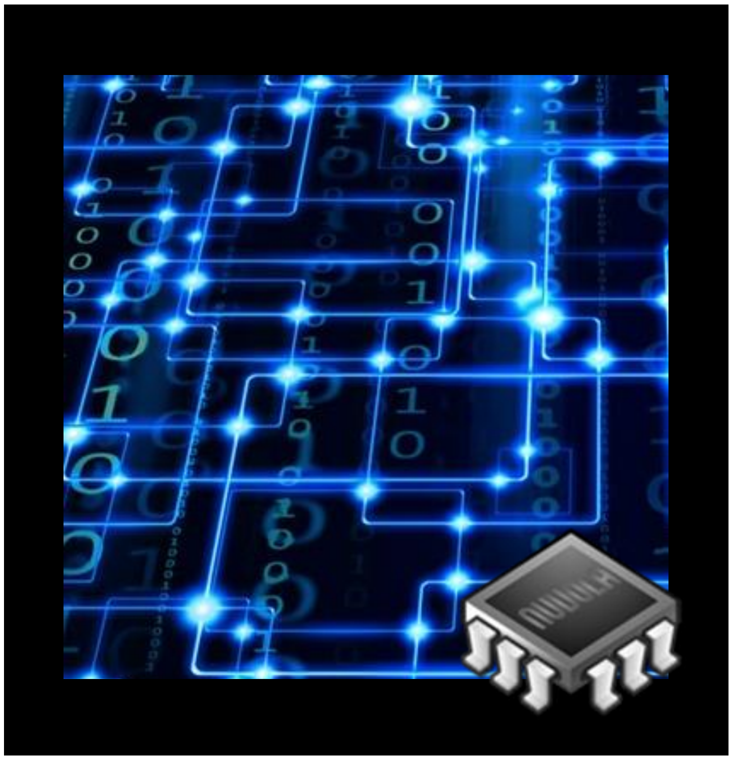


RX Platform

General Architecture and Usage Scenarios

Dusan Ciric

Version 0.1.4

Contents

Chapter 1. Introduction

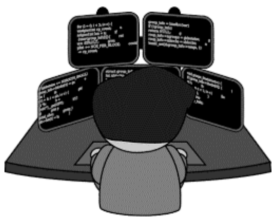
1.1. The Mission

RX Platform is distributed platform for building applications that exchange Real-Time data. The main intention for building this platform is to connect different kind of programmers (Industrial Languages, **C**, **C++**, **C#**, **Java**, **Python**) on the same project and to enable them to collaborate and share ideas and solutions. So, there are basically three groups of users that will benefit from using this platform.

Automation and Process Engineers can find a lot of familiar things, familiar Communication Protocols (Communication Ports etc…), Industrial Languages such as **Functional Block Diagram**, Timers, Alarms, Trends…

Engineers can benefit from this document because it will enable them to better understand what RX Platform offers as a core of solution for Gateway, Soft PLC (PAC), Computation Platform, Historian, Alarm Server etc.

RX Platform is **Industry 4.0** ready.



C Programmers will find asynchronous I/O and execution library that enables event based single threaded development of applications that will run on multiple processor cores and use kernel based I/O processing. RX Platform enables easy way to add real-time extension object to the platform such as new protocols or new programming language for example.

C programmers programing extension objects can use this document to better understand and diagnose problems they experience during development.

High-level language (C#, Java…) programmers can benefit from reading this document in better understanding of RX Platform classes hierarchy. These classes enable them to create higher level applications using predefined classes specified by Engineers. RX Platform enables programmers to directly use it for Edge IoT and Industrial IoT applications, custom applications such as Warehouse Management System or some other applications that need to rely on Real-Time data.

This document provides description of platform architecture and guidelines for using and extending its functionality.

Different Reference documents accompanying this document can be used by different type of users for more detailed description.

1.2. What RX Platform is?

RX Platform is essentially an application framework that enables any application (*Platform Host*) to deploy distributed real-time performance and accuracy.

This is an odd definition. What the Platform Host is? It's an application, any kind of application. You can write any host that uses RX Platform and all of its benefits.

Some hosts are simple like *Interactive Console Host* explained latter. Other hosts enable application to be written in higher level programming languages.

RX Platform is distributed, any device on local network on witch RX platform is deployed can be included in a single application. All the Hosts on the same network can access the same distributed RX Platform and share distributed Applications.

RX Platform is totally independent of the host type on which it is deployed and independent of operating system and hardware it is running on.

###TODO better explain

This enables an application to access, use and configure distributed platform through C#, Java, Python but also to build the parts of application in C, C++, IEC 61131 (FBD, Ladder…).

- RX Platform enables programmers to deploy C or industrial programming language program as a definition for upper language classes dynamic behavior.

It enables rapid development of following products:

- Industrial Automation Server,
- SCADA Communication Gateway,
- HMI,
- Edge IoT,
- Industrial IoT,
- Neural Network Applications,
- Gaming Communication Platform,
- Chat Communication Platform etc.

Chapter 2. RX Platform Console

Why start with a console? Well this is one of the platform interfaces you can easily use to access every part of the RX Platform itself. This is why in the rest of the book all the examples are shown through console interface. To better understand console interface in the following subchapters the basic structure of the RX Platform is described.

2.1. Folders

The primary storage structure of RX Platform Objects is exposed through Folder (Directory-Namespace) structure.

Namespace is abstraction visible to hosting application programmers.

