N001

Nuklear LORD Rewrite

Software Architecture – Client Application

Design plans for the Nuklear LORD Client. This includes C# software constructs and data types.

Version 1.0

Matt Buchwald
Nuklear LORD Development Team
2/19/2009

Revision History

Version	Date	Description
0.5	2/19/2009	Initial Draft Review

Overview

This specification will be used during the development of Nuklear LORD client software to track and document the major architecture decisions and intentions of the C# source code. Software development is a fluid process and it is not practical to pre-engineer the system down to every component prior to writing any code. It is more useful to design guidelines and an overall philosophy to the code structure to give the development and final code consistency and clarity.

As the software is developed, this document will grow and capture the design decisions made during the various phases of the project. Overviews of these phases are listed next.

Initial Design

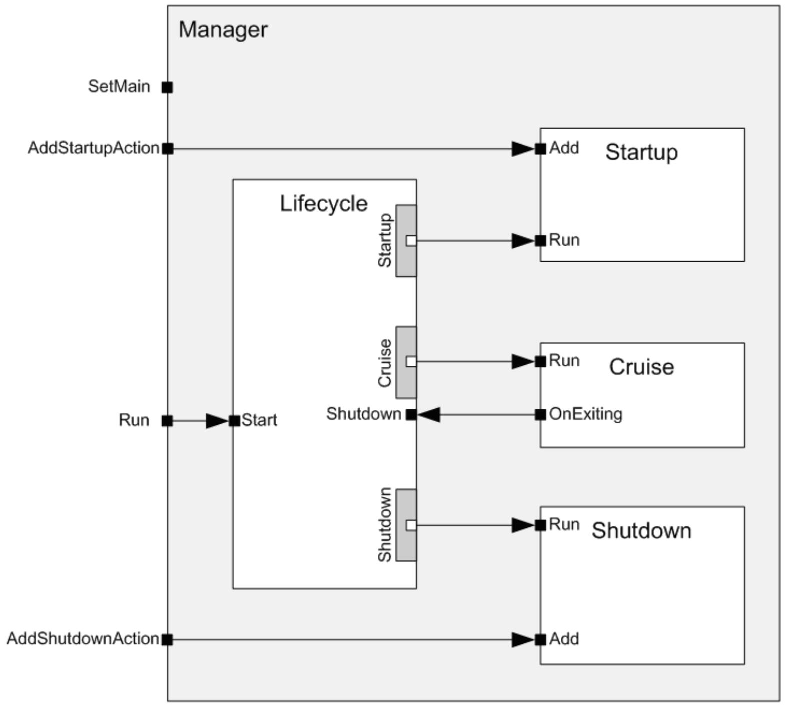
In the initial design, we present the data structures, process flow and similar abstract design considerations.

Assemblies

The C# development will be broken into several assemblies (dll or exe) to keep the code organized and manageable.

Lifecycle

The Lifecycle assembly performs the orderly startup and shutdown of the system.



Other assemblies may register actions to be performed during the startup and shutdown phases. SetMain is used to define the method that will be the main operating routine. There can be only one main method. The main method and actions must be registered prior to the Run command being executed.

Once the Run signal is sent, the following steps will be performed:

1. Execute all Startup Actions, in the order they were added. Messages for each action will be displayed on the Console.
2. Transfer control to the Main method. Wait until the method's execution ends.
3. Execute all Shutdown Actions, in the order they were added. Messages for each action will be displayed on the Console.
4. Exit Application.

NuklearTCP

The NuklearTCP assembly will provide a conduit between the UI client application and the server. All of the client to server library interactions will be encapsulated within the NuklearTCP assembly. This will decouple the server game logic details from the UI logic, and provide a cleaner path to modification of the game development system.

The core design of the assembly is already established. Sending and receiving messages is performed by using the *Write* and *Read* commands exposed by the assembly which use the **NuklearCommands** enumeration type exposed by the assembly to designate the type of data sent by the command.

Various data object classes are exposed by the NuklearTCP assembly for use with the Nuklear Commands found in the **NuklearCommands** enumeration.

This assembly is used by both the client and server applications for communication.

