Dynamic (Object) Fields

There are various ways to use object fields to store primitive data and other objects (wrapping), but there are a few limitations to these:

Fortunately, Sui provides dynamic fields with arbitrary names (not just identifiers), added and removed on-the-fly (not fixed at publish), which only affect gas when they are accessed, and can store heterogeneous values. Use the libraries in this topic to interact with this kind of field.

There are two flavors of dynamic field -- "fields" and "object fields" -- which differ based on how you store their values:

Unlike an object's regular fields where names must be Move identifiers, dynamic field names can be any value that has copy , drop , and store . This includes all Move primitives (integers, Booleans, byte strings), and structs whose contents all have copy , drop , and store .

Use the add function from the relevant Sui framework module to add dynamic fields:

Dynamic field

Dynamic object field

These functions add a field with name name and value value to object . To see it in action, consider these code snippets:

First, define two object types for the parent and the child:

Next, define an API to add a Child object as a dynamic field of a Parent object:

This function takes the Child object by value and makes it a dynamic field of parent with name b"child" (a byte string of type vector). This call results in the following ownership relationship:

It is an error to overwrite a field (attempt to add a field with the same type and value as one that is already defined), and a transaction that does this fails. You can modify fields in-place by borrowing them mutably and you can overwrite them safely (such as to change its value type) by removing the old value first.

You can reference dynamic fields by reference using the following APIs:

Where object is the UID of the object the field is defined on and name is the field's name.

sui::dynamic_object_field has equivalent functions for object fields, but with the added constraint Value: key + store .

To use these APIs with the Parent and Child types defined earlier:

The first function accepts a mutable reference to the Child object directly, and you can call it with Child objects that haven't been added as fields to Parent objects.

The second function accepts a mutable reference to the Parent object and accesses its dynamic field using borrow_mut , to pass to mutate_child . This can only be called on Parent objects that have a b"child" field defined. A Child object that has been added to a Parent must be accessed via its dynamic field, so it can only be mutated using mutate_child_via_parent , not mutate_child , even if its ID is known.

A transaction fails if it attempts to borrow a field that does not exist.

The type passed to borrow and borrow_mut must match the type of the stored field, or the transaction aborts.

You must access dynamic object field values through these APIs. A transaction that attempts to use those objects as inputs (by value or by reference), is rejected for having invalid inputs.

Similar to unwrapping an object held in a regular field, you can remove a dynamic field, exposing its value:

This function takes a mutable reference to the ID of the object the field is defined on, and the field's name . If a field with a value: Value is defined on object at name , it is removed and value returned, otherwise it aborts. Future attempts to access this field on object will fail.

sui::dynamic_object_field has an equivalent function for object fields.

The value that is returned can be interacted with just like any other value (because it is any other value). For example, removed dynamic object field values can then be delete -d or transfer -ed to an address (back to the sender):

Similar to borrowing a field, a transaction that attempts to remove a non-existent field, or a field with a different Value type, fails.

It is possible to delete an object that has (potentially non- drop) dynamic fields still defined on it. Because field values can be accessed only via the dynamic field's associated object and field name, deleting an object that has dynamic fields still defined on it renders them all inaccessible to future transactions. This is true regardless of whether the field's value has the drop ability. This might not be a concern when adding a small number of statically known additional fields to an object, but is particularly undesirable for on-chain collection types that could be holding unboundedly many key-value pairs as dynamic fields.

Sui provides Table and Bag collections built using dynamic fields, but with additional support to count the number of entries they contain to protect against accidental deletion when non-empty. To learn more, see Tables and Bags .

Related links