**Using a EntityQuery to query data**

You can use an EntityQuery to do the following:

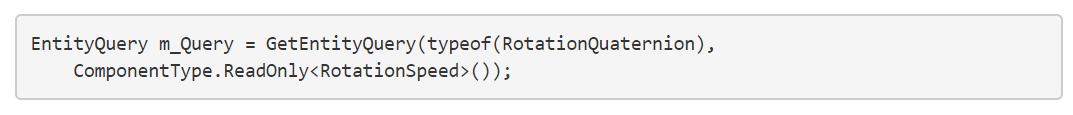
* Run a job to process the entities and components selected for the view
* Get a NativeArray that contains all of the selected entities
* Get NativeArrays of the selected components (by component type)

The entity and component arrays an EntityQuery returns are guaranteed to be "parallel", that is, the same index value always applies to the same entity in any array.

**Note:** The SystemBase.Entities.ForEach constructions create internal EntityQuery instances based on the component types and attributes you specify for these APIs. You cannot use a different EntityQuery object with Entities.ForEach, (though you can get the query object that an Entities.ForEach instance constructs and use it elsewhere).

**Defining a query**

The following example defines a EntityQuery that finds all entities with both RotationQuaternion and RotationSpeed components.



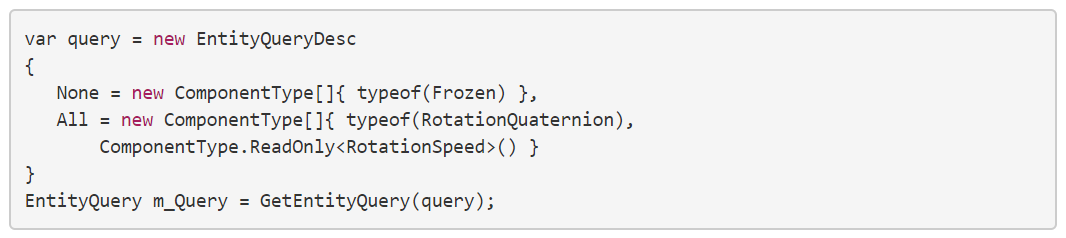
The query uses ComponentType.ReadOnly<T> instead of the simpler typeof expression to designate that the system does not write to RotationSpeed. Always specify read only when possible, because there are fewer constraints on read access to data, which can help the job scheduler execute the jobs more efficiently.

**EntityQueryDesc**

An EntityQueryDesc provides a flexible query mechanism to specify which archetypes to select based on the following sets of components:

* All: All component types in this array must exist in the archetype
* Any: At least one of the component types in this array must exist in the archetype
* None: None of the component types in this array can exist in the archetype

For example, the following query includes archetypes that contain the RotationQuaternion and RotationSpeed components, but excludes any archetypes that contain the Frozen component:

**Note:** Do not include optional components in the EntityQueryDesc. To handle optional components, use the ArchetypeChunk.Has<T>() method to determine whether a chunk contains the optional component or not. Because all entities within the same chunk have the same components, you only need to check whether an optional component exists once per chunk: not once per entity.

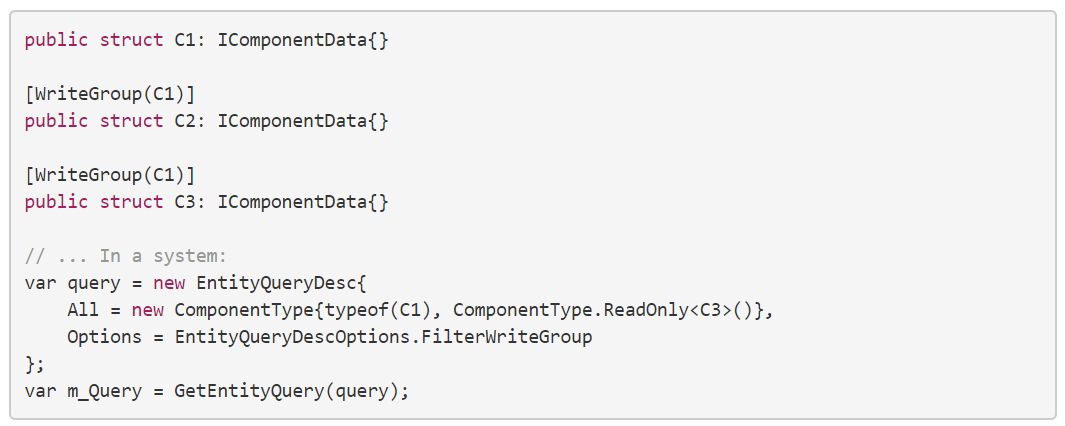
**Query options**

When you create an EntityQueryDesc, you can set its Options variable. The options allow for specialized queries (normally you do not need to set them):

* Default: No options set; the query behaves normally.
* IncludePrefab: Includes archetypes that contain the special Prefab tag component.
* IncludeDisabled: Includes archetypes that contain the special Disabled tag component.
* FilterWriteGroup: Considers the WriteGroup of any components in the query.

