Kiosk Apps

Kiosk apps are a way to extend the functionality of Sui Kiosk while keeping the core functionality intact. You can develop apps to add new features to a kiosk without having to modify the core code or move the assets elsewhere.

There are two types of apps:

Basic Kiosk apps do not require Kiosk Apps API to function. They usually serve the purpose of adding custom metadata to a kiosk or wrapping/working with existing objects such as Kiosk or KioskOwnerCap . An example of an app that does not require the API is the Personal Kiosk app.

Kiosk has an id: UID field like all objects on Sui, which allows this object to be uniquely identified and carry custom dynamic fields and dynamic object fields. The Kiosk itself is built around dynamic fields and features like place and list are built around dynamic object fields.

Kiosk can carry additional dynamic fields and dynamic object fields. The uid_mut_as_owner function allows the Kiosk owner to mutably access the UID of the Kiosk object and use it to add or remove custom fields.

Function signature:

kiosk::uid_mut_as_owner(self: &mut Kiosk, cap: &KioskOwnerCap): &mut UID

Anyone can read the uid of kiosks. This allows third party modules to read the fields of the kiosk if they're allowed to do so. Therefore enabling the object capability and other patterns.

You can attach custom dynamic fields to your kiosks that anyone can then read (but only you can modify), you can use this to implement basic apps. For example, a Kiosk Name app where you as the kiosk owner can set a name for the kiosk, attach it as a dynamic field, and make it readable by anyone.

Permissioned apps use the Kiosk Apps API to perform actions in the kiosk. They usually imply interaction with a third party and provide guarantees for the storage access (preventing malicious actions from the seller).

Just having access to the uid is often not enough to build an app due to the security limitations. Only the owner of a kiosk has full access to the uid , which means that an app involving a third party would require involvement from the kiosk owner in every step of the process.

In addition to limited and constrained access to storage, app permissions are also owner dependent. In the default setup, no party can place or lock items in a kiosk without its owner's consent. As a result, some cases such as collection bidding (offering X SUI for any item in a collection) requires the kiosk owner to approve the bid.

The kiosk_extension module addresses concerns over owner bottlenecks and provides more guarantees for storage access. The module provides a set of functions that enable you to perform certain actions in the kiosk without the kiosk owner's involvement and have a guarantee that the storage of the app is not tampered with.

These are the key points in the lifecycle of a Sui Kiosk app:

For the app to function, the kiosk owner first needs to install it. To achieve that, an app needs to implement the add function that the kiosk owner calls to request all necessary permissions.

The signature of the kiosk_extension::add function requires the app witness, making it impossible to install an app without an explicit implementation. The following example shows how to implement the add function for an app that requires the place permission:

Apps can request permissions from the kiosk owner on installation. Permissions follow the all or nothing principle. If the kiosk owner adds an app, it gets all of the requested permissions; if the kiosk owner then disables an app, it loses all of its permissions.

Permissions are represented as a u128 integer storing a bitmap. Each of the bits corresponds to a permission, the first bit is the least significant bit. The following table lists all permissions and their corresponding bit:

Currently, Sui Kiosk has only two permissions: place (first bit) and lock and place (second bit). The remaining bits are reserved for future use.

It's considered good practice to define a constant containing permissions of the app:

If an app requests and is granted permissions (and isn't disabled), it can access protected functions. The following example shows

how to access the place function:

Currently, two functions are available:

Use the can_place(kiosk: &Kiosk): bool function to check if the app has the place permission. Similarly, you can use the can_lock(kiosk: &Kiosk): bool function to check if the app has the lock permission. Both functions make sure that the app is enabled, so you don't need to explicitly check for that.

Every app gets its isolated storage as a bag type that only the app module can access (providing the app witness). See The Move Book to learn more about dynamic collections, like bags, available in Move. After you install an app, it can use the storage to store its data. Ideally, the storage should be managed in a way that allows the app to be removed from the kiosk if there are no active trades or other activities happening at the moment.

The storage is always available to the app if it is installed. The owner of a kiosk can't access the storage of the app if the logic for it is not implemented.

An installed app can access the storage mutably or immutably using one of the following functions:

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

Basic apps

Basic Kiosk apps do not require Kiosk Apps API to function. They usually serve the purpose of adding custom metadata to a kiosk or wrapping/working with existing objects such as Kiosk or KioskOwnerCap . An example of an app that does not require the API is the Personal Kiosk app.

Kiosk has an id: UID field like all objects on Sui, which allows this object to be uniquely identified and carry custom dynamic fields and dynamic object fields. The Kiosk itself is built around dynamic fields and features like place and list are built around dynamic object fields.

Kiosk can carry additional dynamic fields and dynamic object fields. The uid_mut_as_owner function allows the Kiosk owner to mutably access the UID of the Kiosk object and use it to add or remove custom fields.

