Informal establishment JaggedStudio

E-mail: Jagged.N@yandex.ru

Steam: https://steamcommunity.com/id/JaggedNel/

REFERENCE GUIDE FOR THE IN-GAME SOFTWARE PACKAGE

DEMO VERSION

MACHINE TRANSLATION IS USED

Developer: JaggedNel

Date of start developing: 23.12.2019

Date on which the content of the manual is relevant: 18.09.2020

Actual version Nelbrus: 0.4.0 [18.09.2020]

Saint-Petersburg 2019-2020

CONTENT

INTRODUTION

The use of in-game programs in the game Space Engineers often becomes the solution of applied problems of all possible spectrum in the implementation of various projects of all levels. However, systems may use inconvenient or inefficient methods of configuration and operation. A single project can use a variety of programs, which, if necessary, are time consuming to complete in one programmable block.

In this regard, getting an effective tool for developing and interacting with application systems is one of the most promising tasks. The Nelbrus system is designed to be such a tool.

This set of documentation was created in order to familiarize users of the complex with its functionality in detail, to help in the development of additional components and complexes for the system. The manual implies that the user has basic knowledge about in-game scripts.

Please report any errors or shortcomings in the content or in OS through the appropriate discussions on the work pages in the Workshop[2] or repository[3].

DEFINITIONS

Component – a set of commands considered as a single unit that performs a complete function and used independently or as part of a complex.

Package (complex) – a set of commands consisting of two or more components or complexes that perform interrelated functions, and used independently or as part of another complex.

Core – a package of the Nelbrus system that does not include third-party add-ons, subprograms, etc.

Mother block – programmable block with the Nelbrus system.

Program – a complete list of computer instructions and data recorded in the mother block.

Subprogram – a software package or component that is managed by Nelbrus as a part of the program. In a program the subprogram class is *SubP*.

Solution – program that contains the core along with any set of components, packages, or subprograms.

Tick – the minimum in-game unit of time equal to 1/60 of a real second at a simulation speed of 1.00.

Command interface - Командный интерфейс – OS and subprograms components for user interaction with the available subprogram functionality.

Custom action – an event called by a subprogram with specified frequency or delayed for a certain period of time and executed once.

1 Description of the software package

System NELBRUS should be considered as an operating system designed for use in the in-game scripting environment of the game Space Engineers, which performs the following main functions:

1. Providing of the environment for the operation of application subprograms;

1.1. The number of different subprograms used in a single program is limited by the length of the program expressed in characters (100 000 characters by default).

1.2. The number of subprograms running at a time is not limited by software.

2. Providing user access to the resources and functionality of th system and subprograms by:

2.1. The basic command interface for interaction with, which is offered for support by subprograms of any type and purpose.

2.2. Other types of interfaces embedded in subprograms or added to the program in other ways.

The goal of developing the system is to get unified methods for developing and managing in-game software projects.

1.1 Structure of the system`s program code

Figure 1 shows a diagram of the overall system structure.