When you set the FilterWriteGroup option, only entities with those components in a Write Group that are explicitly included in the query are included in the view. ECS excludes any entities that have any additional components from the same WriteGroup.

In the following example, C2 and C3 are components in the same Write Group based on C1, and this query uses the FilterWriteGroup option that requires C1 and C3:

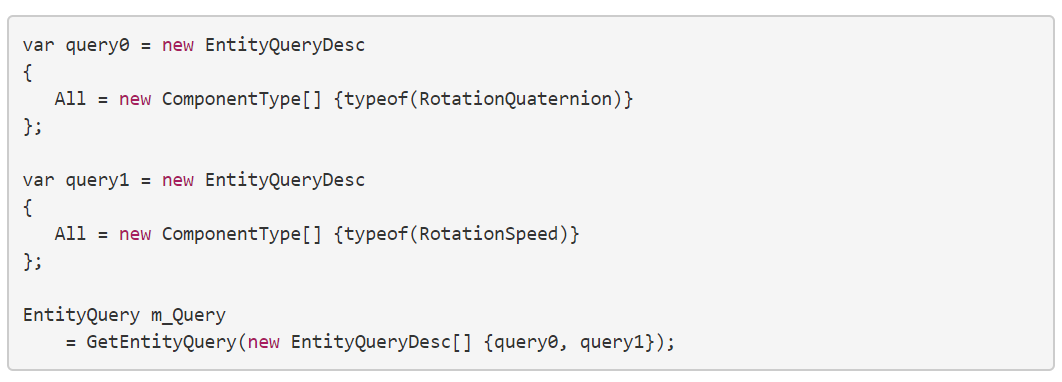
This query excludes any entities with both C2 and C3 because C2 is not explicitly included in the query. While you can use None to design this into the query, doing it through a Write Group provides an important benefit: you don't need to change the queries other systems use (as long as these systems also use Write Groups).

Write Groups are a mechanism that you can use to extend existing systems. For example, if C1 and C2 are defined in another system (perhaps part of a library that you don't control), you can put C3 into the same Write Group as C2 to change how C1 is updated. For any entities which you add to the C3 component, the system updates C1 and the original system does not. For other entities without C3, the original system updates C1 as before.

For more information, see [Write Groups](https://docs.unity3d.com/Packages/com.unity.entities@0.9/manual/ecs_write_groups.html).

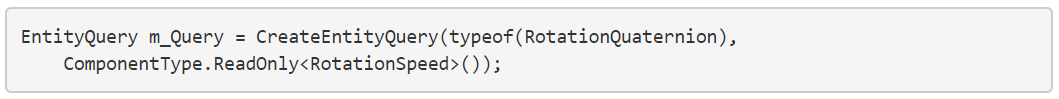
**Combining queries**

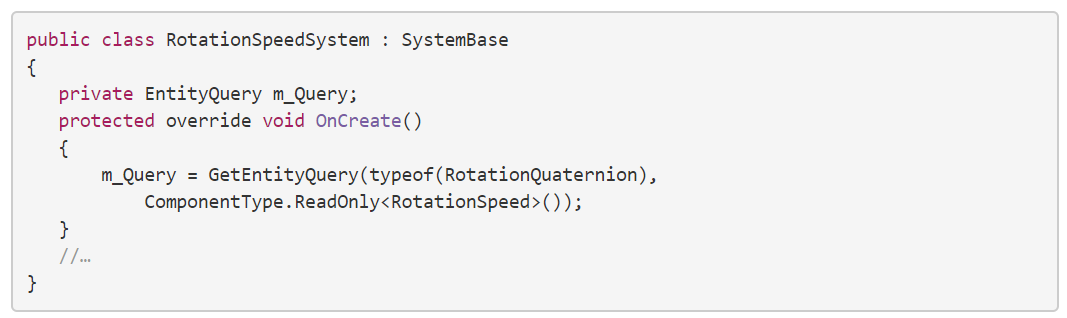
To combine multiple queries, you can pass an array of EntityQueryDesc objects rather than a single instance. You must use a logical OR operation to combine each query. The following example selects any archetypes that contain a RotationQuaternion component or a RotationSpeed component (or both):



**Creating a EntityQuery**

Outside of a system class, you can create a EntityQuery with the EntityManager.CreateEntityQuery() function as follows:

However, in a system class, you must use the GetEntityQuery() function for use with an IJobChunk job:

If you plan to reuse the same view, cache the EntityQuery instance, instead of creating a new one for each use. For example, in a system, you can create the EntityQuery in the system’s OnCreate() function and store the result in an instance variable. The m\_Query variable in the above example is used for this purpose.

