Энергостенд построен с использованием WCF (Windows Communication Foundation – программный фреймворк, используемый для обмена данными между приложениями и входящий в состав .NET Framework). Один из сервисов настроен на обработку запросов в формате JSON. URL для запросов JSON: http://localhost:8004/JSONGreenCity. Данная конфигурация прописана в файле *GreenCity.DataService.exe.config*, размещенный в папке *C:\Program Files\Novator Ltd\GreenCityService* (путь, куда был установлен сервис).

Ниже приведено содержание файла конфигурации сервиса данных (*GreenCity.DataService.exe.config*):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.2" />
    </startup>

  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="bindConfig">
          <security mode="None" />
        </binding>
      </netTcpBinding>

      <webHttpBinding>
        <binding name="webbindConfig">
          <security mode="None" />
        </binding>
      </webHttpBinding>
    </bindings>

    <services>
      <service name="Novator.GreenCity.Service.GreenCityService">
        <!--host>
          <baseAddresses>
            <add baseAddress = "http://localhost:8833/Novator/GreenCity/" />
          </baseAddresses>
        </host-->

        <endpoint address="net.tcp://localhost:8005/Novator/GreenCity" binding="netTcpBinding"
                  contract="Novator.GreenCity.Service.IGreenCityService"
bindingConfiguration="bindConfig">
        </endpoint>

        <!--endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/-->
      </service>
      <service name="Novator.GreenCity.Service.JSONGreenCityService">
        <endpoint address="http://localhost:8004/JSONGreenCity"
binding="webHttpBinding"
```

```
            contract="Novator.GreenCity.Service.IJSONGreenCityService"
            bindingConfiguration="webbindConfig" >
        <!--behaviorConfiguration="webScriptEnablingBehavior"-->
      </endpoint>
      <!--endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" /-->
      <!--host>
      <baseAddresses>
        <add
baseAddress="http://localhost:8834/Novator.GreenCity.Service/JSONGreenCity/" />
      </baseAddresses>
    </host-->
    </service>
  </services>

  <behaviors>
    <serviceBehaviors>
      <!--behavior name="webScriptEnablingBehavior">
        <webHttp defaultOutgoingResponseFormat="Json" />
      </behavior-->
      <!--behavior>
        <serviceMetadata httpGetEnabled="True" httpsGetEnabled="True"/>
        <serviceDebug includeExceptionDetailInFaults="False" />
      </behavior-->
    </serviceBehaviors>
  </behaviors>
 </system.serviceModel>
</configuration>
```

Конфигурация сервиса, обслуживающего запросы JSON, выделена цветом.

Для проверки можете ввести строку запроса в браузере, например, http://localhost:8004/JSONGreenCity/GetConfiguration, отклик будет иметь следующий вид:

{"GameItteration":500,"GameSpeed":20,"SplitBalancing":false}

Необходимо иметь ввиду – кросдоменные запросы (CORS) из браузера обслуживаться не будут.

Сервис данных “Зеленного города” ведет запись протокола ошибок и предупреждений в текстовый файл *datasrv.log.text*, расположенный в *C:\ProgramData\WinDataService*.

Я не могу сейчас проверить каждый запрос, но, если мне не изменяет память, все точки входа могут работать по методам запроса GET и POST. Тем не менее, все запросы, которые должны вернуть данные, используйте метод GET. Для запросов управления без параметров, можно также использовать метод GET. Для запросов управления передающих параметры по шаблону, используйте метод GET. Для запросов управления, передающих объект, используйте метод POST и передаваемый объект должен быть

правильным JSON-объектом (например, сформированный непосредственно средствами Javascript или расширением Javascript – axios.js).

Ниже приведены точки входа для сервиса JSON:

```
    [ServiceContract]
    public interface IJSONGreenCityService
    {
        #region настройка
        [OperationContract]
        [WebGet(ResponseFormat=WebMessageFormat.Json,
                RequestFormat=WebMessageFormat.Json)]
        ModelConfiguration GetConfiguration();

        [OperationContract(IsOneWay = true)]
        [WebInvoke(RequestFormat = WebMessageFormat.Json,
                   ResponseFormat = WebMessageFormat.Json,
                   Method = "*",
                   BodyStyle = WebMessageBodyStyle.Bare)]
        void SetConfiguration(ModelConfiguration newcfg);
        #endregion

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        bool Start();

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        bool Stop();

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        ModelStrobe ModelStatus();

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        ModelTreeResult ModelTree();

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        PowerObjectIdentifier[] GetPowerObjects();

