candidate file, verifies that the file actually contains `JSON` audit events, and sorts the files using the timestamps from the first event of each file. The result is a sequence of files that are subject to access using the log-reading functions:

- `audit_log_read()` reads events from the audit log or closes the reading process.

- `audit_log_read_bookmark()` returns a bookmark for the most recently written audit log event. This bookmark is suitable for passing to `audit_log_read()` to indicate where to begin reading.

`audit_log_read()` takes an optional `JSON` string argument, and the result returned from a successful call to either function is a `JSON` string.

To use the functions to read the audit log, follow these principles:

- Call `audit_log_read()` to read events beginning from a given position or the current position, or to close reading:

  - To initialize an audit log read sequence, pass an argument that indicates the position at which to begin. One way to do so is to pass the bookmark returned by `audit_log_read_bookmark()`:

    ```
    SELECT audit_log_read(audit_log_read_bookmark());
    ```

  - To continue reading from the current position in the sequence, call `audit_log_read()` with no position specified:

    ```
    SELECT audit_log_read();
    ```

  - To explicitly close the read sequence, pass a `JSON null` argument:

    ```
    SELECT audit_log_read('null');
    ```

    It is unnecessary to close reading explicitly. Reading is closed implicitly when the session ends or a new read sequence is initialized by calling `audit_log_read()` with an argument that indicates the position at which to begin.

- A successful call to `audit_log_read()` to read events returns a `JSON` string containing an array of audit events:

  - If the final value of the returned array is not a `JSON null` value, there are more events following those just read and `audit_log_read()` can be called again to read more of them.

  - If the final value of the returned array is a `JSON null` value, there are no more events left to be read in the current read sequence.

  Each non-`null` array element is an event represented as a `JSON` hash. For example:

    ```
    [
      {
        "timestamp": "2020-05-18 13:39:33", "id": 0,
        "class": "connection", "event": "connect",
        ...
      },
      {
        "timestamp": "2020-05-18 13:39:33", "id": 1,
        "class": "general", "event": "status",
        ...
      },
      {
        "timestamp": "2020-05-18 13:39:33", "id": 2,
        "class": "connection", "event": "disconnect",
        ...
    ```