User Defined Persistent Storage Tables

Summary

Guidance and examples for the User Defined Persistent Storage feature.

This feature was introduced in Tachyon 4.0.

On this page:

User Defined Persistent Storage Tables

This feature allows SCALE to store and retrieve tables of data. Tachyon Activity Record (TAR) uses its own persistent storage tables, which are read-only to SCALE, whereas user defined storage tables are fully manageable using SCALE. TAR and user defined storage tables are stored in a local, compressed and encrypted persistent store, which persists during a Tachyon client upgrade, uninstall and re-installation, unless specifically deleted.

A normal table created by SCALE is defined as a @variable which goes out of scope (get destroyed) when a block of code completes. Storage tables let you save and load the contents of a variable for use in code which may run later. This means instructions and policy rules can refer to (and modify) data created by other, previously executed instructions or policy rules. This is a powerful capability as it allows SCALE code to be *stateful*.

A simple example might be an instruction or rule which adjusts a stored value (such as a running counter or sum) each time it is executed and a specific condition is detected, such as when a named process executes. More sophisticated uses might include capturing detailed data - for example, a list of files - and comparing this data to a previous execution of an instruction or rule.

Table names:

- case insensitive alphanumeric strings and can be up to 128 characters in length
- proceeded with a keyword to distinguish them from other forms of persistent storage object.

Table contents are in JSON format. Alphanumeric strings present in the table contents have case preserved. That is, a number would be represented as a string such as "99", and storage of an empty table content is permitted.

> ⚠ Updating a table may fail if the disk is out of space or has other file-system problems.

Freeform Tags are also stored in persistent storage, but they only store individual name=value pairs instead of variables (tables).

Storage Methods

The following methods existing 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.

Each method requires the user defined persistent storage name to be specified.

Examples

Example use of User-defined Persistent Storage

```
@query = NativeServices.RunWmiQuery(Namespace:"root\\cimv2", Query:"Select Caption, WorkingSetSize from
win32_Process");
@workingsetsize = SELECT * FROM @query ORDER BY WorkingSetSize DESC  LIMIT 10;

@storage = SELECT "WorkingSetSize.Top10" AS Storage;

//@found = Storage.Delete(Name: @storage);
@found = Storage.Get(Name: @storage);
IF (@found)
     @result = SELECT "SUCCESS: " || Storage || " already exists and has been re-written." AS Result FROM
@storage;
 ELSE
     @result = SELECT "SUCCESS: " || Storage || " does not exist and has been created." AS Result FROM
@storage;
ENDIF;

// replace contents of table
Storage.Set(Name: @storage, Value:@workingsetsize);

@found = Storage.Get(Name: @storage);
IF NOT (@found)
    @result = SELECT "ERROR: Failed to write to " || Storage || "." AS Result FROM @storage;
ENDIF;

// return the result of checking and writing table
SELECT * FROM @result;
```