

Неформальное заведение JaggedStudio

E-mail: Jagged.N@yandex.ru

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

## **СПРАВОЧНОЕ ПОСОБИЕ ВНУТРИИГРОВОГО ПРОГРАММНОГО КОМПЛЕКСА**



**ПРЕДВАРИТЕЛЬНАЯ ВЕРСИЯ**

Разработчик: JaggedNel

Дата начала реализации проекта: 23.12.2019

Дата, на которую актуально содержание пособия: 18.09.2020

Актуальная версия Nelbrus: 0.4.0 [18.09.2020]

Санкт-Петербург 2019-2020

## **СОДЕРЖАНИЕ**

ВВЕДЕНИЕ.....	3
ОПРЕДЕЛЕНИЯ.....	4
1    Описание программного комплекса.....	5
1.1  Структура программного кода системы .....	6
2    Установка и обновление .....	8
2.1  Чистая установка ядра .....	8
2.2  Обновление ядра.....	8
2.3  Установка подпрограмм .....	8
3    Работа комплекса .....	9
3.1  Командный интерфейс.....	9
4    Разработка подпрограмм .....	10
4.1  Базовое определение подпрограммы .....	11
4.2  Подпрограммы с поддержкой командного интерфейса.....	11
4.3  Периодические и отложенные действия подпрограмм .....	12
ПЕРЕЧЕНЬ ДОСТУПНЫХ КОМАНД СИСТЕМЫ.....	14
ПОЛЕЗНЫЕ ССЫЛКИ.....	15

## **ВВЕДЕНИЕ**

Использование внутриигровых программ в игре Space Engineers часто становится решением прикладных задач всевозможного спектра при реализации различных проектов любых уровней. Однако, системы могут использовать неудобные или неэффективные методы настройки и работы. При этом в отдельно взятом проекте может использоваться множество программ, которые, при необходимости, трудоёмко укомплектовать в один программируемый блок.

В связи с этим, получение инструмента эффективного для разработки и взаимодействия с прикладными системами представляется одной из перспективнейших задач. Система Nelbrus призвана стать таким инструментом.

Данный свод документации создан с целью подробно ознакомить пользователей комплекса с его функционалом, помочь при разработке дополнительных компонентов и комплексов для системы. Пособие подразумевает наличие у его пользователя наличие основных знаний о внутриигровых скриптах. Если их нет, рекомендуется ознакомиться с внутриигровым программированием перед началом ознакомления.

О любых ошибках и недочётах содержания или работы системы прошу сообщать через соответствующие обсуждения на страницах работы в Workshop[2] или репозитории[3].

## ОПРЕДЕЛЕНИЯ

Компонент – набор команд, рассматриваемый как единое целое, выполняющий законченную функцию и применяемый самостоятельно или в составе комплекса.

Комплекс – набор команд, состоящий из двух или более компонентов или комплексов, выполняющих взаимосвязанные функции, и применяемый самостоятельно или в составе другого комплекса.

Ядро – комплекс системы Nelbrus, не включающий сторонние дополнения, подпрограммы и т.п.

Материнский блок – программируемый блок с системой Nelbrus.

Программа – полный перечень компьютерных инструкций и данных, записанных в программном блоке.

Подпрограмма – программный комплекс или компонент, под управлением Nelbrus в составе программы. В программе родителем класса подпрограммы является класс *SubP*.

Сборка – программа, содержащая ядро вместе с любым набором компонентов, комплексов или подпрограмм.

Тик – минимальная внутриигровая единица времени, равная 1/60 реальной секунды при скорости симуляции 1,00.

Командный интерфейс – компоненты ОС и подпрограмм для взаимодействия пользователя с доступным для этого функционалом подпрограмм.

## **1 Описание программного комплекса**

Систему NELBRUS следует рассматривать как операционную систему, созданную для использования в среде внутриигровых скриптов игры Space Engineers, выполняющую следующие основные функции:

1. Организация среды для функционирования прикладных подпрограмм;

1.1. Количество различных подпрограмм, используемых в одной программе ограничено длиной программы выраженной в символах (100 000 символов по умолчанию).

1.2. Количество подпрограмм, запущенных одновременно программно не ограничено.

2. Обеспечение доступа пользователя к ресурсам и функционалу системы и подпрограмм посредством:

2.1. Базового командного интерфейса взаимодействия NELBRUS, предлагаемого к поддержке подпрограммами любых типов и назначений.

2.2. Другими типами интерфейсов, встроенных в подпрограммы или добавленные в программу иными способами.

Цель разработки системы – получение унифицированных методов разработки и управления внутриигровыми программными проектами.

## 1.1 Структура программного кода системы

На рисунке 1 приведена схема общего устройства сборки системы.