Note that queries created for a system are cached by the system. GetEntityQuery() returns the existing query if one already exists rather than creating a new one. However, filter settings are not considered when evaluating whether two queries are the same. In additon, if you set filters on a query, the same filters are set the next time you access that same query with GetEntityQuery(). Use ResetFilter() to clear the existing filters.

**Defining filters**

You can filter the view as well as defining which components must be included or excluded from the query. You can specify the following types of filters:

* **Shared component filter**: Filter the set of entities based on specific values of a shared component.
* **Change filter**: Filter the set of entities based on whether the value of a specific component type has changed.

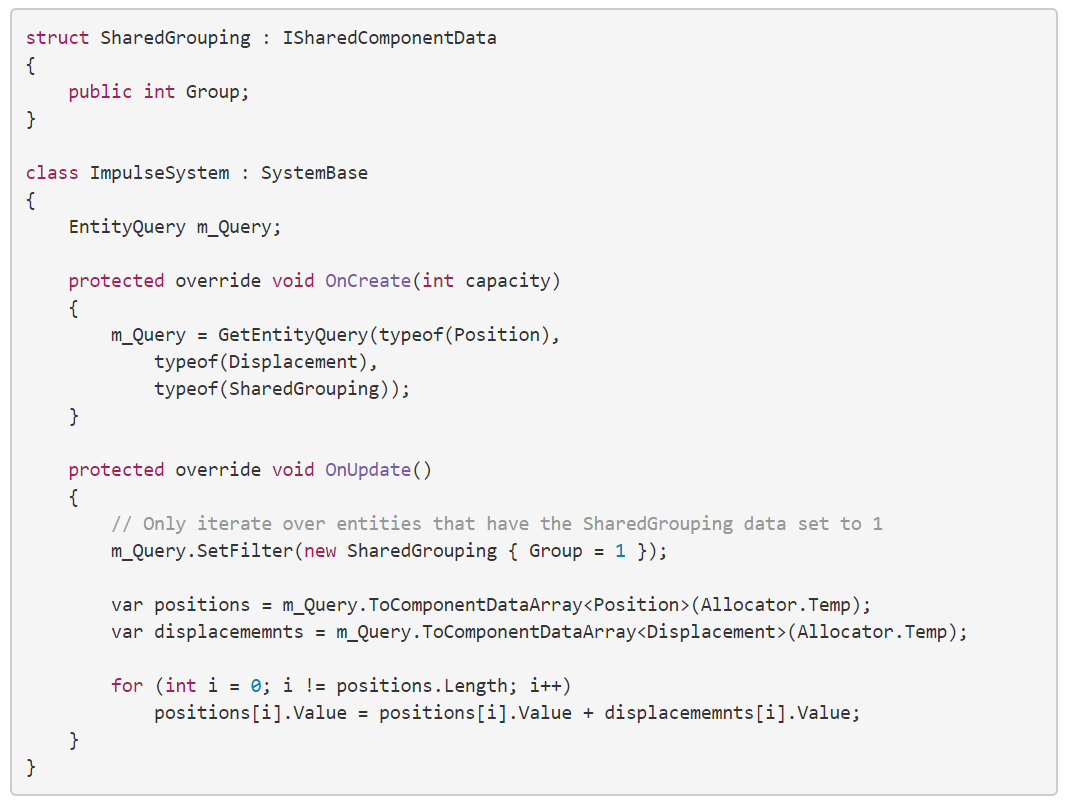
The filters you set remain in effect until you call ResetFilter() on the query object.

**Shared component filters**

To use a shared component filter, include the shared component in the EntityQuery (along with other needed components), and then call the SetFilter() function. Then pass in a struct of the same ISharedComponent type that contains the values to select. All values must match. You can add up to two different shared components to the filter.

