The Move Book

Objects on Sui are explicit in their structure and behavior and can be displayed in an understandable way. However, to support richer metadata for clients, there's a standard and efficient way of "describing" them to the client - the Display object defined in the Sui Framework .

Historically, there were different attempts to agree on a standard structure of an object so it can be displayed in a user interface. One of the approaches was to define certain fields in the object struct which, when present, would be used in the UI. This approach was not flexible enough and required developers to define the same fields in every object, and sometimes the fields did not make sense for the object.

If any of the fields contained static data, it would be duplicated in every object. And, since Move does not have interfaces, it is not possible to know if an object has a specific field without "manually" checking the object's type, which makes the client fetching more complex.

To address these issues, Sui introduces a standard way of describing an object for display. Instead of defining fields in the object struct, the display metadata is stored in a separate object, which is associated with the type. This way, the display metadata is not duplicated, and it is easy to extend and maintain.

Another important feature of Sui Display is the ability to define templates and use object fields in those templates. Not only it allows for a more flexible display, but it also frees the developer from the need to define the same fields with the same names and types in every object.

The Object Display is natively supported by the Sui Full Node , and the client can fetch the display metadata for any object if the object type has a Display associated with it.

While the objects can be owned by accounts and may be a subject to True Ownership , the Display can be owned by the creator of the object. This way, the creator can update the display metadata and apply the changes globally without the need to update every object. The creator can also transfer Display to another account or even build an application around the object with custom functionality to manage the metadata.

The fields that are supported most widely are:

Please, refer to the Sui Documentation for the most up-to-date list of supported fields.

While there's a standard set of fields, the Display object does not enforce them. The developer can define any fields they need, and the client can use them as they see fit. Some applications may require additional fields, and omit other, and the Display is flexible enough to support them.

The Display object is defined in the sui::display module. It is a generic struct that takes a phantom type as a parameter. The phantom type is used to associate the Display object with the type it describes. The fields of the Display object are a VecMap of key-value pairs, where the key is the field name and the value is the field value. The version field is used to version the display metadata, and is updated on the update_display call.

File: sui-framework/sources/display.move

The Publisher object is required to a new Display, since it serves as the proof of ownership of type.

Currently, Display supports simple string interpolation and can use struct fields (and paths) in its templates. The syntax is trivial - {path} is replaced with the value of the field at the path. The path is a dot-separated list of field names, starting from the root object in case of nested fields.

The Display for the type LittlePony above could be defined as follows:

There's no restriction to how many Display objects can be created for a specific T . However, the most recently updated Display will be used by the full node.

Background

Historically, there were different attempts to agree on a standard structure of an object so it can be displayed in a user interface. One of the approaches was to define certain fields in the object struct which, when present, would be used in the UI. This approach was not flexible enough and required developers to define the same fields in every object, and sometimes the fields did not make sense for the object.

bash /// An attempt to standardize the object structure for display. public struct CounterWithDisplay has key { id: UID, /// If this field is present it will be displayed in the UI as `name`. name: String, /// If this field is present it will be displayed in the UI as `description`. description: String, // ... image: String, /// Actual fields of the object. counter: u64, // ... }

If any of the fields contained static data, it would be duplicated in every object. And, since Move does not have interfaces, it is not possible to know if an object has a specific field without "manually" checking the object's type, which makes the client fetching more complex.

To address these issues, Sui introduces a standard way of describing an object for display. Instead of defining fields in the object struct, the display metadata is stored in a separate object, which is associated with the type. This way, the display metadata is not duplicated, and it is easy to extend and maintain.

Another important feature of Sui Display is the ability to define templates and use object fields in those templates. Not only it allows for a more flexible display, but it also frees the developer from the need to define the same fields with the same names and types in every object.