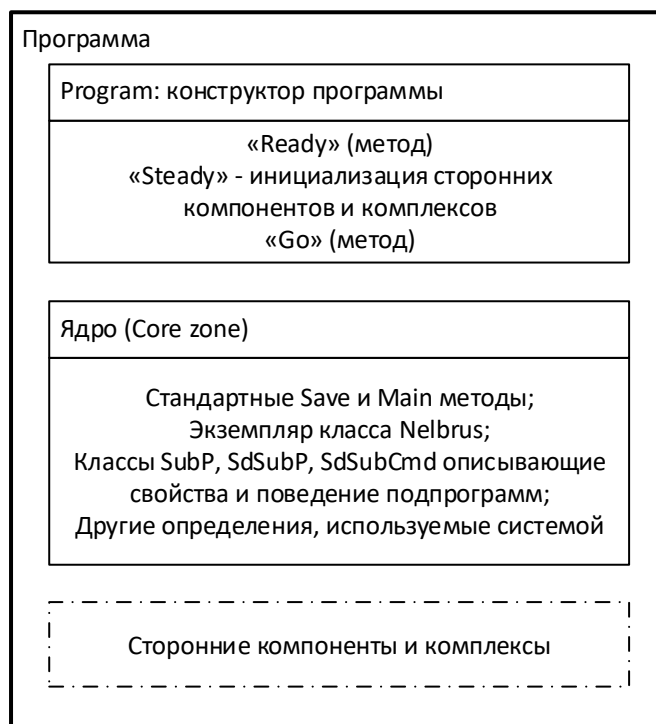


Рисунок 1 – Структура сборки

Программа состоит из следующих блоков:

- Стандартного конструктора *Program*, вызываемого при компиляции программы. Инструкции выполняемые здесь разделены на выполняющиеся один раз после компиляции программы 3 этапа подготовки «Ready, Steady, Go»:

- *Ready* производит первоначальную настройку ОС без чего невозможна выполнения других этапов. Представляет собой вызываемый метод.

- *Steady* инициализирует сторонние компоненты и комплексы (подпрограммы) в области видимости ОС. Является группой методов (SetEchoCtrl, ISP).

- *Go* завершает настройку ОС и переводит её в рабочий режим. Является методом.

- Ядра системы, содержащего: стандартные методы скрипта, родительские классы подпрограмм и другие компоненты системы.

- Определений сторонних подпрограмм, комплексов и компонентов. При этом они должны вноситься в область видимости ОС на этапе Steady в конструкторе программы.

## **2 Установка и обновление**

Есть два основных источника получения программных компонентов и комплексов:

1. Steam Workshop;
2. Репозитории.

Система Nelbrus доступна через оба источника:

- через подписку на сборку нужной версии в Steam Workshop от JaggedNel [4] и последующую частичную или полную вставку кода программы в программируемый блок;
- частичное или полное копирование кода программы из репозитория JaggedNel [3], где хранятся доступные версии комплекса и документация.

Всегда перед установкой любого программного обеспечения тщательно ознакомьтесь с рекомендациями автора по установке или обновлению. Ниже приведены общие инструкции по установке и обновлению.

### **2.1 Чистая установка ядра**

Для любых целей возможна чистая установка ядра системы наиболее подходящим способом с полным копированием кода сборки в программируемый блок. После этого комплекс будет доступен для использования.

### **2.2 Обновление ядра**

Для обновления ядра программы необходимо полностью заменить код устаревшего ядра системы на новое. Ядро выделено в коде ключевыми фразами: «`#region Core zone`» перед ним и «`#endregion Core zone`» после.

### **2.3 Установка подпрограмм**

Для установки подпрограммы её код добавляется в конце кода программы. В зоне *Steady* конструктора *Program* класс подпрограммы обязательно должен инициализироваться для введения в зону видимости ОС. Инициализация производится методом ISP из ядра ОС следующим образом:

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



### **3 Работа комплекса**

#### **3.1 Командный интерфейс**

КИ производит выполнение команды системы и команд подпрограмм из реестра подпрограмм с поддержкой командного интерфейса. Ввод команд производится через:

- Обработку аргумента, с которым вызывается материнский блок.

Строка, содержащая команду, должна начинаться с символа начала команды (/). Затем название команды, которая будет выполнена, и, при необходимости, через пробел перечислены аргументы, предусмотренные синтаксисом команды. Таким образом общий вид команд имеет вид:

*/command\_name*

*/command\_name argument1 argument2 ... argumentN*

В случае если один из аргументов имеет в своём значении пробелы, то во избежание его разделения, перед ним и после него следует добавить символ скрепления - одинарную кавычку ('). Кроме того, последний аргумент, начавшийся символом скрепления, но не закончившийся им будет считаться скреплённым. Например:

*/command\_name 'single argument'*

*/command\_name argument 'single argument'*

Каждая подпрограмма с поддержкой консольных команд имеет базовую команду получения помощи *help*. Поддерживаемый синтаксис команды *help*:

*/help* – вызов команды без аргументов показывает список доступных команд в реестре;

*/help command\_name* – вызов команды с аргументом-именем команды из реестра для получения подробной информации о ней.