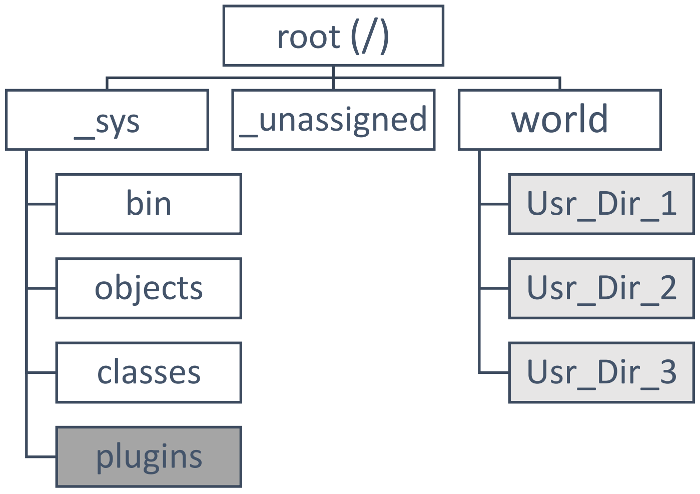
Directory structure is visible to Engineers and **C** programmers.

Console interface is based around Directory structure so we will explain this basic platform structure. This structure is more similar to Linux/BSD then to Windows command line because namespace view is case sensitive. Namespace view also applies some rules regarding directory naming.

Allowed characters are **0…9**, **A…Z**, **a…z**, **_**.

Root folder of the system is **/** and this is also a default delimiter, same as on Linux. Browsing the server using console is done using **ls**, **dir** and **cd** commands.

On the picture is the basic structure of RX Platform Folders.



More detailed structure of Folders structure will be explained through the document.

The **_sys** folder contains system related objects. There are four subfolders in this folder.

The first one in **bin** folder that contains mostly commands objects in server itself.

The **objects** folder contains system application, domain and objects running in platform internally.

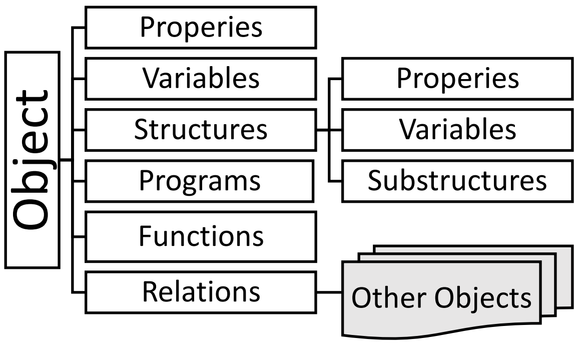
The **classes** folder….

The last folder **plugins** contain plugins (**C**, **C++**) running in the server externally. Each plugin has its own subfolder inside. One of the folders contained in plugins folder is a Host folder that contains host specific classes and objects.

Folder **unassigned** contains all the objects that does not have strictly defined runtime structure regarding **Application**, **Domain**, **Object** relations. (explained in next subchapter).

The last folder in the root folder is **world** folder. It contains user defined objects including remote ones. This folder is actually the place holder for user defined applications.

2.2. Namespaces and Runtime Structure



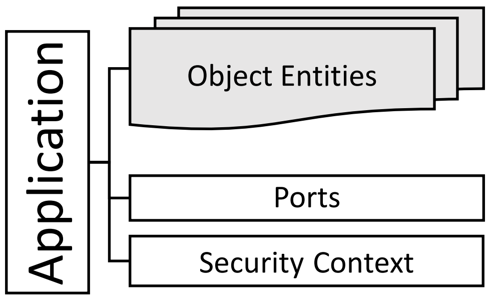
Structure of RX Platform itself is Object-Oriented. Everything is based on an **Object**. Object itself is structure which consist of **Properties**, **Variables**, **Structures**, **Relations**, **Programs, Functions** and some other entities explained latter.

Browsing Inside an Object is done in the same way as inside folders using **ls**, **dir** and **cd** command.

Object itself is placed in a namespace. Namespace of an object is a folder it resides in, for example to access object ***_sys\objects\io_pool*** in C# you can create new object of type **_sys.objects.io_pool**.

2.3. Objects and Runtime Structure

The runtime structure encapsulates execution security and threading context. It incorporates three basic types of objects Application, Domain and Object.



Application is a Domain object which is carrying Security Context information. Every code you run inside a single application can level down to the security context of the single Application Security Context. This enables you to track and manage automatically triggered actions that are often in a Real-Time Systems in a secure manner.

So, in order for you to have your code active you have to have at least one Application Object.

Besides Objects there are a Classes that define structure of every Object and all of the parts of it. Classes are defined in inheritance hierarchy. More about the Classes will be explained through the manual.