The Object Display is natively supported by the Sui Full Node , and the client can fetch the display metadata for any object if the object type has a Display associated with it.

```bash module book::arena;

use std::string::String; use sui::package; use sui::display;

/// The One Time Witness to claim the `Publisher` object. public struct ARENA has drop {}

/// Some object which will be displayed. public struct Hero has key { id: UID, class: String, level: u64, }

/// In the module initializer we create the `Publisher` object, and then /// the Display for the `Hero` type. fun init(otw: ARENA, ctx: &mut TxContext) { let publisher = package::claim(otw, ctx); let mut display = display::new(&publisher, ctx);

```
display.add(
    b"name".to_string(),
    b"{class} (lvl. {level})".to_string()
);

display.add(
    b"description".to_string(),
    b"One of the greatest heroes of all time. Join us!".to_string()
);

display.add(
    b"link".to_string(),
    b"https://example.com/hero/{id}".to_string()
);

display.add(
    b"image_url".to_string(),
    b"https://example.com/hero/{class}.jpg".to_string()
);

// Update the display with the new data.
// Must be called to apply changes.
display.update_version();

transfer::public_transfer(publisher, ctx.sender());
transfer::public_transfer(display, ctx.sender());
```

} ```

While the objects can be owned by accounts and may be a subject to True Ownership , the Display can be owned by the creator of the object. This way, the creator can update the display metadata and apply the changes globally without the need to update every object. The creator can also transfer Display to another account or even build an application around the object with custom functionality to manage the metadata.

The fields that are supported most widely are:

Please, refer to the Sui Documentation for the most up-to-date list of supported fields.
```

While there's a standard set of fields, the Display object does not enforce them. The developer can define any fields they need, and the client can use them as they see fit. Some applications may require additional fields, and omit other, and the Display is flexible enough to support them.

The Display object is defined in the sui::display module. It is a generic struct that takes a phantom type as a parameter. The phantom type is used to associate the Display object with the type it describes. The fields of the Display object are a VecMap of key-value pairs, where the key is the field name and the value is the field value. The version field is used to version the display metadata, and is updated on the update_display call.

File: sui-framework/sources/display.move

```
bash struct Display<phantom T: key> has key, store { id: UID, /// Contains fields for display.
Currently supported /// fields are: name, link, image and description. fields: VecMap<String,
String>, /// Version that can only be updated manually by the Publisher. version: u16 }
```

The Publisher object is required to a new Display, since it serves as the proof of ownership of type.

Currently, Display supports simple string interpolation and can use struct fields (and paths) in its templates. The syntax is trivial - {path} is replaced with the value of the field at the path. The path is a dot-separated list of field names, starting from the root object in case of nested fields.

```bash /// Some common metadata for objects. public struct Metadata has store { name: String, description: String, published_at: u64 }

/// The type with nested Metadata field. public struct LittlePony has key, store { id: UID, image_url: String, metadata: Metadata } ```

The Display for the type LittlePony above could be defined as follows:

```
bash { "name": "Just a pony", "image_url": "{image_url}", "description": "{metadata.description}" }
```

There's no restriction to how many Display objects can be created for a specific T . However, the most recently updated Display will be used by the full node.

## Object Display

To address these issues, Sui introduces a standard way of describing an object for display. Instead of defining fields in the object struct, the display metadata is stored in a separate object, which is associated with the type. This way, the display metadata is not duplicated, and it is easy to extend and maintain.

Another important feature of Sui Display is the ability to define templates and use object fields in those templates. Not only it allows for a more flexible display, but it also frees the developer from the need to define the same fields with the same names and types in every object.

The Object Display is natively supported by the Sui Full Node , and the client can fetch the display metadata for any object if the object type has a Display associated with it.

```bash module book::arena;

use std::string::String; use sui::package; use sui::display;

/// The One Time Witness to claim the `Publisher` object. public struct ARENA has drop {}

/// Some object which will be displayed. public struct Hero has key { id: UID, class: String, level: u64, }

/// In the module initializer we create the `Publisher` object, and then /// the Display for the `Hero` type. fun init(otw: ARENA, ctx: &mut TxContext) { let publisher = package::claim(otw, ctx); let mut display = display::new(&publisher, ctx);

display.add(
    b"name".to_string(),
    b"{class} (lvl. {level})".to_string()
);

display.add(
    b"description".to_string(),
    b"One of the greatest heroes of all time. Join us!".to_string()
);
```
```

```
display.add(
    b"link".to_string(),
    b"https://example.com/hero/{id}".to_string()
);

display.add(
    b"image_url".to_string(),
    b"https://example.com/hero/{class}.jpg".to_string()
);

// Update the display with the new data.
// Must be called to apply changes.
display.update_version();

transfer::public_transfer(publisher, ctx.sender());
transfer::public_transfer(display, ctx.sender());

} ```
```

