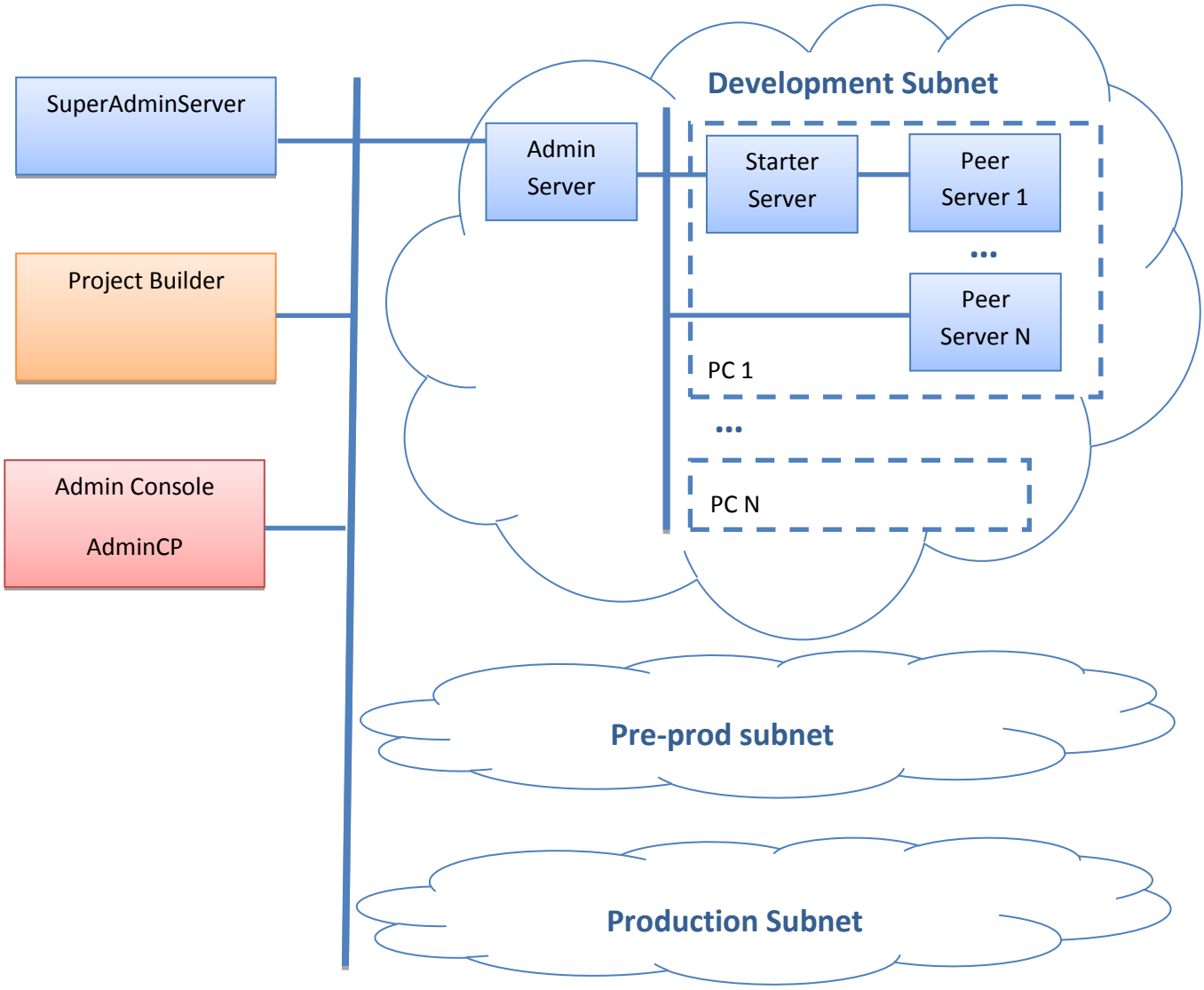
CrystalEngine

Structure and design

CrystalEngine based server solution easily created and scaled-out. All of the creation process is done using visual editors and the Administration is done via the Admin Control Panel.