2.4. Accessing Console

Console is built at lowest engine level so with-it system can be easily configured, tested and administered. There are several ways to access the RX Console. Easiest way to access it is by running an *Interactive Console Host*. On windows just double-click icon ***rx_console.exe*** or start it from console, and on Linux just start ***rx_console*** program. It enables an engine to be tested in an easy way. You can execute program in any terminal emulator and you have interactive server up and running.

```
Process priority class set to RT!
Log started!
Performing initial log test...
Initial log test 0 passed. Delay time: 0.143 ms...
Initial log test 1 passed. Delay time: 0.245 ms...
Initial log test 2 passed. Delay time: 0.248 ms...
Initial log test 3 passed. Delay time: 0.251 ms...
Average response time: 0.248 ms...
Initial log test completed.
Starting RX Engine [ Testing Mode ]
=======================================

        ____________________
       / ____     _  __    /\
      / / __ \   | |/ /   / /\
     / / /_/ /   |   /   / /\
    / / _, _/   /   |   / /\
   / /_/ |_|   /_/|_|  / /\
  /___________________/ /\
  \___________________\/\
   \ \ \ \ \ \ \ \ \ \ \

Interactive Console
Interactive Console Host Ver 1.0.1706.220045

interactive@RX0001:/>█
```

Another way to access the console is terminal access. It is based on telnet protocol so enables the device to open the physically secure connection on hardware COM port. You can also designate some separate ethernet card for administration and access it with telnet support. This requires full duplex support on terminal line. Terminal server forces clients to use remote echo for each character so it produces a lot of small network packages and is therefore not to be used over large network segment. The previous approach enables safer access to server regarding communication errors and other physical factors.

Actually, TCP/IP telnet access to RX engine is supported only as a replacement for COM port for easier access with ethernet port, and for testing purposes on local machine or in secure and isolated network surrounding (default port when started as interactive console is 12345, so it's *telnet 127.0.0.1 12345* and you'll connect to it).

```
        ____________________
       / ____     _  __    /\
      / / __ \   | |/ /   / /\
     / / /_/ /   |   /   / /\
    / / _, _/   /   |   / /\
   / /_/ |_|   /_/|_|  / /\
  /___________________/ /\
  \___________________\/\
   \ \ \ \ \ \ \ \ \ \ \

Terminal Server Ver: 0.7.1706.222207
Interactive Console Host Ver 1.0.1706.220045

console@127.0.0.1:/>█
```

2.5. Console Commands

The starting point of any console that is new is basic ***help*** command. You can try it and get list of all commands with basic explanation.

You can use standard commands ***dir***, ***ls*** and ***cd*** commands for moving around Directories Structure.

2.5.1. Collecting Information

There are several commands that displays information about, hardware, versions, statuses of the platform and its objects. Detecting system on what platform is running is done by using command **pname**. Output is shown below:

```
interactive@RX0001:/>pname
System Information
=======================================
Engine Name: RX0001
Engine Version: Atom Ver 0.2.1707.290112
Library Version: 0.7.1707.290112
Host: Interactive Console Host Ver 1.0.1707.290112
OS: Windows Workstation 10.0.15063 [x64]
CPU: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz ; Total Cores:8  [LE]
Memory: Total 16279MB / Free 12199MB
Page size: 4096 bytes
interactive@RX0001:/world>█
```

Every item in Directory Namespace has information that is accessible through ***info*** command followed by object name (either relative or absolute).

```
interactive@RX0001:/_sys/bin>info cmd_manager

INFO
--------------------------------------------------------------------------------
Name       : cmd_manager
Full Path  : /_sys/bin/cmd_manager
Type       : OBJECT
Attributes : r--- s---o----

--------------------------------------------------------------------------------
Value      : cmd_manager
Quality    : g - - --------------- ----
Time stamp : 2017-08-21 20:39:06.446

--------------------------------------------------------------------------------
Class      : _CommandsManager
Has Code   : true
interactive@RX0001:/_sys/bin>█
```

Besides Items general information code command enables to see information about C or C++ code behind item implementation as shown below:

```
interactive@RX0001:/_sys/bin>code cmd_manager
CODE INFO
--------------------------------------------------------------------------------
name       : cmd_manager
subsystem  : rx
--------------------------------------------------------------------------------
file       : basic\rx_commands.h
class name : basic::commands::server_command_manager
version    : 0.5.0
compiled   : 2017-08-21 07:11:11
comment
/*
server commands management object
bin folder in file hierarchy
*/
interactive@RX0001:/_sys/bin>█
```