While the objects can be owned by accounts and may be a subject to True Ownership , the Display can be owned by the creator of the object. This way, the creator can update the display metadata and apply the changes globally without the need to update every object. The creator can also transfer Display to another account or even build an application around the object with custom functionality to manage the metadata.

The fields that are supported most widely are:

Please, refer to the Sui Documentation for the most up-to-date list of supported fields.

While there's a standard set of fields, the Display object does not enforce them. The developer can define any fields they need, and the client can use them as they see fit. Some applications may require additional fields, and omit other, and the Display is flexible enough to support them.

The Display object is defined in the sui::display module. It is a generic struct that takes a phantom type as a parameter. The phantom type is used to associate the Display object with the type it describes. The fields of the Display object are a VecMap of key-value pairs, where the key is the field name and the value is the field value. The version field is used to version the display metadata, and is updated on the update_display call.

File: sui-framework/sources/display.move

```
bash struct Display<phantom T: key> has key, store { id: UID, /// Contains fields for display.
Currently supported /// fields are: name, link, image and description. fields: VecMap<String,
String>, /// Version that can only be updated manually by the Publisher. version: u16 }
```

The Publisher object is required to a new Display, since it serves as the proof of ownership of type.

Currently, Display supports simple string interpolation and can use struct fields (and paths) in its templates. The syntax is trivial - {path} is replaced with the value of the field at the path. The path is a dot-separated list of field names, starting from the root object in case of nested fields.

```bash /// Some common metadata for objects. public struct Metadata has store { name: String, description: String, published_at: u64 }

/// The type with nested Metadata field. public struct LittlePony has key, store { id: UID, image_url: String, metadata: Metadata } ```

The Display for the type LittlePony above could be defined as follows:

```
bash { "name": "Just a pony", "image_url": "{image_url}", "description": "{metadata.description}" }
```

There's no restriction to how many Display objects can be created for a specific T . However, the most recently updated Display will be used by the full node.

## Creator Privilege

While the objects can be owned by accounts and may be a subject to True Ownership , the Display can be owned by the creator of the object. This way, the creator can update the display metadata and apply the changes globally without the need to update every object. The creator can also transfer Display to another account or even build an application around the object with custom functionality to manage the metadata.
```

The fields that are supported most widely are:

Please, refer to the Sui Documentation for the most up-to-date list of supported fields.

While there's a standard set of fields, the Display object does not enforce them. The developer can define any fields they need, and the client can use them as they see fit. Some applications may require additional fields, and omit other, and the Display is flexible enough to support them.

The Display object is defined in the sui::display module. It is a generic struct that takes a phantom type as a parameter. The phantom type is used to associate the Display object with the type it describes. The fields of the Display object are a VecMap of key-value pairs, where the key is the field name and the value is the field value. The version field is used to version the display metadata, and is updated on the update_display call.

File: sui-framework/sources/display.move

```
bash struct Display<phantom T: key> has key, store { id: UID, /// Contains fields for display.
Currently supported /// fields are: name, link, image and description. fields: VecMap<String,
String>, /// Version that can only be updated manually by the Publisher. version: u16 }
```

The Publisher object is required to a new Display, since it serves as the proof of ownership of type.