Structural server scheme

It is recommended to implement at least 3 subnets for each project

1) Development – for structure and business logic development
2) Pre-prod – for testing and fine-tuning
3) Production – prod network with actual clients

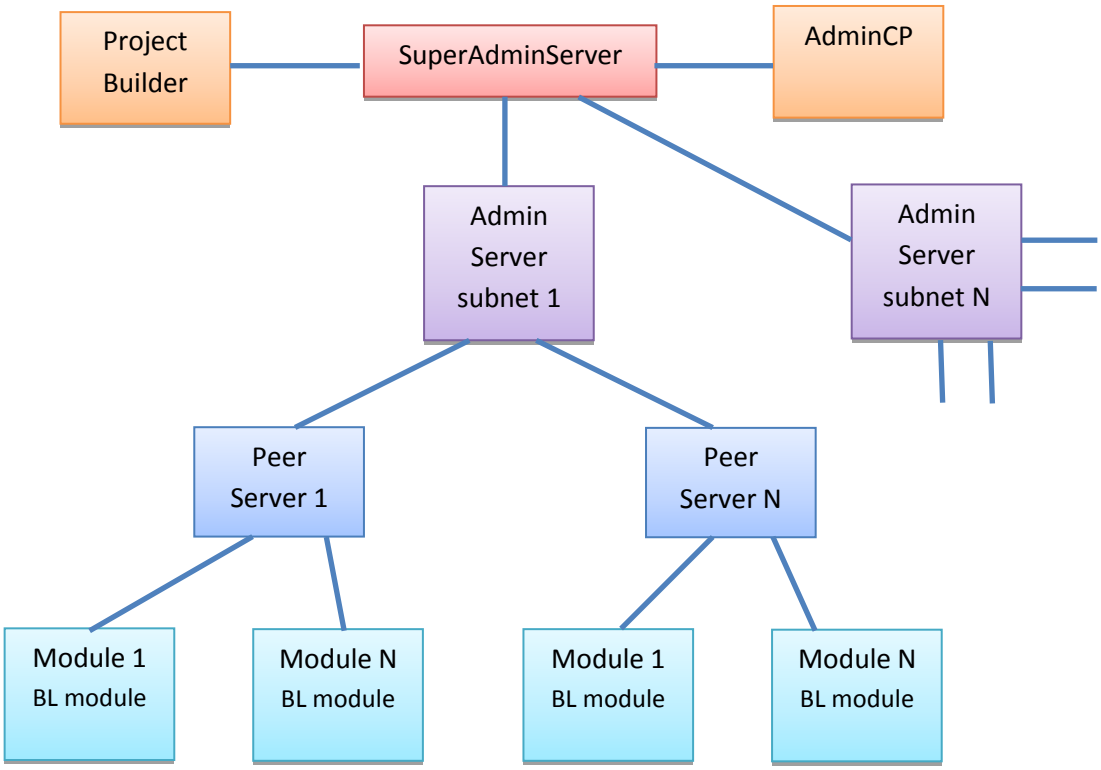
Package contents

The full version includes the following applications:

- **SuperAdminServer.exe** – subnets administration tool, serve-side designer integration (ProjectBuilder) and server administration
- **AdminServer.exe** – single subnet management tool
- **StarterServer.exe** – starts and stops serve components
- **PeerServer.exe** – main server, it runs the logic modules, provides network protocols and sockets(TCP UDP), it runs the business logic of the project
- **ProjectBuilder.exe** – main server component editor. This graphical editor creates the data structure for the whole project, automatically creates the necessary DB tables and file stores, client communication commands and inter-module communication framework.
- **AdminCP.exe** – graphic management tool for the whole grid. Installs PeerServers, links logic modules, links the needed server library depending on the subnet, archive management

All of the components can be run as Windows service or as an ordinary application

Server hierarchy scheme:



At to there is Super Admin server that manages the whole deployment and integrates with the PojectBuilder. It commands Admin servers which are managing their respective subnet Peer

Servers. The Peer Servers are running the logic modules. This structure allows rapid deployment and scalability without consideration for server's physical location.

For example, at peak times the AdminServer will launch additional Peer Servers to spread the load. Also the whole system can operate in a Cloud environment, for example Amazon. In this case the AdminServer uses Amazon API to utilize additional computing power when needed.

The used resources are released as soon as the user load is reduced.