Function signature:

kiosk::uid_mut_as_owner(self: &mut Kiosk, cap: &KioskOwnerCap): &mut UID

Anyone can read the uid of kiosks. This allows third party modules to read the fields of the kiosk if they're allowed to do so. Therefore enabling the object capability and other patterns.

You can attach custom dynamic fields to your kiosks that anyone can then read (but only you can modify), you can use this to implement basic apps. For example, a Kiosk Name app where you as the kiosk owner can set a name for the kiosk, attach it as a dynamic field, and make it readable by anyone.

Permissioned apps use the Kiosk Apps API to perform actions in the kiosk. They usually imply interaction with a third party and provide guarantees for the storage access (preventing malicious actions from the seller).

Just having access to the uid is often not enough to build an app due to the security limitations. Only the owner of a kiosk has full access to the uid , which means that an app involving a third party would require involvement from the kiosk owner in every step of the process.

In addition to limited and constrained access to storage, app permissions are also owner dependent. In the default setup, no party can place or lock items in a kiosk without its owner's consent. As a result, some cases such as collection bidding (offering X SUI for any item in a collection) requires the kiosk owner to approve the bid.

The kiosk_extension module addresses concerns over owner bottlenecks and provides more guarantees for storage access. The module provides a set of functions that enable you to perform certain actions in the kiosk without the kiosk owner's involvement and have a guarantee that the storage of the app is not tampered with.

These are the key points in the lifecycle of a Sui Kiosk app:

For the app to function, the kiosk owner first needs to install it. To achieve that, an app needs to implement the add function that the kiosk owner calls to request all necessary permissions.

The signature of the kiosk_extension::add function requires the app witness, making it impossible to install an app without an explicit implementation. The following example shows how to implement the add function for an app that requires the place permission:

Apps can request permissions from the kiosk owner on installation. Permissions follow the all or nothing principle. If the kiosk owner adds an app, it gets all of the requested permissions; if the kiosk owner then disables an app, it loses all of its permissions.

Permissions are represented as a u128 integer storing a bitmap. Each of the bits corresponds to a permission, the first bit is the least significant bit. The following table lists all permissions and their corresponding bit:

Currently, Sui Kiosk has only two permissions: place (first bit) and lock and place (second bit). The remaining bits are reserved for future use.

It's considered good practice to define a constant containing permissions of the app:

If an app requests and is granted permissions (and isn't disabled), it can access protected functions. The following example shows how to access the place function:

Currently, two functions are available:

Use the can_place(kiosk: &Kiosk): bool function to check if the app has the place permission. Similarly, you can use the can_lock(kiosk: &Kiosk): bool function to check if the app has the lock permission. Both functions make sure that the app is enabled, so you don't need to explicitly check for that.

Every app gets its isolated storage as a bag type that only the app module can access (providing the app witness). See The Move Book to learn more about dynamic collections, like bags, available in Move. After you install an app, it can use the storage to store its data. Ideally, the storage should be managed in a way that allows the app to be removed from the kiosk if there are no active trades or other activities happening at the moment.

The storage is always available to the app if it is installed. The owner of a kiosk can't access the storage of the app if the logic for it is not implemented.

An installed app can access the storage mutably or immutably using one of the following functions:

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

Permissioned apps using the Kiosk Apps API

Permissioned apps use the Kiosk Apps API to perform actions in the kiosk. They usually imply interaction with a third party and provide guarantees for the storage access (preventing malicious actions from the seller).

Just having access to the uid is often not enough to build an app due to the security limitations. Only the owner of a kiosk has full access to the uid , which means that an app involving a third party would require involvement from the kiosk owner in every step of the process.

In addition to limited and constrained access to storage, app permissions are also owner dependent. In the default setup, no party can place or lock items in a kiosk without its owner's consent. As a result, some cases such as collection bidding (offering X SUI for any item in a collection) requires the kiosk owner to approve the bid.

The kiosk_extension module addresses concerns over owner bottlenecks and provides more guarantees for storage access. The module provides a set of functions that enable you to perform certain actions in the kiosk without the kiosk owner's involvement and have a guarantee that the storage of the app is not tampered with.

These are the key points in the lifecycle of a Sui Kiosk app:

For the app to function, the kiosk owner first needs to install it. To achieve that, an app needs to implement the add function that the kiosk owner calls to request all necessary permissions.

The signature of the kiosk_extension::add function requires the app witness, making it impossible to install an app without an explicit implementation. The following example shows how to implement the add function for an app that requires the place permission:

Apps can request permissions from the kiosk owner on installation. Permissions follow the all or nothing principle. If the kiosk owner adds an app, it gets all of the requested permissions; if the kiosk owner then disables an app, it loses all of its permissions.

