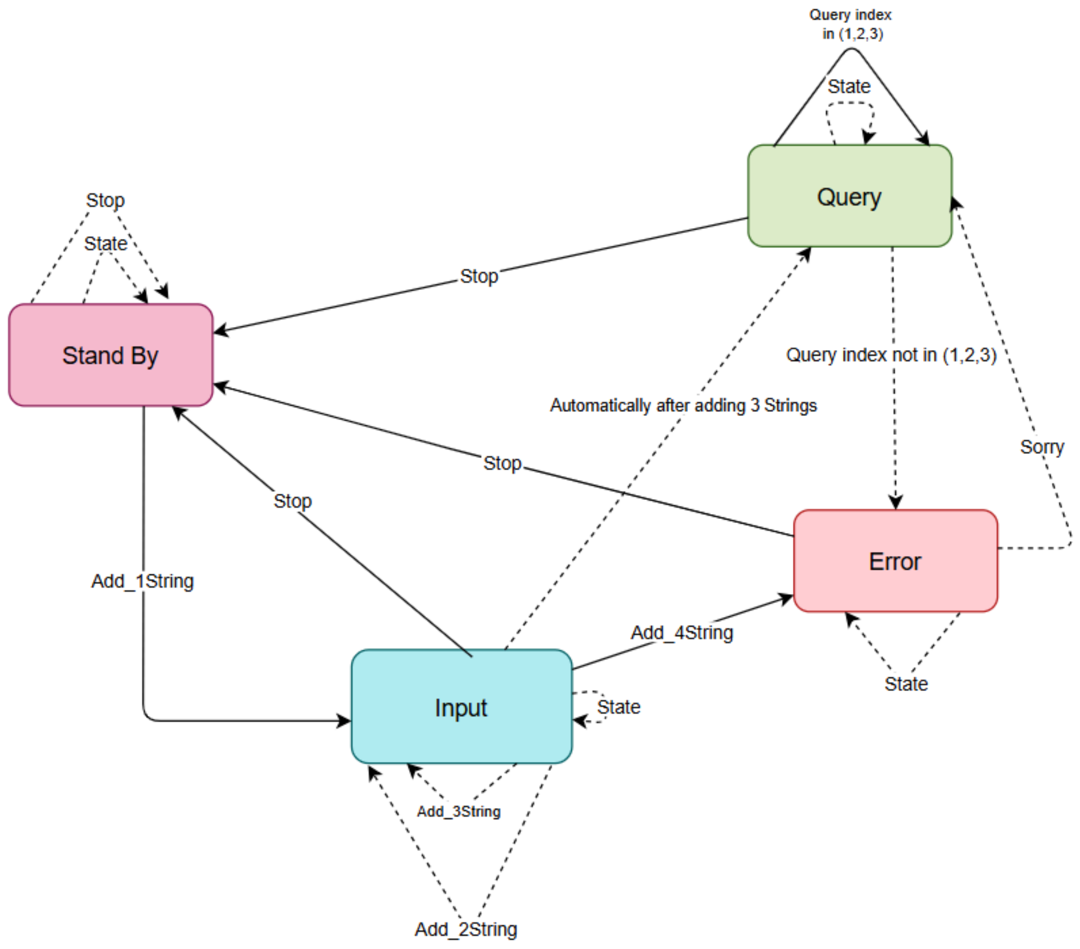
Storage API – State Machine Description

The **Storage API** enables structured handling of string inputs through a finite-state model. Its operation is governed by a well-defined set of states and transitions, each triggered by specific API commands. The following section details the system's design and provides a walkthrough of a typical usage scenario, as outlined in the official documentation.



States:

- **StandBy**:
  The initial state. In this state, the storage is empty and ready to accept new strings.

- **Input**:
  Entered after the `add` command is received. In this state, the system accepts up to three strings.

- **Query**:
  Entered automatically after three strings are successfully added. In this state, stored strings can be retrieved using the `query` command.

- **Error**:
 Reached when an invalid `query` index is provided. In this state, queries are disabled until the system is reset.

Transitions:

- **StandBy → Input**:
 Triggered by the `add` command (with the first string). Prepares the system to receive strings.

- **Input → Input**:
 Triggered by the `add` command with the second and third strings. This can happen up to two more times.

- **Input → Query**:
 Occurs automatically after the third string is added.

- **Input → Error**:
 Triggered if a fourth string is added (invalid).

- **Query → Query**:
 Triggered by a `query` command with a valid index (1, 2, or 3). Returns the string at the specified index.

- **Query → Error**:
 Triggered by a `query` command with an invalid index (not 1–3).
- **Error → Query**:
 Triggered by the `sorry` command. Restores access to the stored strings.

- **Any State → StandBy**:
 Triggered by the `stop` command. Resets the system and clears stored strings.

- **Any State → Same State (Self-loop)**:
 Triggered by the `state` command. Returns the current state information without changing the state.

Example Flow Analysis

This sequence demonstrates the behavior of the Storage API through a complete interaction scenario:

1. `http://t-tweak.gershon.info/storage/stop`
   → Sets the system to the **StandBy** state.

2. `http://t-tweak.gershon.info/storage/state`
   → Confirms the system remains in **StandBy**.

3. `http://t-tweak.gershon.info/storage/add`
   → Transitions to the **Input** state.

4. `http://t-tweak.gershon.info/storage/state`
   → Confirms the system remains in **Input**.

5. `http://t-tweak.gershon.info/storage/add?string=1_string`
   → Adds the first string. System remains in **Input**.

6. `http://t-tweak.gershon.info/storage/add?string=2_string`
   → Adds the second string. System remains in **Input**.

7. `http://t-tweak.gershon.info/storage/add?string=3_string`
   → Adds the third string. Automatically transitions to the **Query** state.

8. `http://t-tweak.gershon.info/storage/state`
   → Confirms the system is in **Query**.

9. `http://t-tweak.gershon.info/storage/query?index=1`
   → Retrieves the first string. System remains in **Query**.

10. `http://t-tweak.gershon.info/storage/state`
    → Confirms the system is still in **Query**.

11. `http://t-tweak.gershon.info/storage/query?index=5`
    → Invalid index. Transitions to the **Error** state.

12. `http://t-tweak.gershon.info/storage/state`
    → Confirms the system is in **Error**.

13. `http://t-tweak.gershon.info/storage/sorry`
    → Recovery command. Transitions back to **Query**.

14. `http://t-tweak.gershon.info/storage/state`
    → Confirms return to **Query**.

15. `http://t-tweak.gershon.info/storage/query?index=0`
    → Invalid index. Transitions to **Error**.

16. `http://t-tweak.gershon.info/storage/state`
    → Confirms the system is in **Error**.

17. `http://t-tweak.gershon.info/storage/sorry`
    → Recovers again. Transitions to **Query**.

18. `http://t-tweak.gershon.info/storage/state`
    → Confirms the system is in **Query**.

19. `http://t-tweak.gershon.info/storage/stop`
    → Resets the system. Transitions to **StandBy**.

20. `http://t-tweak.gershon.info/storage/state`
    → Confirms return to **StandBy**.