Technical Requirements

Minimal PC requirements

Intel P4 CPU or equal AMD
RAM 500 Mb
HDD 200 Mb
OS Windows server 2003 x86
SQL MySQL, MSSQL
.NET framework 4

Recommended PC configuration

Intel i7 CPU or equal AMD
RAM 8 Gb
HDD 500 Gb
OS Windows server 2008 x64
SQL MSSQL (free version is supported)
.NET framework 4

The overhead for the server components are minimal, this almost all the computing power is used for project business logic.

The solution uses parallel processing "by design" for data processing. Thus additional CPU cores improve performance.

As of now an optimal PC configuration is around 100$ per month (i7 8Gb RAM Windows 2008 x64).

Personnel requirements

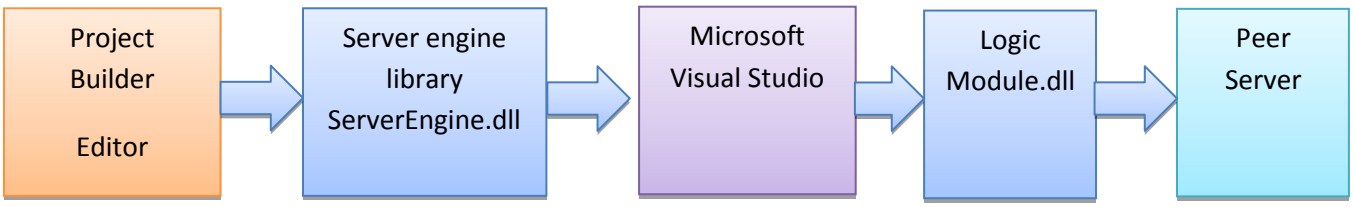
The server engine uses graphical editors. The design and implementation of the server component can be done by a mid-level developer. Deep understating of system programming is not required. For business logic module development the developer must be proficient in math and logic. Guru-style programming skills are not required due to the fact that the whole object structure is generated using the editor. All that the developer needs to do is to use them in his code without consideration for storage, transport and caching. All of those are managed by CrystalEngine automatically. For example, to get the username all that is needed is one line of code Root. Users [id]. The caching and the transport of session information is handled by the server engine.

Development speed

Using graphic editors and automatic source code generation reduces the development time by 2 to 3 times compared to the existing methods. The developer needs to click the mouse 3-4 times and tens of lines of code are generated already tested and fine-tuned.