Permissions are represented as a u128 integer storing a bitmap. Each of the bits corresponds to a permission, the first bit is the least significant bit. The following table lists all permissions and their corresponding bit:

Currently, Sui Kiosk has only two permissions: place (first bit) and lock and place (second bit). The remaining bits are reserved for future use.

It's considered good practice to define a constant containing permissions of the app:

If an app requests and is granted permissions (and isn't disabled), it can access protected functions. The following example shows how to access the place function:

Currently, two functions are available:

Use the can_place(kiosk: &Kiosk): bool function to check if the app has the place permission. Similarly, you can use the can_lock(kiosk: &Kiosk): bool function to check if the app has the lock permission. Both functions make sure that the app is enabled, so you don't need to explicitly check for that.

Every app gets its isolated storage as a bag type that only the app module can access (providing the app witness). See The Move Book to learn more about dynamic collections, like bags, available in Move. After you install an app, it can use the storage to store its data. Ideally, the storage should be managed in a way that allows the app to be removed from the kiosk if there are no active trades or other activities happening at the moment.

The storage is always available to the app if it is installed. The owner of a kiosk can't access the storage of the app if the logic for it is not implemented.

An installed app can access the storage mutably or immutably using one of the following functions:

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

kiosk_extension module

The kiosk_extension module addresses concerns over owner bottlenecks and provides more guarantees for storage access. The module provides a set of functions that enable you to perform certain actions in the kiosk without the kiosk owner's involvement and have a guarantee that the storage of the app is not tampered with.

These are the key points in the lifecycle of a Sui Kiosk app:

For the app to function, the kiosk owner first needs to install it. To achieve that, an app needs to implement the add function that the kiosk owner calls to request all necessary permissions.

The signature of the kiosk_extension::add function requires the app witness, making it impossible to install an app without an explicit implementation. The following example shows how to implement the add function for an app that requires the place permission:

Apps can request permissions from the kiosk owner on installation. Permissions follow the all or nothing principle. If the kiosk owner adds an app, it gets all of the requested permissions; if the kiosk owner then disables an app, it loses all of its permissions.

Permissions are represented as a u128 integer storing a bitmap. Each of the bits corresponds to a permission, the first bit is the least significant bit. The following table lists all permissions and their corresponding bit:

Currently, Sui Kiosk has only two permissions: place (first bit) and lock and place (second bit). The remaining bits are reserved for future use.

It's considered good practice to define a constant containing permissions of the app:

If an app requests and is granted permissions (and isn't disabled), it can access protected functions. The following example shows how to access the place function:

Currently, two functions are available:

Use the can_place(kiosk: &Kiosk): bool function to check if the app has the place permission. Similarly, you can use the can_lock(kiosk: &Kiosk): bool function to check if the app has the lock permission. Both functions make sure that the app is enabled, so you don't need to explicitly check for that.

Every app gets its isolated storage as a bag type that only the app module can access (providing the app witness). See The Move Book to learn more about dynamic collections, like bags, available in Move. After you install an app, it can use the storage to store its data. Ideally, the storage should be managed in a way that allows the app to be removed from the kiosk if there are no active trades or other activities happening at the moment.

The storage is always available to the app if it is installed. The owner of a kiosk can't access the storage of the app if the logic for it is not implemented.

An installed app can access the storage mutably or immutably using one of the following functions:

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and

prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

App lifecycle

These are the key points in the lifecycle of a Sui Kiosk app:

For the app to function, the kiosk owner first needs to install it. To achieve that, an app needs to implement the add function that the kiosk owner calls to request all necessary permissions.

The signature of the kiosk_extension::add function requires the app witness, making it impossible to install an app without an explicit implementation. The following example shows how to implement the add function for an app that requires the place permission:

Apps can request permissions from the kiosk owner on installation. Permissions follow the all or nothing principle. If the kiosk owner adds an app, it gets all of the requested permissions; if the kiosk owner then disables an app, it loses all of its permissions.

Permissions are represented as a u128 integer storing a bitmap. Each of the bits corresponds to a permission, the first bit is the least significant bit. The following table lists all permissions and their corresponding bit:

Currently, Sui Kiosk has only two permissions: place (first bit) and lock and place (second bit). The remaining bits are reserved for future use.

It's considered good practice to define a constant containing permissions of the app:

If an app requests and is granted permissions (and isn't disabled), it can access protected functions. The following example shows how to access the place function:

Currently, two functions are available:

Use the can_place(kiosk: &Kiosk): bool function to check if the app has the place permission. Similarly, you can use the can_lock(kiosk: &Kiosk): bool function to check if the app has the lock permission. Both functions make sure that the app is enabled, so you don't need to explicitly check for that.

Every app gets its isolated storage as a bag type that only the app module can access (providing the app witness). See The Move Book to learn more about dynamic collections, like bags, available in Move. After you install an app, it can use the storage to store its data. Ideally, the storage should be managed in a way that allows the app to be removed from the kiosk if there are no active trades or other activities happening at the moment.

