Save Data

Namazu Elements Core Features · https://namazustudios.com/docs/namazu-elements-core/save-data/

Elements allows saving data associated with Users and Profiles with multiple options to prevent conflicts using version control.

Save data allows for the storage of arbitrary data associated either with a User or a Profile. This data may be any arbitrary string. There is no particular size limit, however, save documents exceeding a few kilobytes should be discouraged.

Elements organizes each Save Data Document in a slot identified by an integer, starting at zero. The number of slots per User or Profile is the maximum value of a signed 32-bit integer.

Each Save Data Document has the following properties:

- **id** - The integer-based slot identifier for the save data document
- **user** - The User that owns this Save Data, will always be set even if the Save Data Document is Profile scoped.
- **profile** - The Profile which owns the Save Data document. If User scoped, then this field will be null.
- **timestamp** - Represents the last update time of the document.
- **version** - Indicates the last incremental revision of the contents of the save file. This is a hash or checksum of the Contents and must match when updating the data in the slot. This allows client code to detect and correct consistency for the slot.
- **contents** - A string representing the contents of the save data file.

Creating Save Data

When creating a new Save Data document, you must specify an empty slot, userId, and profileId. Elements will reject the request if the save data is already present in the supplied slot. When saving Contents for the first time, Elements will generate a hash of the contents and provide that in the

version field of the response.

Updating Save Data

When updating Save Data, the client may specify one of two update modes: Forced or Checked update. The purpose of this process is to allow checking in the case that the client contains an outdated version of the save data document.

Checked Update

To preserve the integrity of your game or application's Save Data, we recommend attempting a checked update first. The basic process for performing a checked update is as follows:

1. The first time the player saves data, issue a create request to save for the first time.
2. As early as possible in the application lifecycle, fetch the save data from Elements.
3. Save both Version and Contents to local storage or in memory.
4. When updating, supply the Version along with new Contents.

A checked update detects a scenario in which the same User operates on multiple devices. For example:

- User logs in on Device A, creates a Save Data document, and successfully creates the Save Data in Elements.
- User logs in on Device B, reads the Save Data document from Elements, and stores locally.
- On Device B, the user makes changes to the Save Data and successfully updates Save Data.
- On Device A, user logs in, makes changes, and for some reason is unable to retrieve the data from Elements (e.g. - Airplane Mode).
- Once Device A has a connection again, the update will fail due to version mismatch.
- At this point the client can perform one of a few actions:
- Attempt to merge the data.
- Present the option allowing the User to select which version of the data they wish to use.

Forced Update

When performing a forced update, Elements will ignore the version field and write the specified Contents directly to the database as supplied. Elements provides this as an option for when data

consistency isn't terribly important or as a conflict resolution strategy (as outlined in the example above).

Note

Using this option is dependent on the needs of your application.