Development cycle

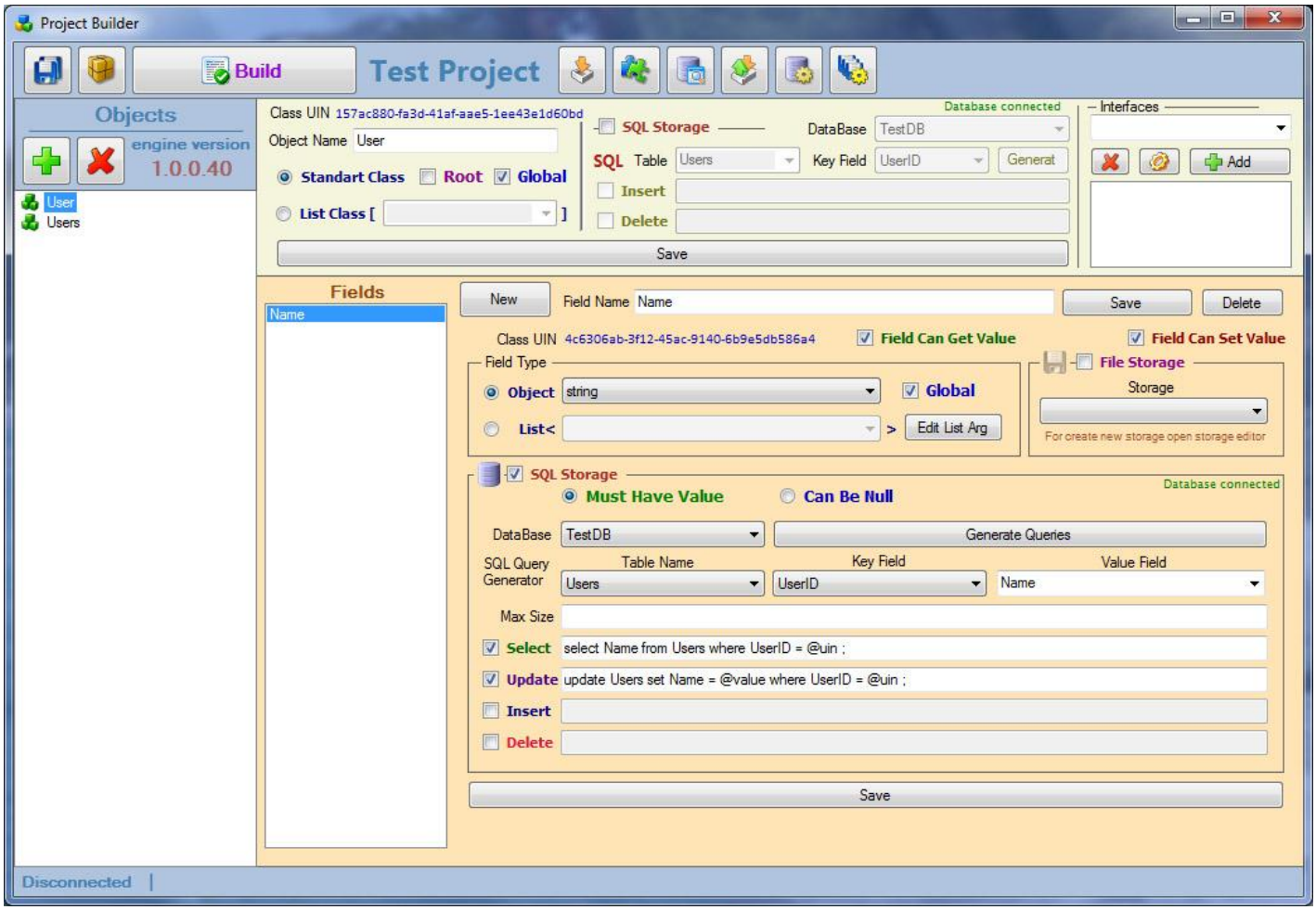
1) The developer installs SuperAdminServer and ProjectBuilder editor.
2) Using the ProjectBuilder the developer creates the object structure and commands (the design and the source code are saved on the SuperAdminServer).
3) After the developer clicks the «Build» button, the server objects are compiled and linked into «ServerEngine.dll», after which the library is automatically uploaded to the developer's pc. The developer includes the DLL in Microsoft Visual Studio and using this best-in-class IDE writes business logic scripts for the project.
4) After the BL libraries compilation, the developer uploads them to his server using AdminCP editor and launches them.



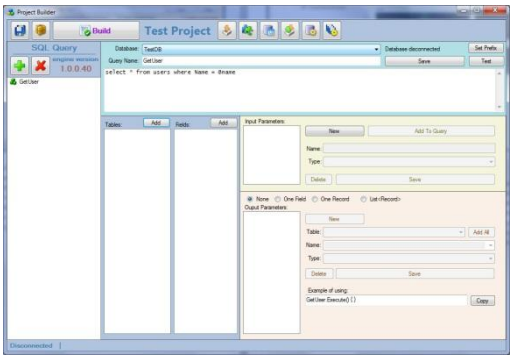
Server code creation process

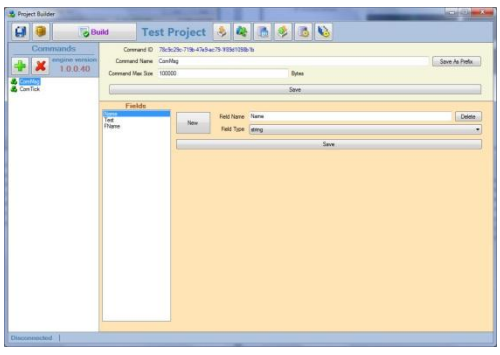
Using graphic editors and machine code generation (the output code is stable and tuned) allows to use the powerful Visual Studio IDE for business logic insures highly stable serve code thus reducing the risks for the whole project.