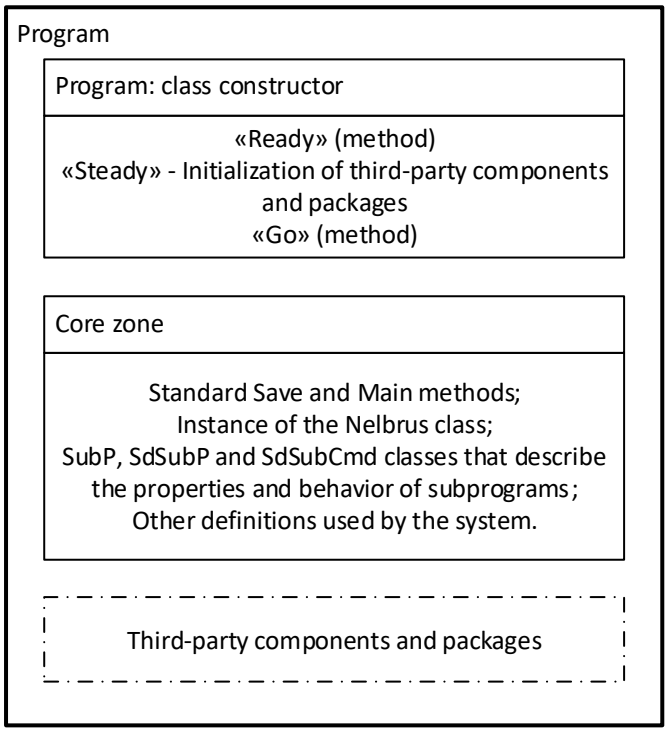

Figure 1 – Solution structure

The program consists of the following blocks:

- The standard program constructor that is called when the program is compiled. The instructions executed here are divided into 3 stages of preparation that are executed once after the program is compiled: "*Ready*, *Steady*, *Go*":

- *Ready* performs the initial configuration of the OS without which it is impossible to perform other stages. Represents the method being called.

- *Steady* initializes third-party components and packages (subprograms) in the OS scope. Is a group of methods (SetEchoCtrl, ISP).

- *Go* completes the OS configuration and puts it in operating mode. Be a method.

- System core consisting: standard script methods, parent subprograms classes, and other system components.

- Definitions of third-party subprograms, packages and components. However, they must be entered in the OS scope at the *Steady* stage in the program constructor.

2 Install and update

There are two main sources for obtaining software components and complexes:

1. Steam Workshop;
2. Repository.

System Nelbrus available from both sources:

- by subscribing to the work of the desired version in the Steam Workshop from JaggedNel[4] and then partially or completely inserting the program code into the programmable block;

- partial or complete copying of the program code from the JaggedNel repository[3], where available versions of the package and documentation are stored.

Always read the author`s installation or upgrade recommendation carefully before installing any software. General installation and upgrade instructions are provided below.

2.1 General core install

For any purpose, a general installation of the system core is possible in the most appropriate way with full copying of the solution code to the programmable block. After that, the complex will be available for use.

2.2 Core update

To update the program core, you must completely replace the code of the outdated system core with the new one. The core is highlighted in the code with keywords: «`#region Core zone`» before it and «`#endregion Core zone`» after.

2.3 Subprograms install

To install a subprogram, its code is added at the end of the program code. In the *Steady* zone of the *Program* constructor, the subprogram class must be initialized to enter the OS visibility zone. Initialization is performed using the ISP method from the OS core as follows:

```
OS.ISP(new JNew());
```

3 Operation of the complex

3.1 Command interface

CI executes system commands and subprograms commands from the subprogram registry that supports the command interface. Commands are entered via:

- Run mother block with argument.

The string containing the command must start with the command start character (/). Then the name of the command to be executed, and, if necessary, the arguments provided by the command syntax are separated by a space. So the general view of the commands is as follows:

/command_name

/command_mame argument1 argument2 … argumentN

If one of the arguments gas spaces in its value, then to avoid separating it, you should add a single quotation mark (’) before and after it. In addition, the last argument that starts with the bond symbol but does not end with it will be considered a bond. For example there command with only one argument:

/command_name ’single argument’

/command_name argument ’single argument

Each subprogram that supports command interface has a basic command for getting *help*. Supported syntax for the *help* command:

/help – execute a command without arguments shows a list of available commands in the registry;

/help command_name – execute a command with argument from the registry to get detailed information about it.

IMPORTANT: All commands you enter are executed from the operating system registry. To execute a command from the command registry of another subprogram, use the *sp* command.

The list of available system commands is given at the end of the manual.

4 Subprogram developing

Project development using the Nelbrus OS is designed to reduce labor costs by defining the functionality required for most components and packages in advance.

Figure 2 shows the components that make up the core. Arrows indicate the direction of class inheritance.

