


Filter by keyword 

# Getting and Setting Data

## ON THIS PAGE

[Overview](#)[Scopes](#)[Visibilities](#)[Size Limit](#)[REST API Access](#)[t.get\(\)](#)[...](#)

The Power-Up platform provides a number of methods that allow Power-Ups to store custom settings and other data in Trello. There are a number of use-cases for storing data in a Power-Up and we've tried to provide for each by allowing you to manage the data that is stored (key/value pairs), where it is stored (scopes), and who has access to it (visibility).

These methods rely on the [Power-Up Client Library](#) and refer to it as though it is currently available via a variable named `t`.

## Overview

`t.get()` and `t.set()` ([documentation](#)) are the two main methods for getting and setting data, respectively. Each takes some combination of scopes and visibilities.

Trello uses key/value pairs, in addition to the scopes and visibilities, to store data. The method signatures for `t.set(scope, visibility, key, value)` and `t.get(scope, visibility, key, default)` mirror each other because they should be used in tandem. Trello also provides the ability to access stored data in bulk via `t.getAll()` ([documentation](#))

## Scopes

Scopes represent the Trello object that you want to store the data against. Valid scopes include: `board`, `card`, `member`, `organization`, or the ID of a card on the current board.

You can think of the scope as the location of where the data will be stored. For instance, imagine your Power-Up allows users to rank cards via a card "rating." Because you want to track a single rating for each card on a board, you would use the `card` scope when calling `t.set()` to set the rating value on the card.

When using a named scope, Trello will store the data on the particular object of that name if it is considered in context. So for example `t.set('member', ...)` will set data against the Trello member who is currently interacting with your Power-Up. Whereas, when using `t.set('card', ...)` if there isn't currently a card open, Trello doesn't know which card is in context and the call will fail.

Another example, when using data methods inside the `board-buttons` capability callback there will be no card in scope; whereas inside the `card-buttons` or `attachment-sections` capability callbacks there will be a card in scope.

You can tell whether a named object is currently in scope or not by looking at the current context of your `t` object. You can get the current context with `t.getContext()`. You can check that `context.card` or `context.member` is present and an ID to know that the named object is in scope.

## Visibilities

You can control who is able to read the data you set with visibilities.

There are two options for visibility: `shared` and `private`.

A visibility of `shared` means that any Trello member who can see the object it is stored against (determined by Scope) can see the data.

A visibility of `private` means that only the member who set the data will be able to see it, regardless of what scope it is stored at.

For instance, using the card rating example from above, if you wanted every user to be able to see every other user's ratings, you would use `shared` visibility. Whereas, if you have a secret token or key that belongs to a single user (like a user token from Trello's RESTful API!) you should store it at a `private` visibility such that only the user it belongs to has access to it.

## Size Limit

When storing data at a given scope and visibility, the data is serialized to a string and stored. The size limit on the resulting stringified object is 4096

characters per scope/visibility pair. This means that you could store 4096 characters at the `shared` and `card` scope/visibility pair and an addition 4096 characters at the `private` and `card` scope/visibility pair.

At the point in time you try to exceed this limit via a `t.set()` call, Trello will reject the Promise with the following error message: `PluginData length of 4096 characters exceeded`. See:

<https://developer.atlassian.com/cloud/trello/power-ups/client-library/getting-and-setting-data/>

## REST API Access

Data stored via `t.set` is only available via GET requests on the objects to whose context the data has been stored. For instance, to access the data stored within the context of a card you can make a GET request to [/cards/{id}/pluginData](#).

Currently, PUT, POST, and DELETE methods are not supported for managing `pluginData`.