Все вводимые команды выполняются из реестра операционной системы. Для выполнения команды из реестра команд другой подпрограммы следует пользоваться командой *sr*.

Перечень доступных команд системы приведён в конце пособия.

## 4 Разработка подпрограмм

Разработка проектов с использованием ОС NELBRUS призвана сократить трудозатраты, заранее определяя необходимый для большинства компонентов и комплексов функционал.

На рисунке 2 приведены комплексы и компоненты, составляющие ядро. Стрелками указано направление наследования классов.

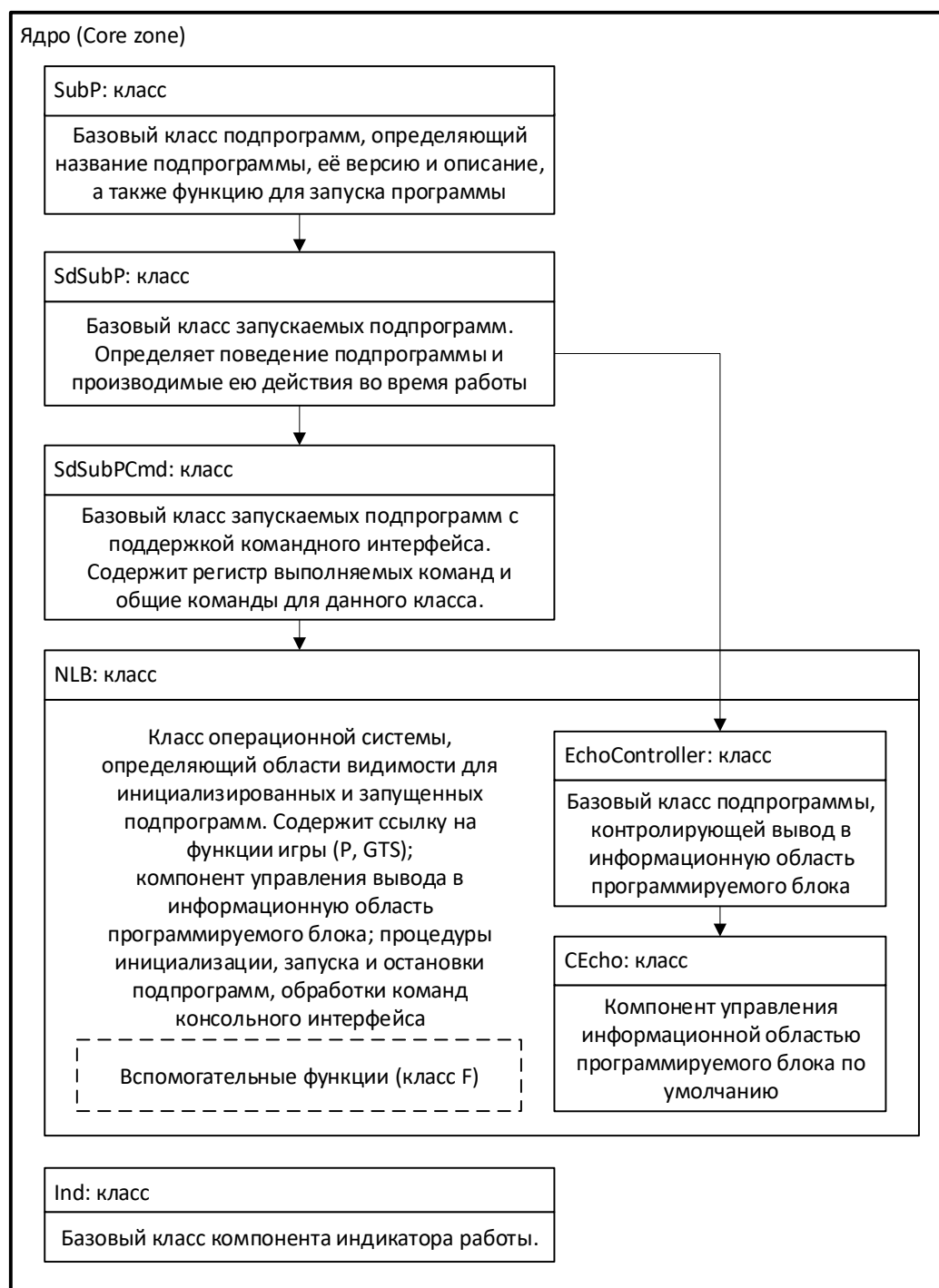


Рисунок 2 – Архитектура ядра системы Nelbrus

## 4.1 Базовое определение подпрограммы

Ниже представлен программный код, определяющий подпрограмму:

```
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 }
```

Класс *JNew* определяет инициализируемую в системе подпрограмму. Его конструктор без аргументов, вызывающий один из наследуемых (от класса *SubP*) конструкторов, вызывается в зоне *Ready* при помощи метода *OS.ISP*.