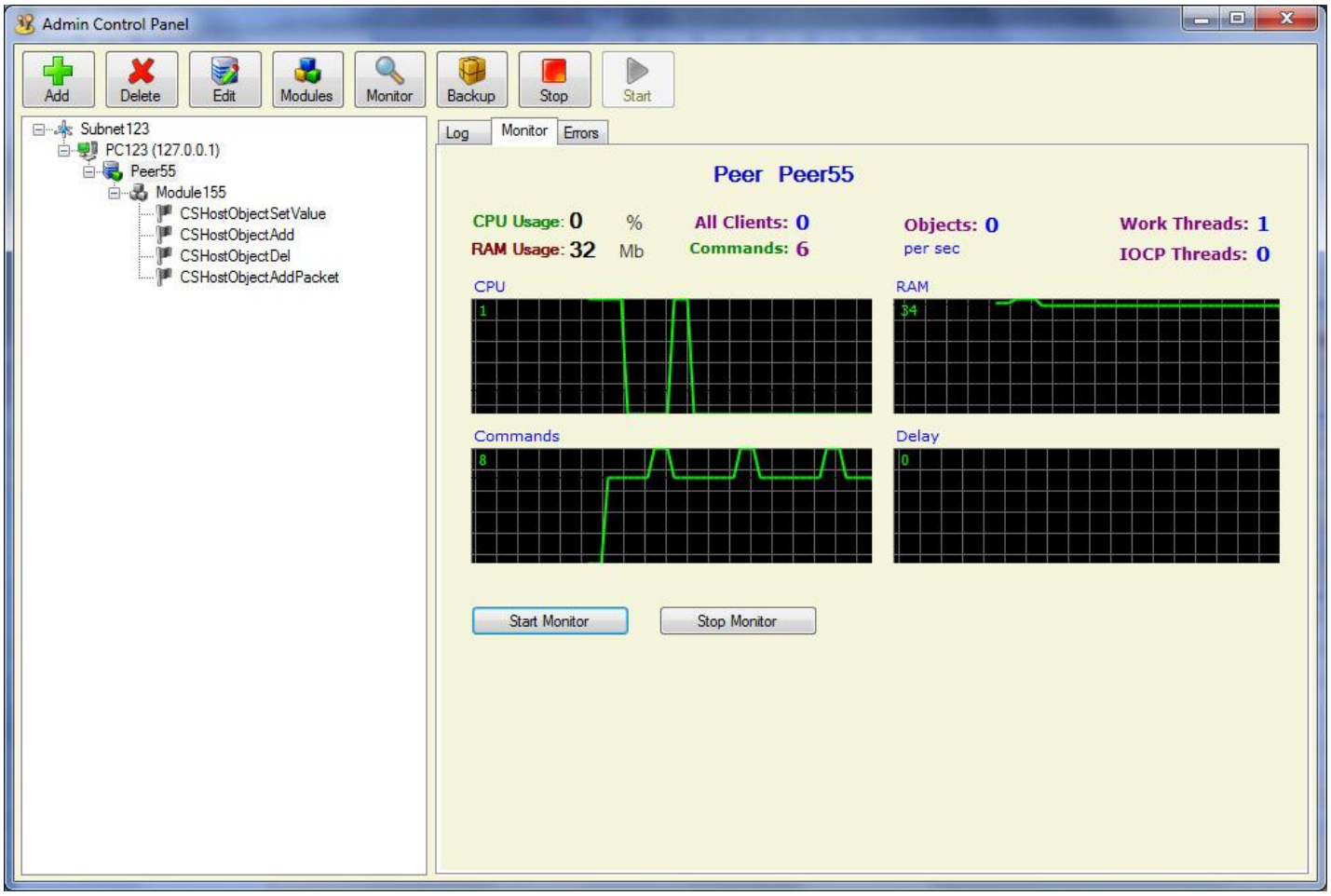
Editor Screenshots



«Project Builder» object editor. Although many options are available, object creation is a streamlined process since templates and wizards are available. Object creation is basically selecting a name and a type, the rest is done automatically. It is possible to change everything at any time if needed. All the editors are integrated into one window and are switched with tabs. All the editors are modular "by-design" taking into account the engine structure.





