Storage.Set

Method	**Set**
Module	Storage
Library	Core
Action	Set or change the value of the named persistent storage table.
Parameters	`Name` (string): The name of the persistent storage table. Case is not significant.
	`Value` (table): The table contents to be written to, or overwritten in, the named persistent storage table.
	`Append` (bool; optional, default `false`): Whether to append to whatever is already in the table (true) or overwrite it (false). **Available from v8.0 onwards**. It is not an error to append if the table does not already exist; in this situation the parameter's value is irrelevent. There is no check that the schema of the appended rows is the same as that of those already there.
	`MaxRowCount` (int; optional, default `1000`): When appending, limit the table to this maximum number of rows. **Available from v8.0 onwards**. The range is 1 to the SelectRowsLimit configuration value (100000 by default). This should be specified only if Append is true. There is no limit when overwriting. Rows already in the table are never deleted when `Append` is `true`, and `MaxRowCount` only affects the addition of new rows - it never truncates. For example, if the table already contains 10 rows and we try to append more with `MaxRowCount` set to 5 , then the table will still contain just the 10 original rows at the end of the operation.
Return values	`Name` (string): The name of the new or updated persistent storage table.

Examples

A simple example.

```
@table = Agent.GetSummary();
Storage.Set(Name: "Agent_Summary.Oct2018", Value: @table);
```

The method overwrites whatever was previously in the table. If you want to **accumulate** data by appending, these show examples of a task that would be run periodically to record who is logged on at that time. They also return the accumulated data.

Version (a) not using Append, up to v5.2

```
@loggedOnUsers = Users.GetLoggedOnUsers();
@newEntries = SELECT *, datetime("now") AS Timestamp FROM @loggedOnUsers;
@storageName = SELECT "LoggedInAudit" AS Name;

// Store data
@found = Storage.Check(Name:@storageName);
IF (@found)
    @currentEntries = Storage.Get(Name:@storageName);
    @newEntries = @currentEntries + @newEntries;
ENDIF;
Storage.Set(Name:@storageName, Value:@newEntries);
Storage.Get(Name:@storageName);
```

Version (b) using Append, from v8.0 onwards

```
@loggedOnUsers = Users.GetLoggedOnUsers();
@newEntries = SELECT *, datetime("now") AS Timestamp FROM @loggedOnUsers;
@storageName = SELECT "LoggedInAudit" AS Name;

// Store data
Storage.Set(Name:@storageName, Value:@newEntries, Append:true);
Storage.Get(Name:@storageName);
```

Platforms

- Windows
- Linux
- MacOS
- Solaris Intel
- Solaris Sparc
- Android

Notes

See User Defined Persistent Storage page for guidance using persistent storage tables.

The following methods exist in the Agent Storage module:

- Storage.Check — Tests the existence of a user defined persistent storage table.
- Storage.Delete — Removes an existing user defined persistent storage table.
- Storage.Get — Indicate whether a persistent storage table of the specified name is present and return its contents if present.
- Storage.GetRemote — Retrieve the requested datum directly from the Platform central repository.
- Storage.Set — Set or change the value of the named persistent storage table.