Вложенный класс *TP* (*This Program*) является запускаемым телом подпрограммы, который определяет весь функционал, разрабатываемого компонента или комплекса. Класс *JNew* при этом отвечает только за введение класса *TP* в зону видимости ОС и его запуск.

При выполнении команды запуска инициализированной подпрограммы вызывается переопределённый метод *Start*, который должен вернуть ссылку на созданный экземпляр класса *TP*. Если необходимо, чтобы подпрограмма могла быть запущена только в одном экземпляре, то предлагается переопределить метод *Start* следующим образом:

```
1 public override SdSubP Start(ushort id)
2 {
3     return OS.CSP<TP>() ? null : new TP(id, this);
4 }
```

При этом, при попытке повторного запуска уже работающей подпрограммы, будет возвращено значение *null* и ОС проигнорирует попытку.

## 4.2 Подпрограммы с поддержкой командного интерфейса

При наследовании класса *TP* от класса *SdSubPCmd*, становятся доступно использование элементов КИ. Ниже приведён пример определения такой подпрограммы.

```
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 }
```

В конструкторе класса *TP* таким образом инициализируются команды подпрограммы методом *SetCmd*. Обратите внимание, что команда *help* определяется в конструкторе наследуемого класса *SdSubPCmd* и уже инициализирована.

### 4.3 Периодические и отложенные действия подпрограмм

Каждая запущенная подпрограмма, определяемая классом *SdSubP*, может выполнять различные действия с заданной частотой. За хранение отвечает делегат (void Act) *EAct* (*Every tick actions*) (выделен отдельно с целью оптимизации) и коллекция делегатов *Acts*.

Для управления периодическими действиями в подпрограмме определяется переменная типа *CAct* (*Custom Action*) и используются следующие функции:

- Создание: *AddAct*
- Удаление: *RemAct*
- Изменение: *ChaAct*

Периодическое действие будет выполняться каждые *n* тиков.

Также доступно использование действий, отложенных на *n* тиков, управляемых методами:

- Создание: *AddDefA*
- Удаление: *RemDefA*

Общий вид подпрограммы, использующей действия заданной частоты, приведён ниже:

```
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 }
```

## ПЕРЕЧЕНЬ ДОСТУПНЫХ КОМАНД СИСТЕМЫ

Аргументы команд, обрамлённые символами `<...>` являются обязательными. Аргументы, обрамлённые символами `[...]` являются дополнительными

Команда	Описание	Детали
start	Запускает инициализированную подпрограмму по id.	Пример: <code>/start &lt;id&gt;</code> Проверка id производится по команде «/isp».
stop	Останавливает запущенную подпрограмму по id.	Пример: <code>/stop &lt;id&gt;</code> Проверка id производится по команде «/sp».
sp	Просмотр запущенных подпрограмм или выполнение команды подпрограммой.	Пример: <code>/sp</code> – (Без аргументов) Просмотреть список запущенных подпрограмм; <code>/sp &lt;id&gt;</code> – Просмотреть информацию о запущенной подпрограмме по id; <code>/sp &lt;id&gt; &lt;command&gt; [arguments]</code> – Выполнение подпрограммой с id команды command и аргументами arguments.
isp	Просмотр инициализированных подпрограмм.	Пример: <code>/isp</code> – (Без аргументов) Просмотр списка инициализированных подпрограмм; <code>/isp &lt;id&gt;</code> - Просмотреть информацию об инициализированной подпрограмме по id.
clr	Очистить командный интерфейс.	Пример: <code>/clr</code>

## ПОЛЕЗНЫЕ ССЫЛКИ

1. Steam-профиль JaggedNel: <https://steamcommunity.com/id/JaggedNel/>
2. Актуальная версия Nelbrus в Steam Workshop:  
<https://steamcommunity.com/sharedfiles/filedetails/?id=2014553432>
3. Репозиторий Nelbrus: <https://github.com/JaggedNel/Nelbrus>
4. Мастерская Steam JaggedNel:  
<https://steamcommunity.com/id/JaggedNel/myworkshopfiles/?appid=244850>