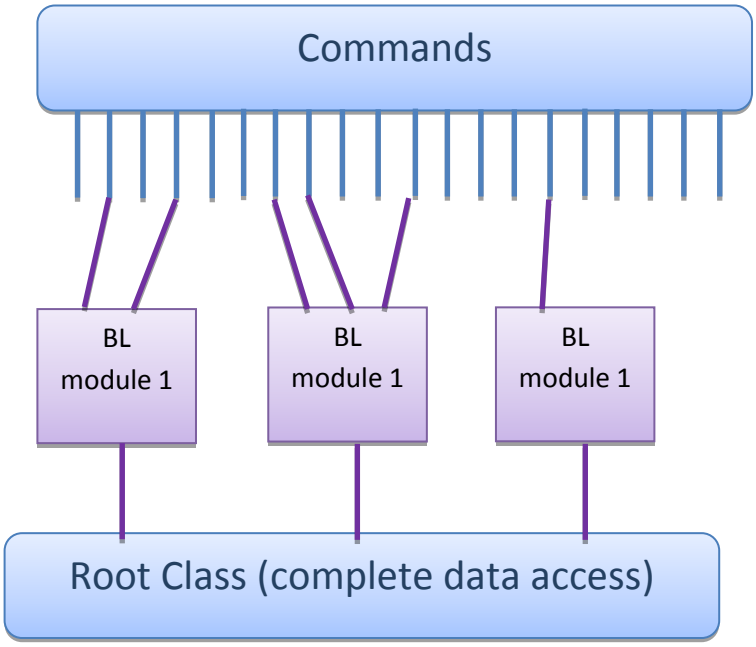


«AdminCP» tool with open Peer55 monitor tab. On the left there is the server tree where all servers and their status is displayed. By simply clicking a button any server can be stopped or started. A module can be launched any time without the need to restart the server –"warm boot", this is particularly important when a module needs to be update "on the fly" without disconnecting users. For every module there is a list of command it can perform.

Business logic module structure

Each module is implemented as a dll (.NET library). Each BL module implements data processing functions which are delivered by ProjectBuilder «ServerEngine.dll» library. Each method invocation is done through commands originating on the client or other modules.

Simply put, the BL module is linked to the required command and is activated by the command.



When started – the module connects to the defined commands and awaits invocation signal.

When stopped – the module disconnects from the command. The module access date through thre static Root class which has access to data from all available data stores: Из кэша

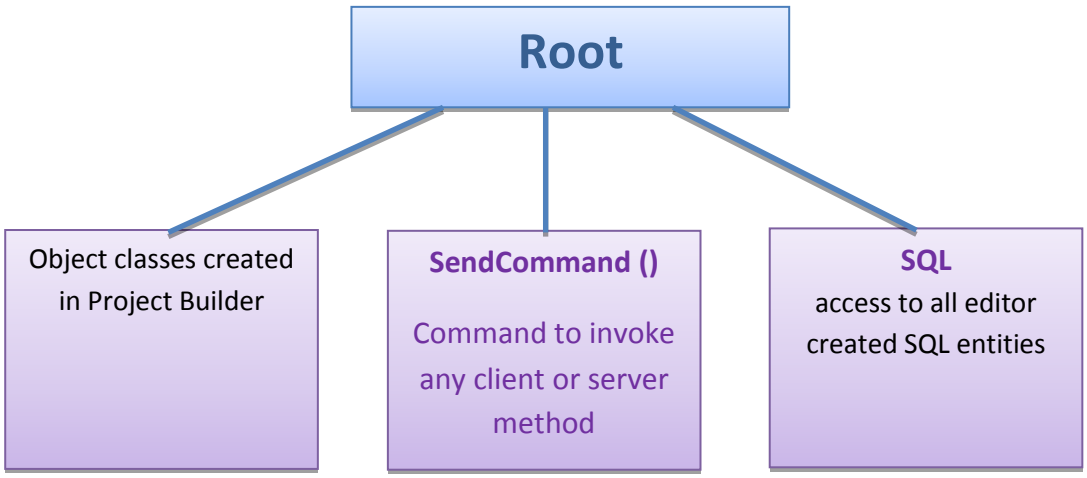
- Cache
- SQL DB
- File store

All using a single line of code

Any command can be sent to the client, module or data object.