Application

This assembly will collect all the other assemblies, perform the inter-assembly binding and compile into the executable.

The application will take, as commandline parameters, the login name and password of the player.

Classes

BuilderBinder

The **BuilderBinder** class contains all other objects, creates them, and binds their events to each other. The builder binder will accept as arguments the commandline arguments passed to the client.

Members

Member	Type	Meaning
config	Configuration	Object that contains and manages the configuration data for the client.
communications	CommunicationManager	Objec that handles the communications with the Nuklear LORD Server application.
lifecycleManger	LifeCycle.Manager	Handles the startup, execution, and shutdown of the application.
args	String[]	Commandline arguments of the client application.

Methods

This section details the various public methods of the **BuilderBinder**.

Build

```
internal void Build()
```

The Build method creates all objects using their constructors to initialize.

Bind

```
internal void Bind()
```

The Bind method binds all objects together, wiring events as needed, including the startup and shutdown actions of the lifecycle.

Configuration

The **Configuration** class manages all configuration parameters for the client.

Properties

Member	Type	Meaning
ServerIP	String	Read-only parameter that represents the IP Address of the Nuklear LORD Server to connect to.
ServerPort	Int	Read-only parameter that represents the port used to connect to the Nuklear LORD Server.

Methods

This section details the various public methods of the **Configuration**.

Load

```
internal void Load()
```

The load method loads the config.xml of the client. At the present design this only loads server ip and port.

CommunicationManager

The communication manager handles logic needed for communicating with the server.

Properties

Member	Type	Meaning
Server	TcpClient	TcpClient connection to the Nuklear LORD Server application.
Config	Configuration	Configuration for the client application.
Login	String	Login name of the player
Password	String	Password of the player

Methods

This section details the various public methods of the **CommunicationManager**.

Connect

```
internal void Connect()
```

The connect method creates the server member by attempting to connect to the Nuklear LORD server.

Execute

```
internal void Execute()
```

The execute method contains the main client/server interaction logic.

Disconnect

```
internal void Disconnect()
```

The disconnect method instructs the client to end communications. This will send the Disconnect NulearCommand and then disconnect from the server.

Application Lifecycle

Building/Binding Phase

Upon execution, the commandline parameters are passed to the **BuilderBinder**. Then the Build method of the BuilderBinder is invoked, followed by the Bind method.

During the Build method, all objects are created.

During the Bind method, all objects are wired together. The login and password properties of the Communications object are set based on the commandline arguments. The Configuration object is attached to the Communications object by means of the Communications object's Config property. The lifecycle's main routine is set as the Communications object's Execute method.

Methods should be attached to the lifecycle's startup actions in the following order: The Configuration object's Load method, then the Communication object's Connect method. At the current date of the design, the Communications object's Disconnect method is the only shutdown action.

After binding objects, the lifecycle's Run method will be invoked, beginning the next phase.

Startup Phase

The lifecycle will immediately execute all startup actions in the order they were bound, resulting in the configuration being loaded and the connection to the Nuklear LORD server being made.

Cruise Phase

The lifecycle object will execute its main routine. Upon completion of the routine, the lifecycle will move on to the next phase.

Shutdown Phase

The lifecycle will execute all shutdown actions in the order they were bound, resulting in the client disconnecting from the Nuklear LORD server and the application closing.

Communication

Communication with the server application is performed over a TCP/IP connection. For the initial design of this project, the connection will be made over the localhost. The client will connect to the server using the **TCPClient** class. Once connected, the client can send and receive messages using the *Write* and *Read* commands exposed by the NuklearTCP assembly. The NuklearTCP protocol encodes object data into an xml stream using a command header and transmits this xml stream over a **Stream** such as a **NetworkStream**.

Sending a Command

To send a command, use the *Write* command exposed by the NuklearTCP assembly.

```
public static void Write(Stream stream, object o, NuklearCommands cmd)
```

Pass the client's **NetworkSteam**, the object to serialize into the xml stream, and the command from the **NuklearCommands** enumeration exposed by the NuklearTCP assembly. Most of the object classes used as data for the commands are also exposed by the NuklearTCP assembly.