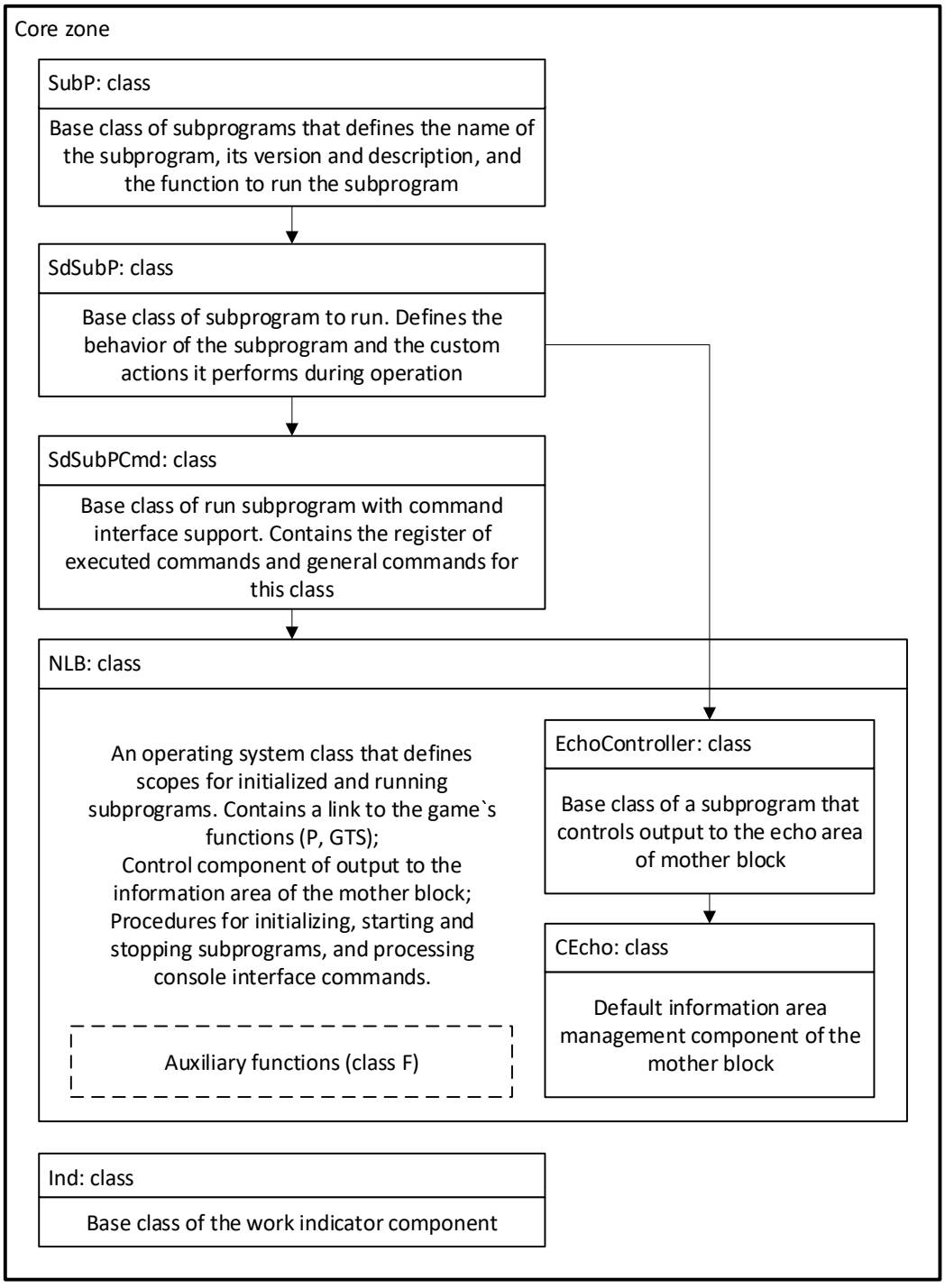

Figure 2 – Core system Nelbrus structure

4.1 Basic definition of a subprogram

The recommended program code that defines the subprogram is shown below:

```
1   class JNew : SubP
2   {
3       public JNew() : base("JNew", new MyVersion(1, 0)) { }
4
5       public override SdSubP Start(ushort id) { return new TP(id, this); }
6
7       class TP : SdSubP
8       {
9           public TP(ushort id, SubP p) : base(id, p)
10          {
11
12          }
13      }
14  }
```

The *JNew* class defines a subprogram that is initialized in the system. Its constructor without arguments, which calls one of the inherited constructors (from the *SubP* class), is called in the *Ready* zone using the *OS.ISP* method.