Root class structure

Data access to all of the server-side data components is implemented by a single class, which greatly reduces development time and complexity. Just type Root in the editor and it will advise you on the usage of the objects and functions you've created

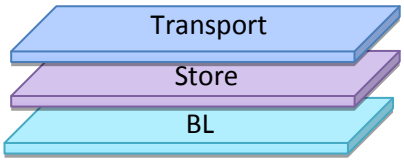


Key server engine advantages

- Very flexible **structure**, allows implementing different scale solutions. For example, it is very easy to create an optimal server for both location based games and large scale game worlds.
- **Supports any client**. Ability to use any client Unity3D, Flash, UDK etc, on any platform Windows, MacOS, Linux

- No limitation on BL development. The developer can utilize **any tools** for .NET or other compilers (C++, Delphi).

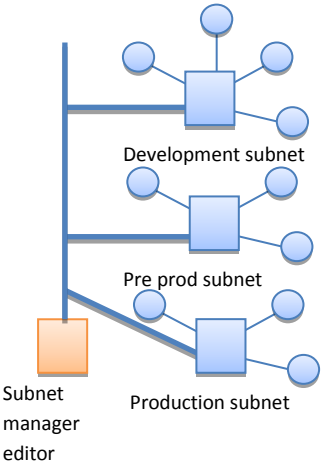
- Transport and data storage are independent from the logic layer, thus allowing to change the logic without the need to modify the data or transport layers



- Using the fast and distributed **NoSQL in RAM** for processing data storage(which is used by the BL during processing) allows achieving extremely fast data processing. The data is stored in the RAM of the server. The SQL DB is used for permanent data storage only.

- **Local transport system** (cross server communication) is using dynamic connection pooling thus allowing greater scalability. New connections are opened and closed on-demand. All connections are full duplex.

- **The Domain structure** allows streamline creation of subnets for different porpoises. For example, development, testing and production. Using the Subnet Manager you can easily install different versions of the server engine, if needed, fast rollback to the previous version can be done.



- **Reserve copy** of all the engine versions and object structures. A copy is automatically saved each time a "Build" button is pressed. Both the code and the data structure. All files are stored in a catalog system and sorted by date. Access and management of backups is done using the «Backup Manager», which is included in the Project Builder.

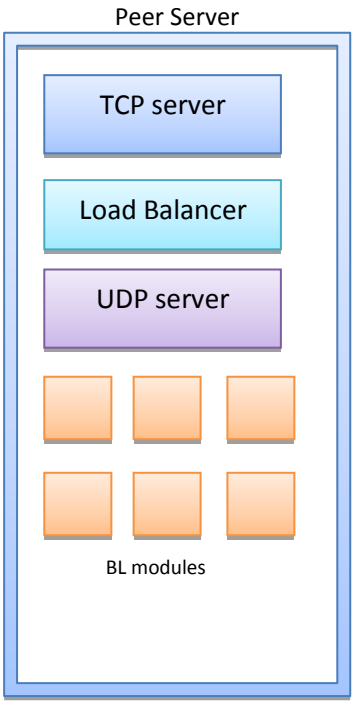
- **Automatic creation of tables and fields** in the SQL DB. There is no need to design the DB infrastructure and have a dedicated DBA. «Project Builder» will automatically create the DB, tables and fields to store the project's objects. Intuitive object creation wizard will assist you in creating population scripts and DB requests thus reducing development time. For advanced developers manual DB configuration possible at any time.

- **Easy system's object access** from logic modules. The developer is using his comfortable and customized Visual Studio IDE for the BL modules development. Each module is compiled into a .NET .dll which implements a simple interface in order to connect and disconnect to the server system. All server-side object access is done through a static Root class that is available anywhere. For example. We have a list of "Users", that contains object of "User" class with an attribute "Name". To get the username of any user, that can be stored on any server in the system or in an SQL DB, all that needs to be written in the BL module is Root.Users[id] where id the object ID in the system.

- **Transparent identification system.** Guid is used as an identification token that is used by Microsoft in the OS. Usage of this data structure reduces the risk of reoccurring numbers and allows the sage of common transparent object lists in NoSQL DB.