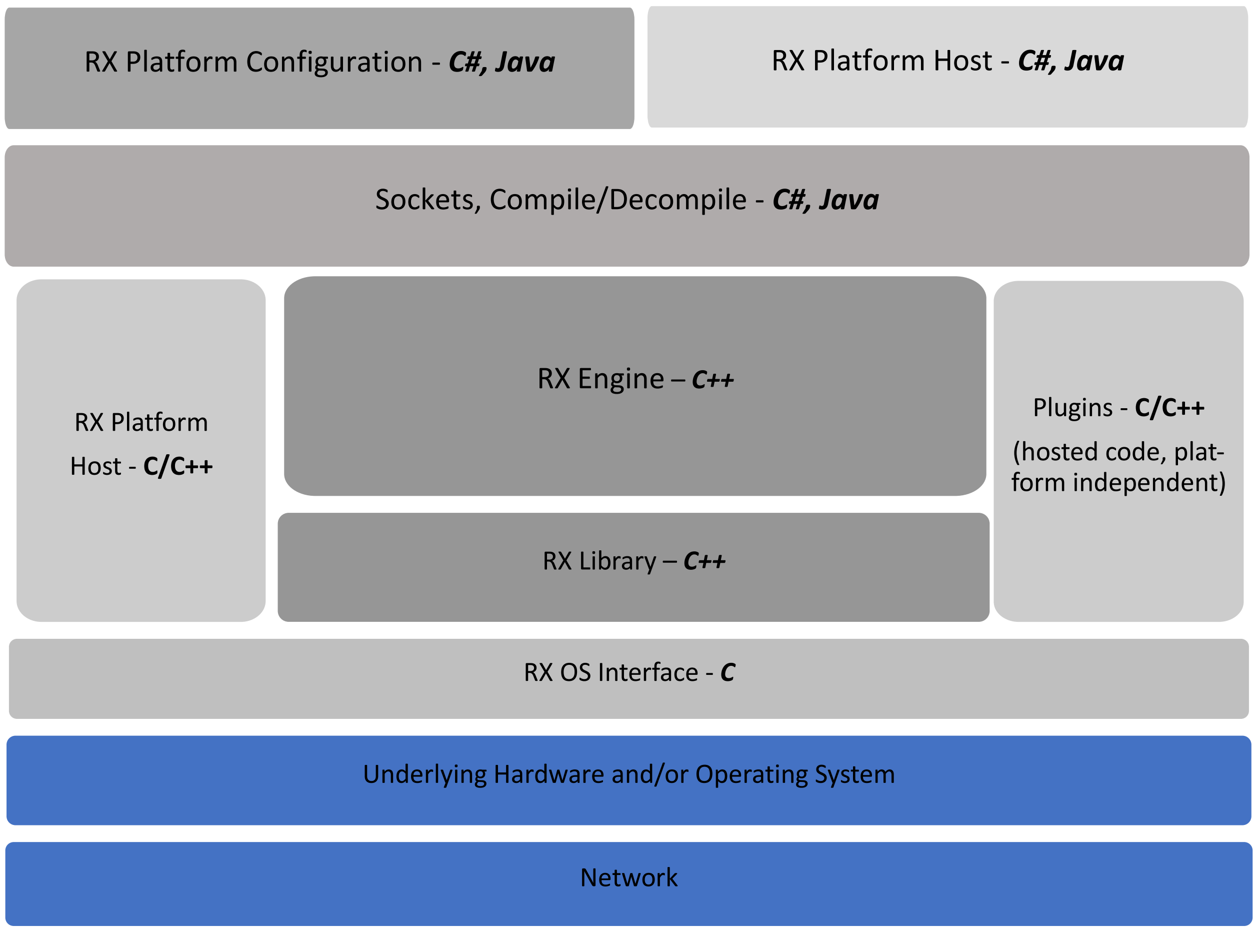
More details about these items information can be found through this manual.

Chapter 3. RX Platform Architecture

This chapter describes RX platform architecture and provides overview of technologies used and design principles of the platform itself.

3.1. Software Architecture

Overall architecture of entire system is shown on picture below:



RX Platform Configuration - C#, Java
RX Platform Host - C#, Java
Sockets, Compile/Decompile - C#, Java
RX Platform Host - C/C++
RX Engine – C++
Plugins - C/C++ (hosted code, platform independent)
RX Library – C++
RX OS Interface - C
Underlying Hardware and/or Operating System
Network

This architecture enables writing totally platform independent code as long as you do not access the network and underlying hardware and/or operating system directly.

3.1.1. RX OS Interface

RX OS Interface basically allow to create platform layer to compile on any platform. You can implement ***C*** library for a specific OS/HW platform and you will be able to compile entire RX Platform for that OS/HW combination.

Many information can be collected about the software and hardware environment it is running in. You can use ***pname*** command to inquire about the details regarding OS and hardware details. Below are shown outputs from executing this command on Linux VM and Windows in parallel.

```
interactive@ubuntuvm01:/>pname
System Information
========================================
Engine Name: ubuntuvm01
Engine Version: Atom Ver 0.2.1706.222328
Library Version: 0.7.1706.222328
Host: Interactive Console Host Ver 1.0.1706.222328
OS: Linux Ver:4.4.0-81-generic [x86_64]
CPU: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz ; Total Cores:2  [LE]
Memory: Total 3951MB / Free 961MB
Page size: 4096 bytes
interactive@ubuntuvm01:/>
```

```
interactive@RX0001:/>pname
System Information
========================================
Engine Name: RX0001
Engine Version: Atom Ver 0.2.1707.290112
Library Version: 0.7.1707.290112
Host: Interactive Console Host Ver 1.0.1707.290112
OS: Windows Workstation 10.0.15063 [x64]
CPU: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz ; Total Cores:8  [LE]
Memory: Total 16279MB / Free 12199MB
Page size: 4096 bytes
interactive@RX0001:/>
```

Information about the processor, available and free physical memory and page size are read directly through RX OS Interface. As presented in examples above, you can see that both OS's are running on the same processor. Below is presented code using RX OS Interface Functions that creates System information presentation:

```
// rx_win.h included on windows platforms
#include "windows/rx_win.h"
// rx_linux.h included on linux platforms
#include "linux/rx_linux.h"
// OS Interface functions
#include "rx_ositf.h"
// std c++ library of the compiler
#include "rx_std.h"


int main(int argc, char* argv[])
{
       ///////////////////////////////////////////////////////
       // OS Version
       char buff[0x100];
       rx_collect_system_info(buff, sizeof(buff) / sizeof(buff[0]));
       out << "CPU: " << buff << “\r\n”;
       ////////////////////////////////////////////////////////////////////////
       // Processor
       rx_collect_processor_info(buff, sizeof(buff) / sizeof(buff[0]));
       out << "CPU: " << buff
              << ( rx_big_endian ? " [BE]:" : " [LE]" )
              << "\r\n";
       ////////////////////////////////////////////////////////////////////////
       // memory
       qword total = 0;
       qword free = 0;
       rx_collect_memory_info(&total, &free);
       out << "Memory: Total "
              << (int)(total / 1048576ull)
              << "MB / Free "
              << (int)(free / 1048576ull)  << "MB \r\n";
       ////////////////////////////////////////////////////////////////////////
       out << "Page size: " << (int)rx_os_page_size() << " bytes\r\n";
}
```

3.1.2. RX Platform Host

Host is….