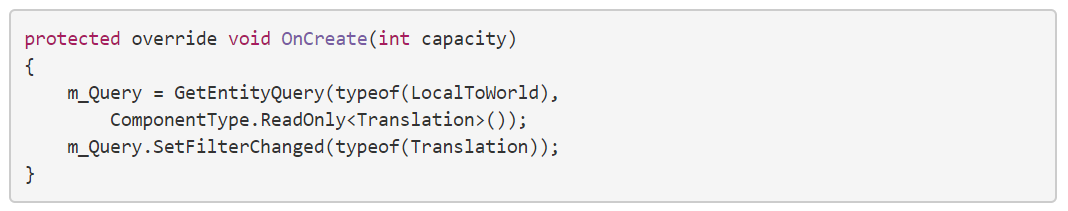
You can change the filter at any time, but if you change the filter, it does not change any existing arrays of entities or components that you received from the group ToComponentDataArray() or ToEntityArray() functions. You must recreate these arrays.

The following example defines a shared component named SharedGrouping and a system that only processes entities that have the Group field set to 1.



**Change filters**

If you only need to update entities when a component value has changed, you can add that component to the EntityQuery filter using the SetFilterChanged() function. For example, the following EntityQuery only includes entities from chunks that another system has already written to the Translation component:

**Note:** For efficiency, the change filter applies to whole chunks, not individual entities. The change filter also only checks whether a system has run that declared write access to the component, not whether it actually changed any data. In other words, if another job which had the ability to write to that type of component accesses the chunk, then the change filter includes all entities in that chunk. This is why you should always declare read only access to components that you do not need to modify.

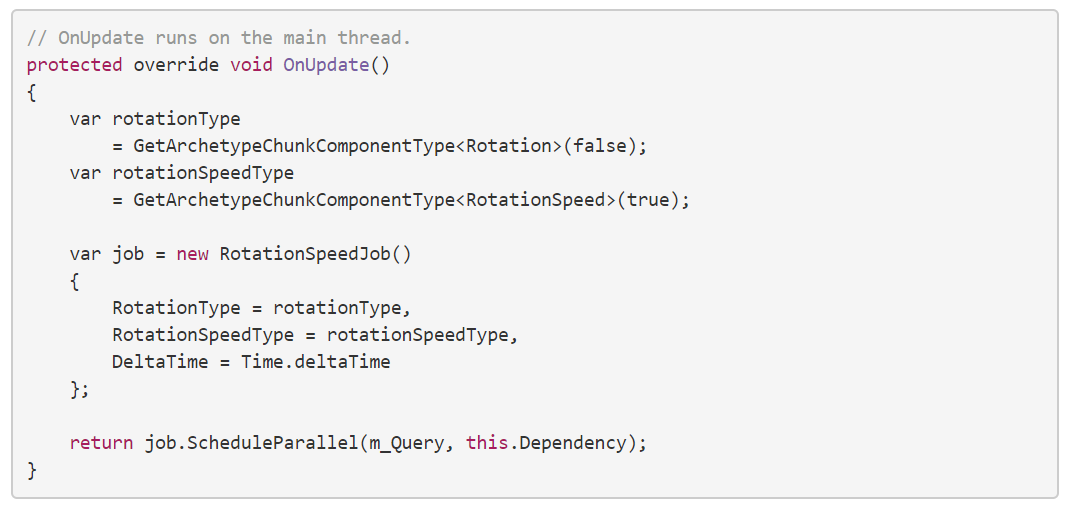
**Executing the query**

An EntityQuery executes its query when you use the EntityQuery in a job or you call one of the EntityQuery methods that returns arrays of entities, components, or chunks in the view:

* ToEntityArray() returns an array of the selected entities.
* ToComponentDataArray<T> returns an array of the components of type T for the selected entities.
* CreateArchetypeChunkArray() returns all of the chunks that contain the selected entities. Because a query operates on archetypes, shared component values, and change filters, which are all identical for all the entities in a chunk, the set of entities stored win the returned set of chunks is exactly the same as the set of entities ToEntityArray() returns .

**In jobs**

In a system that schedules an IJobChunk job, pass the EntityQuery object to the job's ScheduleParallel() or ScheduleSingle() methods. In the following example, from the HelloCube IJobChunk sample, the m\_Query argument is the EntityQuery object

An EntityQuery uses jobs internally to create the required arrays. When you pass the group to one of the Schedule() methods, ECS schedules the EntityQuery jobs along with the system's own jobs and as such you can take advantage of parallel processing.

记忆：

EntityQuery

EntityQueryDesc

Query Options

Filters

component filters

change filters