One Platform by Oscar Tark

In memory node execution environment

Documentation Revision 0.2.1



Philosophy

Silent Projects

One Platform Team is a network of people that work together even if not in direct contact on improving the One Platform core system, for usage, research into data scattering, and encryption methods, storage methods, asynchronous node-ing and in memory data basing. One Platform in itself being open source and “forkable” means that such does not have a single goal other than that to provide useful tools for learning and for use. We welcome people of all genders, ages and races to code with us, fork our projects or analyze and criticize them.

Why Open Source?

Open Source is a standard for which all code is visible to those that want to obtain it, and encourages the ability to fix problems, make alterations to personal preference and most of all to learn. Solid systems are created out of collaboration this is why open source is at the core of our projects. All projects and resources are available under an open source license.

Forking, Resale & Licensing Instructions

Are you looking to fork or sell an open source project? Please do remember that forking or altering is completely allowed yet so please read the license of the root system before releasing the new fork as to abide by a few courtesies the authors have declared.

Patenting of Open source projects is strictly not allowed and so is formal copyrighting as this is a breach of license. Make sure that open source is your cup of tea before drinking it.

One Platform

One Platform

The One Platform is a research project composed of one code repository (<https://github.com/Oscar-Tark/One-Platform>). Developed entirely in C# One is a unification of utilitarian tools, One system, multiple applications that communicate and learn from each other in schools or as individuals. Applications, files, user data, general variables are stored in encrypted object Databases these databases can either represent a single application or multiple applications depending on how the database has been programmed.

Commands are asynchronous they do not abide by flows and are completely independent from other commands. Programming dependent components with One can be tricky, it’s suggested that elements such as variable assignments and creation are executed beforehand. Scripts are written in C# and compiled on command by the .NET compiler.

One Platform

General Syntax

Syntax in scorpion is simplistic, the moto is to keep it simple. Scorpion contains only the necessary runtime commands for execution, memory management, function calling, databasing, IO, networking & other.

Every command starts with something called an directive. This allows One to determine what kind of command you will be executing:

directive.function(\*arg,\*arg,…)

Directives as listed:

|  |  |
| --- | --- |
| Accessor | Functionaliy |
| mem | Memory |
| one | Access Application commands |
| fnc | Function based commands |
| db | Database commands |
| io | Input output commands for files |
| net | Networking commands |
| gui | Winform/GTK# |
| cmd | Run Windows/Bash command |

Functions allow you to access a specific functional command within the system, these are written after the directive.

directive.function(\*arg,\*arg,…)

Functions as listed by directive:

One Platform

|  |  |  |  |
| --- | --- | --- | --- |
| Directive | Function | Arguments | Description(Short) |
| mem |  |  |  |
| mem | new | (\*name,\*name,…) | Create variable(s) |
| mem | delete | (\*name,\*name,…) | Delete variable(s) |
| mem | set | (\*name, \*value) | Set value to variable |
| mem |  |  |  |
|  |  |  |  |
|  |  |  |  |

Variable syntax:

|  |  |
| --- | --- |
| Syntax | Description |
| \*name | Get or set to variable with a specific name |
| \*”” | Get or set without existing variable |
| \*db@section@variable | Get or set within database |

Memory

One uses a list structure to store and retrieve memory. All memory components are stored inside System.Collections.Arraylist components. Typical storage consists of one System.Collections.Arraylist with System.string name references and one System.Collections.Arraylist with actual values. There are two main forms of memory in One. System memory and database memory.

System memory:

Main Memory:

Named references contained in *AL\_CURR\_VAR\_REF* arraylist on objects.cs

Values contained in the *AL\_CURR\_VAR* arraylist on objects.cs

Main Functions:

Named references contained in *AL\_EVT* arraylist on objects.cs

Values contained in *AL\_ref\_EVT* arraylist on objects.cs

Recursive Functions:

One Platform

Named references contained in *AL\_REC\_REF* arraylist on objects.cs

Timer Values contained in *al\_rec\_tmp* arraylist on objects.cs

SHS (Shell Systems):

Named references contained in *AL\_SHS\_REF* arraylist on objects.cs

Values contained in *AL\_SHS* arraylist on objects.cs

System memory also contains system functions and other important System.Collections.Arraylist which will be uncovered in their related documentation sections. these in turn are stored in a system database: System\Data\one.vds

Database memory:

Databases contain multiple arraylists of memory, depending on your release of One the amount of arraylists may vary:

DB[arraylist{references{},values{]}, arraylist{references{},values{]},…]

ts