3.1.3. RX Library

RX Library Interface….

3.1.4. RX Engine

RX Engine Interface….

3.1.5. Plugins

Plugins are….

3.1.6. Sockets, Compile/Decompile

This level enables

3.1.7. RX Platform Configurator

User Application for engineers and programmers.

3.1.8. RX Platform Host

Why two hosts? Well …

3.2. Global software elements

Throughout *RX Platform* there are several software components that are accessible from every part of the system despite what part of system code is running on. These components enable diagnostics and serialization infrastructure for platform code.

3.2.1. Log

Log Object enables asynchronous logging capabilities for logging events. Log is implemented in a single thread so it preserves order of events. Using console command ***log*** you can handle log.

log hist can give you log history of some log records kept in memory (configurable by the host). Below is the output segment when executed on console:

```
2017-06-23 22:16:25.168 INFO@Security manager:User host@ubuntuvm01, security context created.
2017-06-23 22:16:25.168 INFO@Host Main:Initializing Rx Engine...
2017-06-23 22:16:25.175 INFO@Host Main:Starting Rx Engine...
2017-06-23 22:16:25.178 INFO@Security manager:User interactive@ubuntuvm01, security context created.
```

Log Object is used in many latter chapters.

3.2.2. Storage

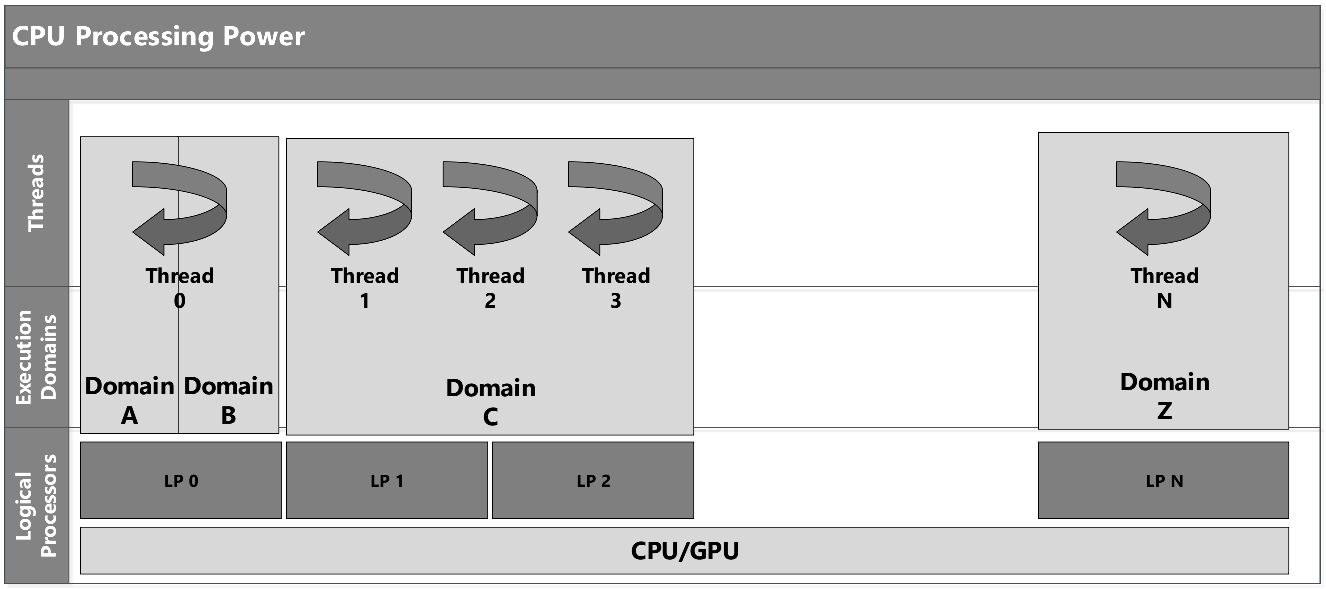
Storage is…

3.3. CPU Usage

In order to behave like real-time application RX Platform has to be properly configured by the RX Platform Host. It can be scaled up or down to run on either large server with strong processing power or on small device with embedded processor type.

Basic abstracts upon which RX Platform execution model is built on are Job, Thread, Domain and Logical Processor. Jobs are execution segments that can be created by some I/O operation, timer execution or send from code directly. Threads are abstract that represent OS thread objects. Domains are execution containers for Job objects. Every Job is connected to one Domain and can be sent directly to it throughout the platform.

The basic structure of RX Execution Model is shown on the next picture:



Some Domains, like the one that is shown as **Domain C** on the previous picture, have several threads and are used for asynchronous functions that are I/O intensive. Others Domains are actually executing programs and drivers These domains can provide single threaded environment for all the jobs in in queue, every job sent to some domain ends up in a single thread execution.