## t.get()

`t.get(scope, visibility, key, default)`

Get data at a specific scope and visibility

```
1 var t = window.TrelloPowerUp.iframe();
2
3 return t.get('board', 'shared', 'myKey')
4 .then(function (data) {
5   console.log(JSON.stringify(data, null, 2));
6 });
```

You can optionally include a default value to return if that key doesn't exist

```
1 var t = window.TrelloPowerUp.iframe();
2
3 return t.get('board', 'shared', 'myKey', 'Uh oh, not yet set')
4 .then(function (data) {
5   console.log(JSON.stringify(data, null, 2));
6 });
```

Alternatively, if you want all data at a scope and visibility, not just a single key:

```
1 var t = window.TrelloPowerUp.iframe();
2
3 return t.get('board', 'shared')
```

```
4 .then(function (data) {  
5   console.log(JSON.stringify(data, null, 2));  
6 });
```

You can also get data from a card given it's ID:

```
1 var t = window.TrelloPowerUp.iframe();  
2  
3 return t.get('542b0bd40d309dc6eba7ec91', 'shared', 'myKey')  
4 .then(function (data) {  
5   console.log(JSON.stringify(data, null, 2));  
6 });
```

## t.getAll()

Return all pluginData for all scopes & visibilities currently in context

```
1 var t = window.TrelloPowerUp.iframe();  
2  
3 return t.getAll()  
4 .then(function (data) {  
5   console.log(JSON.stringify(data, null, 2));  
6 });
```

The data response might look like:

```
1 {  
2   board: {  
3     shared: {  
4       enabled: true  
5     },  
6     private: {  
7       shh: 'its a secret'  
8     }  
9   },  
10  organization: {  
11    shared: {  
12      interval: 1500  
13    }  
14  }  
15 }
```

## t.set()

t.set(scope, visibility, key, value)

Used to store data within the Trello Power-Up platform.

`t.set()` returns a Promise and accepts the following options:

Key	Values
<b>scope</b> string	One of: <code>board</code> , <code>card</code> , <code>member</code> , <code>organization</code> , or the ID of card that is in scope.
<b>visibility</b> string	One of: <code>shared</code> , <code>private</code> .
<b>key</b> string	Any string.
<b>value</b> Serializable item	Accepts individual string, number, boolean, or object containing them.



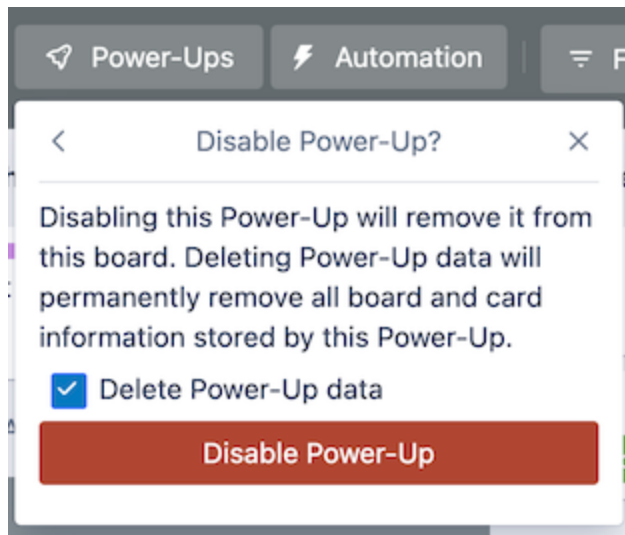
### Visibility: "shared" vs. "private"

When selecting a visibility for the data that you are setting, you should be thinking about who should be able to see it.

**shared** - You should use this visibility if the data you are storing should be visible / accessible to everyone who can read the "scope" it is stored against. So for example `t.set('board', 'shared', 'key', 'value')` would make that data (`{key: 'value'}`) visible to everyone who can see the board. Keep in mind, a board could be made public, meaning *everyone* including people who are not even Trello members would then have access to that data.

*Do **not** store secrets or personal data in 'shared' visibility*

Note that board and workspace admins that disable Power-Ups from their board(s) may choose to clear the board-level storage for that Power-Up as a part of the disable action. This action clears all board-scoped and card-scoped Power-Up data that has a visibility of `shared`.



