the `ModelContext` class to insert, update, and delete instances of that model, and to write unsaved changes to disk.

To display models in a SwiftUI view, use the `Query()` macro and specify a predicate or fetch descriptor. SwiftData performs the fetch when the view appears, and tells SwiftUI about any subsequent changes to the fetched models so the view can update accordingly. You can access the model context in any SwiftUI view using the `modelContext` environment value, and specify a particular model container or context for a view with the `modelContainer(_:)` and `modelContext(_:)` view modifiers.

Topics

Essentials

Preserving your app's model data across launches

Describe your model classes to SwiftData using the framework's macros, and store instances of those models so they exist beyond the app's runtime.

Adding and editing persistent data in your app

Create a data entry form for collecting and changing data managed by SwiftData.

{} Adopting SwiftData for a Core Data app

Persist data in your app intuitively with the Swift native persistence framework.

SwiftData updates

Learn about important changes to SwiftData.

Adopting inheritance in SwiftData

Add flexibility to your models using class inheritance.

Model definition

`macro Model()`

Converts a Swift class into a stored model that's managed by SwiftData.

`macro Attribute(Schema.Attribute.Option..., originalName: String?, hashModifier: String?)`

Specifies the custom behavior that SwiftData applies to the annotated property when managing the owning class.

`macro Unique<T>([PartialKeyPath<T>]...)`

Specifies the key-paths that SwiftData uses to enforce the uniqueness of model instances.

`macro Index<T>([PartialKeyPath<T>]...)`

Specifies the key-paths that SwiftData uses to create one or more binary indices for the associated model.

`macro Index<T>(Schema.Index<T>.Types<T>...)`

Specifies the key-paths that SwiftData uses to create one or more indicies for the associated model, where each index is either binary or R-tree.

{} Defining data relationships with enumerations and model classes

Create relationships for static and dynamic data stored in your app.

`macro Relationship(Schema.Relationship.Option..., deleteRule: Schema.Relationship.DeleteRule, minimumModelCount: Int?, maximumModelCount: Int?, originalName: String?, inverse: AnyKeyPath?, hashModifier: String?)`

Specifies the options that SwiftData needs to manage the annotated property as a relationship between two models.

`macro Transient()`

Tells SwiftData not to persist the annotated property when managing the owning class.

Model life cycle

`class ModelContainer`

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

`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.

Model fetch

Filtering and sorting persistent data

Manage data store presentation using predicates and dynamic queries.

`macro Query()`

Fetches all instances of the attached model type.

Additional query macros

Supplementary macros that enable you to narrow query results and tell SwiftData how to sort and order those results.

`struct` `Query`

A type that fetches models using the specified criteria, and manages those models so they remain in sync with the underlying data.

`struct` `FetchDescriptor`

A type that describes the criteria, sort order, and any additional configuration to use when performing a fetch.

Model storage

{} Maintaining a local copy of server data

Create and update a persistent store to cache read-only network data.

`class` `DefaultStore`

A data store that uses Core Data as its undelying storage mechanism.

`protocol` `DataStore`

An interface that enables SwiftData to read and write model data without knowledge of the underlying storage mechanism.

`protocol` `DataStoreBatching`

An interface that enables a custom data store to support batch requests.

`protocol` `HistoryProviding`

An interface that enables a custom data store to provide the history of changes for its persisted models.

{} Building a document-based app using SwiftData

Code along with the WWDC presenter to transform an app with SwiftData.

`struct` `ModelDocument`

A document type that uses SwiftData to manage its storage.

History life cycle

`enum` `HistoryChange`

Values that describe data history transactions.

`protocol` HistoryDelete

An interface that enables a custom data store to delete items from the history of changes to its persisted models.

`protocol` HistoryInsert

`protocol` HistoryToken

`protocol` HistoryTransaction

`protocol` HistoryUpdate

`struct` HistoryTombstone

`struct` DefaultHistoryInsert

`struct` DefaultHistoryUpdate

`struct` DefaultHistoryDelete

`struct` DefaultHistoryToken

`struct` DefaultHistoryTransaction

Codeable support

`enum` DataStoreSnapshotCodingKey

The key space to use when implementing custom coders and decoders for data store snapshots,

Errors

`struct` SwiftDataError

A type that describes a SwiftData error.

`enum` DataStoreError

A type that describes a data store error.