- **Easy to use command transport structure**. Local and global data exchange is done using a command structure. Much effort was put not only to a simple structure representation of the data structure but also to a fast data serialization and de-serialization (low level memory usage). Sending a command to a user, object or a module is done via one universal function: Root.SendCommand (id, command); where id is the user id, object or module id in the system. Since the identification system is transparent – the server engine will decide whom to send the command to.

- **Communication between business logic modules** is done using commands. Each module must implement two functions «Start()» and «Stop()» that perform subscription or detachment for command receiving. A module can subscribe to a certain command. Both broadcast and specific. If someone sends a command broadcast command, every module will receive it. For example, a module subscribes to «ComMove» command for object with an «id» that uses «onMove()» method to process the request. The code is as follows: ComMove.AddEvent(onMove, id); It is mandatory to have a matching «RemoveEvent()» to unsubscribe from the command. This is necessary to provide module "hot-swap" capability.
- **Hot-swap for business logic modules**. The server engine can swap logic modules without restarting the server and without disconnecting users. Stopping, uploading and launching a new version of the module is done using «Start» and «Stop» in the «Admin Control Panel» tool. While the servers provides service during the whole time.
- **Unified servers.** Each subnet (domain) of the server-based solution consists of several application types. Those are AdminServer» «StarterServer» «PeerServer». «AdminServer» is used for administration of other members of a particular domain, integration with «Project Builder» and «Admin Control Panel», license checks and communication with other domains. Identification of the subnet is done via the «AdminServer». «StarterServer» is used to start and stop processing components («PeerServer») on a particular host. «PeerServer» is the main processing unit. By default the «PeerServer» is an "empty" container for business logic and transport modules. It can be **compared to an empty box where the developer can put a** «LoadBalancer», «TCPServer» «UDPServer» or any other business logic module. Thus transforming the server to a LB, transport server, logic processing or any combination of the above. This is when the true power and flexibility of CrystalEngine can be seen. By designing the most suitable architecture for your particular project you can insure stability and fast response time. Since the optimal structure for project with small locations differentiates significantly from large open world environment. The ability to implement ANY logical structures is one the key advantage of CrystalEngine.
- **Unified servers.** Each subnet (domain) of the server-based solution consists of several application types. Those are AdminServer» «StarterServer» «PeerServer». «AdminServer» is used for administration of other members of a particular domain, integration with «Project Builder» and «Admin Control Panel», license checks and communication with other domains. Identification of the subnet is done via the «AdminServer». «StarterServer» is used to start and stop processing components («PeerServer») on a particular host. «PeerServer» is the main processing unit. By default the «PeerServer» is an "empty" container for business logic and transport modules. It can be **compared to an empty box where the developer can put a** «LoadBalancer», «TCPServer» «UDPServer» or any other business logic module. Thus transforming the



server to a LB, transport server, logic processing or any combination of the above. This is when the true power and flexibility of CrystalEngine can be seen. By designing the most suitable architecture for your particular project you can insure stability and fast response time. Since the optimal structure for project with small locations differentiates significantly from large open world environment. The ability to implement ANY logical structures is one the key advantage of CrystalEngine.

- **Entities for SQL queries organisation.** In order to insure high stability and performance it possible to use code instruments only for SQL requests. Methods and classes. Using the «SQL Editor», which is part of the Project Builder, the developer creates and tunes an SQL querry. The Project Builder automaticly generates a matching method and a result data structue. Thus the code is checked before the query is executed in production. The method is invoked through the Root class. For example, ResultUser user = Root.SQL.GetSQLUser(name); In the example a request is executed with a single parameter «name».
- **Editor and IDE integration. «OneClick» system**. The connection between the Visual Studio IDE and the editor is done through ServerEngine.dll and ClientEngine.dll. The libraries are automaticly compiled on the server and uploaded to the developer's PC. The libraries are installed in the GAC global cache and are available to any Visual Studio project in the «NET» tab. If a change is made to the proeject structure, the developer need only one click on the «Build» button in editor and all of the changes will be almost instantly available in the Visual Studio IDE. *This capabilty greatly increases the speed of the development process.*
- **Team project development**. CrystalEngine server supports collaborate development in the «Project Builder» editor. All of the changes done by one developer are instantly available to the whole team.