**private** - You should use this visibility when you are storing data that you want to be visible / accessible only to the Trello member who you are storing it against. So for example `t.set('board', 'private', 'key', 'value')` would make that data (`{key: 'value'}`) only accessible to that particular member. If another member were to load the board and the data stored against it, they would not be able to see or access that piece of data.

If you are storing external service tokens, such as an oauth token to your app or service, you should consider using `t.storeSecret`

## Errors

Invalid value for scope. - Occurs when an invalid value is passed for the scope.

Invalid value for visibility. - Occurs when an invalid value is passed for the visibility.

PluginData length of 4096 characters exceeded. See: <https://developer.atlassian.com/cloud/trello/power-ups/client-library/getting-and-setting-data/> - Occurs when trying to set data that exceeds 4096 characters in current scope/visibility pair.

Detected potential secret. You should never store secrets like tokens in shared pluginData. See: <https://developer.atlassian.com/cloud/trello/power-ups/client-library/getting-and-setting-data/> - We want to make it harder to accidentally store secrets in a place that might be publicly accessible so we will block calls to `t.set('{object}', 'shared', {key}, {value})` where the key contains the data that looks like it might be a secret: `auth`, `refresh`, `token`, `secret`.

## Setting Multiple Values

`t.set()` allows for multiple keys to be set by accepting an object full of the key/value pairs instead of the single `key` and `value` parameter:

```
1 let myKeyValueObject = {
2   myKey: 'myValue',
3   otherKey: 25,
4   enabled: true
5 }
6 t.set('card', 'shared', myKeyValueObject);
```

Shared storage operations are not atomic. If two `set` operations happen at the same time on two different keys they can override one another.

At the point in time a Power-Up uses `t.set()`, a single PUT request is made for all of the shared storage at once. Under the hood, Trello doesn't store separate key/value pairs—it is a single JSON blob. If client A changes something and client B sets something before getting the socket update for A's change, they're going to collide and result in data loss.

If a Power-Up needs to set multiple keys concurrently, the safest thing to do is to pass an object containing the multiple key/value pairs as described above.

## Example Usage

You can set a specific key and value at the target visibility and scope:

```
1 var t = window.TrelloPowerUp.iframe();
2
3 return t.set('card', 'shared', 'myKey', 'myValue');
```

Or if you have a plain javascript object, you can set that all at once:

```
1 var t = window.TrelloPowerUp.iframe();
2
3 return t.set('card', 'shared', { myKey: 'myValue', more: 25, enable
```

Additionally, you can set data on a card from any context by calling `t.set` with its ID.

```
1 var t = window.TrelloPowerUp.iframe();
2
3 return t.set('542b0bd40d309dc6eba7ec91', 'shared', 'myKey', 'myValu
```

You may want to check what kind of permissions this user has before using `t.set`:

```

1  var t = window.TrelloPowerUp.iframe();
2
3  var permissions = t.getContext().permissions;
4
5  if (permissions.board === 'write') {
6      return t.set('board', 'shared', { myKey: 'myValue', more: 25, ena
7  }

```



## Write Permissions

Keep in mind, the user of your Power-Up isn't guaranteed to have write access in Trello, or on a specific object. This means your `t.set` call could fail. You may want to catch these errors, and have fallback logic.

You can check the permissions object in the context as shown above to know whether the active user can set "shared" data at a certain scope. In general, a Trello member, (as long as there is a member in the context), can set "private" data at any scope, with the exception of "organization" when the organization in question is itself private, and the active member is not a member of that organization.

In that particular case, you can catch the error, and fallback to writing that at the board level.

## t.remove()

`t.remove(scope, visibility, key)`

You can remove a key at a specific scope:

```

1  var t = window.TrelloPowerUp.iframe();
2
3  return t.remove('member', 'private', 'myKey');

```

Or you can remove multiple keys at once:

```

1  var t = window.TrelloPowerUp.iframe();
2
3  return t.remove('member', 'private', ['myKey', 'anotherKey', 'enabl

```

Rate this page: ☆ ☆ ☆ ☆ ☆