The storage is always available to the app if it is installed. The owner of a kiosk can't access the storage of the app if the logic for it is not implemented.

An installed app can access the storage mutably or immutably using one of the following functions:

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the

kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

Adding an app

For the app to function, the kiosk owner first needs to install it. To achieve that, an app needs to implement the add function that the kiosk owner calls to request all necessary permissions.

The signature of the kiosk_extension::add function requires the app witness, making it impossible to install an app without an explicit implementation. The following example shows how to implement the add function for an app that requires the place permission:

Apps can request permissions from the kiosk owner on installation. Permissions follow the all or nothing principle. If the kiosk owner adds an app, it gets all of the requested permissions; if the kiosk owner then disables an app, it loses all of its permissions.

Permissions are represented as a u128 integer storing a bitmap. Each of the bits corresponds to a permission, the first bit is the least significant bit. The following table lists all permissions and their corresponding bit:

Currently, Sui Kiosk has only two permissions: place (first bit) and lock and place (second bit). The remaining bits are reserved for future use.

It's considered good practice to define a constant containing permissions of the app:

If an app requests and is granted permissions (and isn't disabled), it can access protected functions. The following example shows how to access the place function:

Currently, two functions are available:

Use the can_place(kiosk: &Kiosk): bool function to check if the app has the place permission. Similarly, you can use the can_lock(kiosk: &Kiosk): bool function to check if the app has the lock permission. Both functions make sure that the app is enabled, so you don't need to explicitly check for that.

Every app gets its isolated storage as a bag type that only the app module can access (providing the app witness). See The Move Book to learn more about dynamic collections, like bags, available in Move. After you install an app, it can use the storage to store its data. Ideally, the storage should be managed in a way that allows the app to be removed from the kiosk if there are no active trades or other activities happening at the moment.

The storage is always available to the app if it is installed. The owner of a kiosk can't access the storage of the app if the logic for it is not implemented.

An installed app can access the storage mutably or immutably using one of the following functions:

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

App permissions

Apps can request permissions from the kiosk owner on installation. Permissions follow the all or nothing principle. If the kiosk owner adds an app, it gets all of the requested permissions; if the kiosk owner then disables an app, it loses all of its permissions.

Permissions are represented as a u128 integer storing a bitmap. Each of the bits corresponds to a permission, the first bit is the least significant bit. The following table lists all permissions and their corresponding bit:

Currently, Sui Kiosk has only two permissions: place (first bit) and lock and place (second bit). The remaining bits are reserved for future use.

It's considered good practice to define a constant containing permissions of the app:

If an app requests and is granted permissions (and isn't disabled), it can access protected functions. The following example shows how to access the place function:

Currently, two functions are available:

Use the can_place(kiosk: &Kiosk): bool function to check if the app has the place permission. Similarly, you can use the can_lock(kiosk: &Kiosk): bool function to check if the app has the lock permission. Both functions make sure that the app is enabled, so you don't need to explicitly check for that.

Every app gets its isolated storage as a bag type that only the app module can access (providing the app witness). See The Move Book to learn more about dynamic collections, like bags, available in Move. After you install an app, it can use the storage to store its data. Ideally, the storage should be managed in a way that allows the app to be removed from the kiosk if there are no active trades or other activities happening at the moment.

The storage is always available to the app if it is installed. The owner of a kiosk can't access the storage of the app if the logic for it is not implemented.

An installed app can access the storage mutably or immutably using one of the following functions:

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

App storage

Every app gets its isolated storage as a bag type that only the app module can access (providing the app witness). See The Move Book to learn more about dynamic collections, like bags, available in Move. After you install an app, it can use the storage to store its data. Ideally, the storage should be managed in a way that allows the app to be removed from the kiosk if there are no active trades or other activities happening at the moment.

The storage is always available to the app if it is installed. The owner of a kiosk can't access the storage of the app if the logic for it is not implemented.

An installed app can access the storage mutably or immutably using one of the following functions:

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

Disabling and removing

The kiosk owner can disable any app at any time. Doing so revokes all permissions of the app and prevents it from performing any actions in the kiosk. The kiosk owner can also re-enable the app at any time.

Disabling an app does not remove it from the kiosk. An installed app has access to its storage until completely removed from the kiosk.

Use the disable(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to disable an app. It revokes all permissions of the app and prevents it from performing any protected actions in the kiosk.

Example PTB

You can remove an app only if the storage is empty. Use the remove(kiosk: &mut Kiosk, cap: &KioskOwnerCap) function to facilitate removal. The function removes the app, unpacks the app storage and configuration and rebates the storage cost to the kiosk owner. Only the kiosk owner can perform this action.

The call fails if the storage is not empty.

Example PTB

Related links