This is standard asynchronous application model. RX Platform enables Jos to maintain connections to their Threads so the global picture is different:

###TODO just started to explain here, will get better

The basic speed test for processor power can be tested by the ***log test*** command. It tests log responsiveness and on empty RX Platform Configuration can give HW platform performance. This command can be used as a responsiveness test of the platform on a specific hardware when configuration is empty, as shown on the output below:

```
interactive@ubuntuvm01:/>log test
User interactive@ubuntuvm01 performing log test.
Console log test 0 passed. Delay time: 0.022 ms...
Console log test 1 passed. Delay time: 0.053 ms...
Console log test 2 passed. Delay time: 0.033 ms...
Console log test 3 passed. Delay time: 0.066 ms...
Average response time: 0.0506667 ms...
User interactive@ubuntuvm01 log test completed.
interactive@ubuntuvm01:/>█
```

This gives approximate time for one processor context switch on given platform.

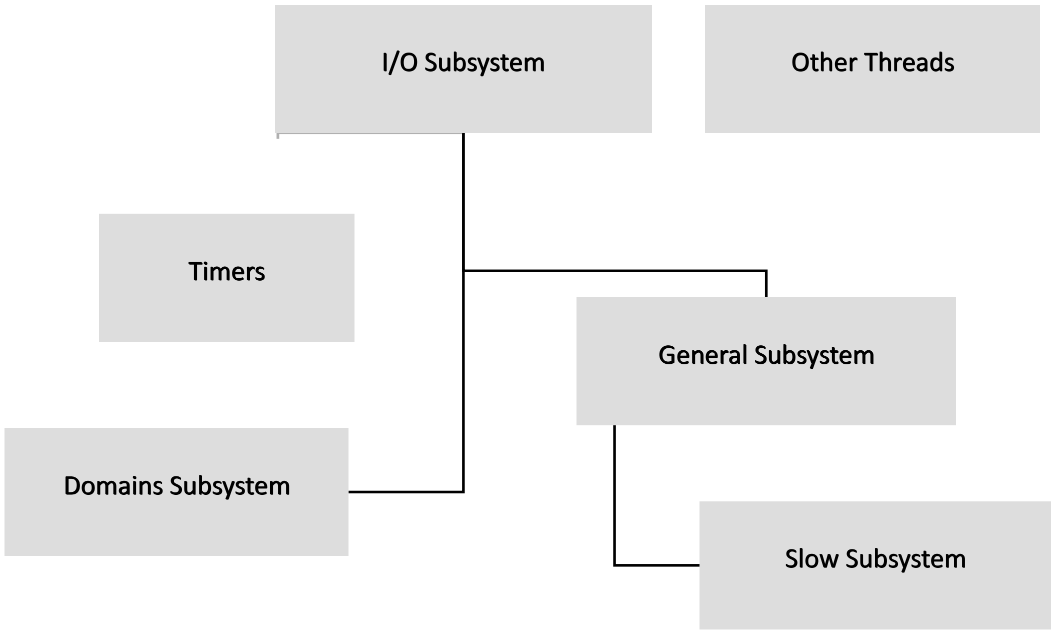
The server processing part is divided in several *Execution Domains*. Domains run programs in a single threaded environment. So, platform can be scaled up by increasing number of available Domains regarding available physical cores and threads available on CPU platform is running on. The platform is enabling communication between Domains and still preserve single threaded environment.

This kind of architecture actually enables to run platform as a plain single threaded application, and also to run it as a powerful server based application platform.

It is recommended to run RX process as real-time process if Operating System RX is running allows it. The platform is optimized to run as real-time process and gives best results regarding speed and responsiveness when it's priority is raised on Operating System itself. The process priority has to be set outside of the platform itself and it is responsibility of RX Platform Host. Some hosts that are included with the platform provides this capability (it is possible to turn off this option on each one of them).

The RX threading model consists of three major parts: I/O subsystem, Domains subsystem and General subsystem. Besides these three major parts there might be other threads running in system that perform different specific tasks (log framework is one of them for example).

Basic architecture of threading model subsystems is shown in picture bellow:



At the top of the hierarchy is I/O subsystem with timers that is only part of platform that needs to run in any configuration. The Slow Subsystem is replaced by General subsystem if it is not configured.

3.3.1. I/O Subsystem

This subsystem of RX Platform consists of one or any number of threads depending on the host initial settings. These threads provide processing power for interfacing to the underlying hardware and network resources and run on higher priority. Part of this this system are also optional 1 or 2 system timers. One of them is system timer for most of the system events regarding usually I/O operations and some high-speed applications. This timer is running on highest thread priority in system, besides real-time threads. The other timer is for lower accuracy rated jobs such as calculations for analog process value for example. The I/O subsystem is responsible for read and write operations on operating system objects (handles), for timer implementations and can be used for regular process tasks. Basically, it consists of one *kernel dispatcher object* from *RX OS Interface* and from threads acting as timers only. RX Platform Host can scale this subsystem such as set up number of I/O worker threads and witch timers are active in subsystem. RX Server Library includes objects that use this subsystem such as network sockets, file system, communication ports that can be used to create server extensions for protocols in an easy and straight forward manner.

Also, this subsystem is the only running subsystem in a single threaded environment.

3.3.2. Domains Subsystem

This is actually the main executing system of the RX Platform. Domain subsystem consists of any number of threads that execute all the programs (soft-logic, scripting, plugins…) in the system. Every program is running in a virtual single-threaded environment. This means that program can easily leave variables unprotected as a single threaded application and still get fast responses from I/O system. It enables programing safe system plugin in an easy way and still uses performance of multi-core processors.