The nested class *TP* (*This Program*) is a runnable subprogram body that defines all the functionality of the developed component or complex. The *JNew* class is only responsible for introducing the *TP* class into the OS visibility zone and launching it.

When executing the start command of an initialized subprogram, the redefined *Start* method is called, which should return a reference to the created instance of the *TP* class.

4.2 Subprograms with command interface support

When the *TP* class inherits form the *SdSubPCmd* class, the use of CI elements becomes available. An example of defining such a subroutine is shown below.

```
1   class JNew : SubP
2   {
3       public JNew() : base("JNew", new MyVersion(1, 0)) { }
4
5       public override SdSubP Start(ushort id) { return new TP(id, this); }
6
7       class TP : SdSubPCmd
8       {
9           public TP(ushort id, SubP p) : base(id, p)
10          {
11              SetCmd("CommandName", new Cmd(CmdAction));
12          }
13      }
14  }
```

In the constructor of the *TP* class, subprogram commands are initialized using the *SetCmd* method. Note that the *help* command is defined in the constructor of the *SdSubPCmd* and is already initialized.

4.3 Custom actions: frequency and deferring

Each running subprogram defined by the *SdSubP* class perform various actions at a specified frequency. The *EAct* (*Every tick actions*) delegate (allocated separately for optimization purposes) and the Acts delegate collection are responsible for storage.

To control periodic actions, the subprogram defines a variable of type *CAct* (*Custom action*) and uses the following functions:

- Create: *AddAct*
- Delete: *RemAct*
- Change: *ChaAct*

The periodic action will be performed every n ticks.

You can also use actions that are deferred for n ticks, controlled by methods:

- Create: *AddDefA*
- Delete: *RemDefA*

A general view of a subprogram that uses custom actions of a given frequency is shown below:

```
1   class JNew : SubP
2   {
3       public JNew() : base("JNew", new MyVersion(1, 0)) { }
4
5       public override SdSubP Start(ushort id) { return new TP(id, this); }
6
7       class TP : SdSubP
8       {
9           CAct MA;
10          public TP(ushort id, SubP p) : base(id, p)
11          {
12              AddAct(ref MA, Main, 2);
13          }
14
15          void Main()
16          {
17
18          }
19      }
20  }
```

LIST OF AVAILABLE SYSTEM COMMANDS

The arguments of the commands, framed by the symbols <…> are mandatory. Arguments framed with […] characters are optional.

Command	Description	Details
start	Start initialized subprogram by id.	Example: */start <id>* Id verification is performed using the "/isp " command.
stop	Stop runned subprogram by id.	Example: */stop <id>* Id verification is performed using the "/isp " command.
sp	View runned subprograms or execute command by subprogram.	Example: */sp* – (Without arguments) View runned subprograms list; */sp <id>* – View information obout runned subprogram by *id*; */sp <id> <command> [arguments]* – Execute *command* with *arguments* by subprogram with *id*.
isp	View initialized subprograms.	Example: */isp* – (Without arguments) View initialized subprograms list; */isp <id>* - View information about initialized subprogram by *id*.
clr	Clear command interface.	Example: */clr*

USEFUL LINKS

1. Steam-profile JaggedNel: https://steamcommunity.com/id/JaggedNel/
2. Actual Nelbrus version in Steam Workshop: https://steamcommunity.com/sharedfiles/filedetails/?id=2014553432
3. Repository Nelbrus: https://github.com/JaggedNel/Nelbrus
4. Steam Workshop JaggedNel: https://steamcommunity.com/id/JaggedNel/myworkshopfiles/?appid=244850