Commands from Client to Server

NuklearCommands	Data Object	Meaning
LoginCommand	LoginData	Sends username and password to the server.
KeyInputCommand	KeyData	Sends a single keystroke to the server.
TextInputCommand	TextData	Sends a line of text to the server.
DisconnectCommand	`null`	Sends a client disconnected message to the server. This should always be the last action taken by the client before closing the TCP/IP connection.

Receiving Commands

To receive a command, use the *Read* command exposed by the NuklearTCP assembly.

```
public static object Read(NetworkStream stream, out NuklearCommands cmd)
```

The object returned is the data object of the command passed to *cmd*. The client's **NetworkStream** should be passed in as the stream. Use the cmd to determine which data object to use as the cast for the return object. And take action as needed.

Commands to Client from Server

NuklearCommands	Data Object	Meaning
ConnectionResult	ConnectionData	Status of the connection attempt. (Incl. invalid log data, banned, etc)
EnableKeyInputCommand	EnableData	Enables or disables single keystroke input for the client (a la GetKey)
EnableTextInputCommand	EnableData	Enables or disables single keystroke
DisplayTextCommand	TextData	Instructs the client to display a line of text. NuklearCode formatting codes are processed by the client before displaying.
DisplayFileCommand	FileDisplayData	Displays the contents of a file or part of a file.

		NuklearCode formatting codes are processed by the client before displaying.
DisconnectCommand	`null`	Instructs the client that the game session has concluded.

NuklearCommands Data Objects

This section details the data object classes exposed by the NuklearTCP assembly.

LoginData

Login data stores the player's login name and password for transmission to the server.

Parameter	Type	Meaning
LoginName	String	Login name of the player
Password	String	Password of the player. (Possibly encrypted, based on final design decision)

KeyData

Key data stores the value of the last keystroke made by the player. (Possible only after an EnableKeyInputCommand from the server)

Parameter	Type	Meaning
Key	char	Value of the key pressed.

TextData

TextData contains a string of text data. This could be from an input or for display.

Parameter	Type	Meaning
Text	String	The string of text.

ConnectionData

ConnectionData contains the status of the connection attempt.

Parameter	Type	Meaning
Status	ConnectionStatus	Enumeration of possible connection results. See

		NuklearTCP documentation for more details.
Text	String	Additional message information.

EnableData
KeyEnableData contains a bool indicating whether or not a key input will be accepted.

Parameter	Type	Meaning
Enabled	bool	If true, enable a single key or text line input. Only one key or text line input shall be accepted per enabling. If false, any current key or input input enabling is disabled.

FileDisplayData
Login data stores the player's login name and password for transmission to the server.

Parameter	Type	Meaning
FileName	String	Filename containing the text (may include ANSI codes) to display.
SectionName	String	Subsection of the file to display. If the subsection name were to be VIOLET the section of the file to display is the section beginning with the line after @#VIOLET and ending with the last character before the next @# combination.

Communication Logic Execution
Upon invocation of its Execute method, the CommunicationManager shall send a LoginCommand to the server using its login and password properties to send the player's login name and password. If the login is successful, it will transition into the main execution loop. If unsuccessful, the connection result shall be displayer, along with any additional message contained in the result data, and execution shall end.

The main execution loop will display text and data when any of the display commands are received. This shall be done as soon as the command is received. Data to be displayed shall be done in the manner described by the descriptions shown in the *Communication* section of this document.

Upon receiving an EnableKeyInputCommand with the Enabled property set true, the system shall accept one keystroke from the user, display it on the s screen, and continue operation. While waiting for a keypress, all display commands received shall still be processed. If an EnableKeyInputCommand with the Enabled property set false is received, any keypress shall be discarded and not displayed.

Upon receiving an EnableTextInputCommand with the Enabled property set true, the system shall accept one line of text input from the user, display it on the s screen, and continue operation. While waiting for text input, no display commands received shall still be processed. If an EnableTextInputCommand with the Enabled property set false is received, the resulting text input shall be cancelled.

Upon receiving a Disconnect command from the server, the method shall end execution.