3.3.3. General Subsystem / Slow Subsystem

This subsystem consists of zero or more threads as set by host settings. These threads are responsible to handle various job that cannot be assigned to any of the domain. Jobs and structures that use these subsystems are not thread safe and require careful locking mechanisms. Slow Subsystem is used for jobs that require some …

3.4. Security

This is a “***must have***” matter in industry so let’s discuss how it is implemented in RX Platform. Security implementation is not relying on hosting operating system and is fully RX internal. The only reason why host security is an issue is for storage and communication resources access.

The security in RX Platform is more complex because different protocols carry different levels of security. Different protocols in RX Platform are there to be treated as special user with own set of privileges. Also, there are special groups regarding safety of a protocol and/or network interface.

The balance between the speed and security is made that way that you can determine how big level of security code must have. The basic security concept is based on token that is an integer. This integer enables access to full security context for privilege check.

You can see currently active security context by using console ***sec active*** command as shown on picture bellow:

```
console@127.0.0.1:/>sec active
Dumping active users:

Id   User Name           Console  System  Port
==================================================================
[1]  host@RX0001         [ ]      [X]     internal
[2]  interactive@RX0001  [X]      [X]     internal
[3]  console@127.0.0.1*  [X]      [X]     telnet@TCP/IP[127.0.0.1]
console@127.0.0.1:/>█
```

On the output of the command, you can see listed all currently active security contexts. Console column displays if security context has user console attached to it. System column determines if security context is system based (has full access to system). This accounts that have System bit can benefit for better performance than more secured one. Security context with user name marked with ***** sign denotes currently context that executed command.

3.5.Versioning

This chapter explains Versioning Policy of RX Pltaform.

Chapter 4. RX Platform Values

This chapter describes values that are used in RX platform and their characteristics. Every value in RX Platform has several other characteristics such as quality, time-stamp and geographic position.

4.1. Basic Value Types

RX Platform has data types that are used for data representation. These basic data types are given in the table below:

Null	Null data type that represents no data available
Boolean	Single bit data type that can be either true or false (0 or 1)
Byte	Unsigned 8-bit integer data type
SByte	Signed 8-bit integer data type
Word	Unsigned 16-bit integer data type
Sword	Signed 16-bit integer data type
Dword	Unsigned 32-bit integer data type
SDword	Signed 32-bit integer data type
Qword	Unsigned 64-bit integer data type
SQword	Signed 64-bit integer data type
Float	32-bit floating point data type
Double	64-bit floating point data type
String	String data type that is based on UTF-8 notation
Time	Time stamp value
Uuid	Globally unified identifier
BString	Byte string that represents any byte's array
Complex	Complex number data type that represents complex number in 64-bit floating point parts

Any of these basic types can be represented in an array form so array is still a basic data type.

4.2. Value Origin and Quality in RX Platform

As in any Real-time system quality of data in RX Platform describe general information about value origin and value quality.

This information is stored in two 32-bit unassigned integer values.

The first 32-bit value is origin of the value and it's representing general information about the value. Structure of this 32-bit resister is shown below:

F	**B**	**T**	**C**	**E**	**G**		**Limit**	**Level**	**OK**
31	30	29	28	27	26	18	17 16	15 1	0

F – value is forced (substituted)

B – value is externally blocked

T – value is for testing

C – value is calculated

E – value is estimated

G – value has valid Geographical Position

Limit – value limit: 00 – not limited, 01 – low limit, 10 – high limit, 11 - constant

Level – value level (0,1…) - how many hoops through object boundaries this value had

OK – value has good quality has no additional quality information (memory optimization), basically 75% of every real-time application values can be represented as plain old good quality so this is significant amount of space that is saved

The second 32-bit value represents quality of the value itself. The structure of this 32-bit register is shown in the table below:

M	Details
31 30	29 0

M – master quality: 00 – Good, 01 – Uncertain (questionable), 10 – Bad(invalid)

Details – detail of value quality (depends on master quality)

Quality Details bits are depending of the master quality value and are shown in bellow tables:

4.2.1. Bad Quality

Quality Invalid	Invalid Input quality	**0x1**
Overflow	Overflow of proper representation	**0x2**
Out of Range	Value out of defined range	**0x4**
Bad Reference	Reference is out of order	**0x8**
Oscillatory	Value is Oscillatory	**0x10**
Failure	Internal or external failure	**0x20**
Device Failure	Device Failure detected	**0x40**
Configuration Error	Item can't be connected to specified input	**0x80**
Not Connected	Item is not connected to required input	**0x100**
Type Mismatch	Item's input is not valid	**0x200**
Syntax Error	Item's input expression is invalid	**0x400**
Division by Zero	Item's input has division by zero	**0x800**

4.2.2. Uncertain Quality

Old Data	The value may have changed and not updated	**0x1**
Inconsistent	Evaluation has detected inconsistency	**0x2**
Out of Range	Value out of defined range	**0x4**
Bad Reference	Reference is out of order	**0x8**
Oscillatory	Value is Oscillatory	**0x10**
Inaccurate	The value is not as stated accuracy	**0x20**
Initial Value	Variable's initial value	**0x100**
Invalid GP	Invalid value of Geographical Position	

4.3. Geographical Position in RX Platform

Geographical position defines position from where originally data comes from it is used in ***GIS*** systems where position of the data is an information that is relevant to the user. This data is contained in one 8-bit unassigned integer and can optionally contain additional data regarding weather data has relevant geographical position or not. This 8-bit data is defined as shown on the picture below:

INH	**OWN**	**POLAR**		
7	6	5	…	0	0

INH –Geographical position is inherited, we don't have it

OWN –Geographical position is here and valid

POLAR –Geographical position is given in polar coordinates

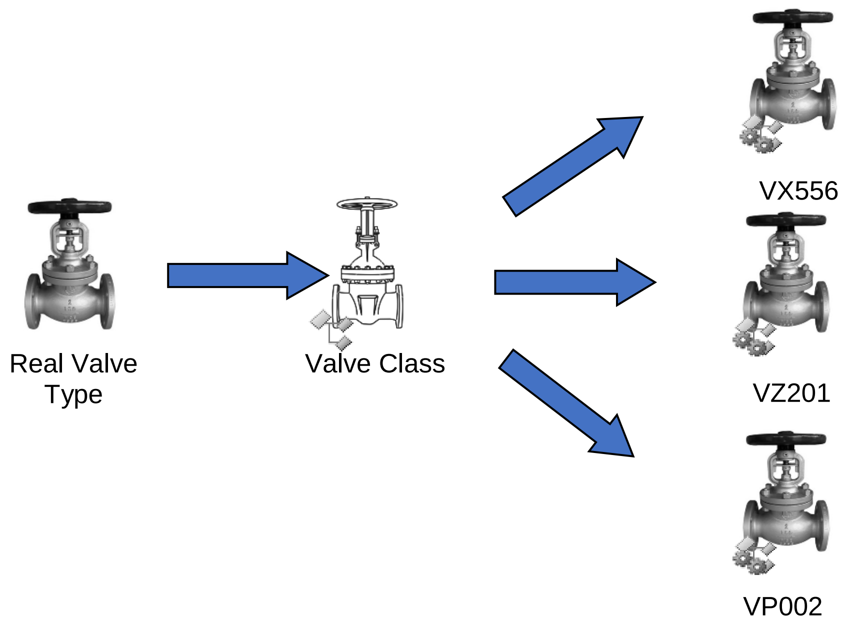
… –all zeroes

If bit **OWN** is set, there are additional three 64-bit integers:

Coordinate1 - coordinate data 1 according to values depending on **POLAR** bit
Coordinate2 -- coordinate data 2 according to values depending on **POLAR** bit
Coordinate3 -- coordinate data 3 according to values depending on **POLAR** bit

Chapter 5. RX Platform Classes

RX Platform is fully Object Oriented so every type of object you create must have its definition. These definitions are called classes. Every real-world type of equipment can be modeled as a class in RX Platform. These models can be used to instantiate several same types of equipment in the project as shown on the picture below:



In the above picture, we have ***Valve Class*** that models real-life type of valve that is used on the project. Exact objects instantiated from this class are ***VX556***, ***VZ201*** and ***VP002***. There are several types of classes that can be defined in RX Platform. These class types are:

- ***Object*** Classes – these class types are for modeling real-life equipment or common logic problems
- ***Domain*** Classes – these class types are for modeling domain types that represent single execution and logic context
- ***Application*** Classes – these class types are for modeling security and distributed logic spread on different nodes of RX Platform
- ***Port*** Classes – these class types are for modeling different industrial protocols with specific security settings and I/O data acquisition and distribution
- ***Variable*** Classes – these class types are for modeling different kind of I/O points that have specific filtering and/or event/alarms settings

- ***Mapper*** Classes – these class types are for modeling mappings of RX Platform data to different Industrial Protocols from the Server/Slave side
- ***Source*** Classes – these class types are for modeling mappings of RX Platform data to different Industrial Protocols from the Client/Master side or from internal RX Platform address space
- ***Structure*** Classes – these class types are for modeling data of different types that can be bound together

5.1. Common Classes Data

Every type of class mentioned above has common elements that defines them. These elements include Constant Properties, Properties and Variables.

Properties are simplest characteristics of the class and can be only defined as one of the simple type or arrays of simple types in RX Platform.

Constant Properties are values of simple type that are constant for the life-time of the instanced class. These

In the above-mentioned Valve Class Example, these properties can defined as in the table below:

Name	***Type***		***Value***
Description			

Chapter 6. Scripting Support

RX Platform have scripting support built-in. Scripting can be used to implement soft-logic problems in automation, building test platforms or any other non-critical part of real-time applications.

6.1. Python

Python implementation is based directly on python engine libraries (https://www.python.org). It can be run as either 2.7 or higher version for Python 2 and 3.5 or higher for Python 3 version.

Python version can be found using the command **python version** or **python ver** as shown below:

```
console@192.168.56.1:/world>python ver
Embedded Python Version
==========================================
2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609]

console@192.168.56.1:/world>█
```

Prerequisites

6.2. Java Script

Java Script implementation is based on V8 Java Script engine library (https://developers.google.com/v8).

Chapter 7. Programming Extensions

RX Platform enables extension software to be added to your solutions.

7.1. Writing Low-Level Protocols

Writing Low-Level protocols….

7.1.1. Port Objects

7.2. Writing Programming Languages

Writing Programming Languages…

7.3. Writing Low-Level Algorithms

Writing Low-Level algorithms…

7.4. Writing Neural Networks

Writing Neural Networks…

Chapter 8. Administration of RX Platform

This chapter is guideline for administration of RX platform. It provides documentation for…

8.1. RX on different platforms

Administration of RX on different platforms…

8.1.1. Windows

On Windows platform…

8.1.2. Linux

On Linux platform…

8.1.3. OS-X

On OS-X platform…