Currently, Display supports simple string interpolation and can use struct fields (and paths) in its templates. The syntax is trivial - {path} is replaced with the value of the field at the path. The path is a dot-separated list of field names, starting from the root object in case of nested fields.

```bash /// Some common metadata for objects. public struct Metadata has store { name: String, description: String, published_at: u64 }

/// The type with nested Metadata field. public struct LittlePony has key, store { id: UID, image_url: String, metadata: Metadata } ```

The Display for the type LittlePony above could be defined as follows:

```
bash { "name": "Just a pony", "image_url": "{image_url}", "description": "{metadata.description}" }
```

There's no restriction to how many Display objects can be created for a specific T . However, the most recently updated Display will be used by the full node.

## Standard Fields

The fields that are supported most widely are:

Please, refer to the Sui Documentation for the most up-to-date list of supported fields.

While there's a standard set of fields, the Display object does not enforce them. The developer can define any fields they need, and the client can use them as they see fit. Some applications may require additional fields, and omit other, and the Display is flexible enough to support them.

The Display object is defined in the sui::display module. It is a generic struct that takes a phantom type as a parameter. The phantom type is used to associate the Display object with the type it describes. The fields of the Display object are a VecMap of key-value pairs, where the key is the field name and the value is the field value. The version field is used to version the display metadata, and is updated on the update_display call.

File: sui-framework/sources/display.move

```
bash struct Display<phantom T: key> has key, store { id: UID, /// Contains fields for display.
Currently supported /// fields are: name, link, image and description. fields: VecMap<String,
String>, /// Version that can only be updated manually by the Publisher. version: u16 }
```

The Publisher object is required to a new Display, since it serves as the proof of ownership of type.

Currently, Display supports simple string interpolation and can use struct fields (and paths) in its templates. The syntax is trivial - {path} is replaced with the value of the field at the path. The path is a dot-separated list of field names, starting from the root object in case of nested fields.

```bash /// Some common metadata for objects. public struct Metadata has store { name: String, description: String, published_at: u64 }

/// The type with nested Metadata field. public struct LittlePony has key, store { id: UID, image_url: String, metadata: Metadata } ```

The Display for the type LittlePony above could be defined as follows:

```
bash { "name": "Just a pony", "image_url": "{image_url}", "description": "{metadata.description}" }
```

There's no restriction to how many Display objects can be created for a specific T . However, the most recently updated Display will be used by the full node.

Working with Display

The Display object is defined in the sui::display module. It is a generic struct that takes a phantom type as a parameter. The phantom type is used to associate the Display object with the type it describes. The fields of the Display object are a VecMap of key-value pairs, where the key is the field name and the value is the field value. The version field is used to version the display metadata, and is updated on the update_display call.

File: sui-framework/sources/display.move

```
bash struct Display<phantom T: key> has key, store { id: UID, /// Contains fields for display.
Currently supported /// fields are: name, link, image and description. fields: VecMap<String,
String>, /// Version that can only be updated manually by the Publisher. version: u16 }
```

The Publisher object is required to a new Display, since it serves as the proof of ownership of type.

Currently, Display supports simple string interpolation and can use struct fields (and paths) in its templates. The syntax is trivial - {path} is replaced with the value of the field at the path. The path is a dot-separated list of field names, starting from the root object in case of nested fields.

```bash /// Some common metadata for objects. public struct Metadata has store { name: String, description: String, published_at: u64 }

/// The type with nested Metadata field. public struct LittlePony has key, store { id: UID, image_url: String, metadata: Metadata } ```

The Display for the type LittlePony above could be defined as follows:

```
bash { "name": "Just a pony", "image_url": "{image_url}", "description": "{metadata.description}" }
```

There's no restriction to how many Display objects can be created for a specific T . However, the most recently updated Display will be used by the full node.

## Template Syntax

Currently, Display supports simple string interpolation and can use struct fields (and paths) in its templates. The syntax is trivial - {path} is replaced with the value of the field at the path. The path is a dot-separated list of field names, starting from the root object in case of nested fields.

```bash /// Some common metadata for objects. public struct Metadata has store { name: String, description: String, published_at: u64 }

/// The type with nested Metadata field. public struct LittlePony has key, store { id: UID, image_url: String, metadata: Metadata } ```

The Display for the type LittlePony above could be defined as follows:

```
bash { "name": "Just a pony", "image_url": "{image_url}", "description": "{metadata.description}" }
```

There's no restriction to how many Display objects can be created for a specific T . However, the most recently updated Display will be used by the full node.

Multiple Display Objects

There's no restriction to how many Display objects can be created for a specific T . However, the most recently updated Display will be used by the full node.

Further Reading