        [OperationContract]
        [WebInvoke(RequestFormat = WebMessageFormat.Json,
                   ResponseFormat = WebMessageFormat.Json,
                   Method = "*",
                   BodyStyle = WebMessageBodyStyle.Bare)]
```

```
        bool UpdatePowerObject(PowerObjectIdentifier chobj);

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        PowerObjectIdentifier GetPowerObject(string key);

        [OperationContract(IsOneWay=true)]
        [WebInvoke(RequestFormat = WebMessageFormat.Json,
                   Method = "*",
                   BodyStyle = WebMessageBodyStyle.Wrapped,
                   UriTemplate =
"TurnOn?key={key}&soketnum={soketnum}")]
        void TurnOn(string key, int soketnum);

        [OperationContract(IsOneWay = true)]
        [WebInvoke(RequestFormat = WebMessageFormat.Json,
                   Method = "*",
                   BodyStyle = WebMessageBodyStyle.Wrapped,
                   UriTemplate =
"TurnOff?key={key}&soketnum={soketnum}")]
        void TurnOff(string key, int soketnum);

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        HardDevicesSettings GetPlcDevices();

        [OperationContract(IsOneWay = true)]
        [WebInvoke(RequestFormat = WebMessageFormat.Json,
                   ResponseFormat = WebMessageFormat.Json,
                   Method = "*",
                   BodyStyle = WebMessageBodyStyle.Bare)]
        void SetPlcDevices(HardDevicesSettings data);

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        DevicesServiceConfiguration GetPlcServiceCfg();

        [OperationContract]
        [WebInvoke(RequestFormat = WebMessageFormat.Json,
                   ResponseFormat = WebMessageFormat.Json,
                   Method = "*",
                   BodyStyle = WebMessageBodyStyle.Bare)]
        bool SetPlcServiceCfg(DevicesServiceConfiguration cfg);

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json,
                RequestFormat = WebMessageFormat.Json)]
        string[] PlcPorts();
```

```
    }
```

GetConfiguration

Возвращает настройки сервиса модели.

- GameSpeed – скорость игры (в сек.): определяет соотношение времени системы и виртуального времени игры ($t_{системы, сек.} = t_{игры, сек.}$). Минимальное значение 2 сек., максимальное – 600 сек. Тип данных: беззнаковое 16-ти разрядное целое число.
- GameItteration – определяет продолжительность “шага” игры в мс. Имеет отношение к производительности сервиса моделирования, т.е. показывает как быстро модель будет реагировать на изменения. Минимальное значение 200 мс., максимальное – 2000 мс. Тип данных: 64-разрядное целое число.
- SplitBalancing – определяет как будет рассчитывать нагрузка и генерация – общая для всех линий (false) и раздельно по каждой линии (true).Тип данных: bool (логический тип данных – false (0) или true (1)).

Метод запроса GET. Запрос выполняется без параметров, результат запроса – text/json. Вид запроса: http://localhost:8004/JSONGreenCity/GetConfiguration. Вид ответа: **{"GameItteration":500,"GameSpeed":20,"SplitBalancing":false}**.

SetConfiguration

Устанавливает новые настройки сервиса модели. Вид передаваемого JSON-объекта:

```
{
    "GameItteration": 500,
    "GameSpeed": 20,
    "SplitBalancing": false
}
```

Метод запроса POST. JSON-объект передается как параметр *newcfg*. Запрос ничего не возвращает клиенту (отправили и забыли).

Start

Запуск модели. Метод GET. Запрос выполняется без параметров, результат запроса – всегда значение ***true***. Вид запроса: http://localhost:8004/JSONGreenCity/Start.

Stop

Останов модели. Метод GET. Запрос выполняется без параметров, результат запроса – всегда значение ***true***. Вид запроса: http://localhost:8004/JSONGreenCity/Stop.

ModelStatus

Возвращает состояние модели на время обработки запроса. Метод GET. Запрос выполняется без параметров. Результат запроса – text/json: **{"EndTime": "\/Date(1593235782656+0700)\/","GameTime":"PT4M31.674S","Running":false, "StartTime":"\/Date(1593235768889+0700)\/"}**. Вид запроса: http://localhost:8004/JSON-GreenCity/ModelStatus.

- EndTime – временной строб завершения игры. Если игра активна (запущена) – EndtTime и StartTime будут идентичны. Тип данных: DateTime (C# JSON). При использовании средств Javascript для восстановления объекта, необходимо привести данную строку к синтаксису типа Date Javascript JSON.
- GameTime – временной строб текущего времени игры. Тип данных: TimeSpan (C# JSON).
- Running – состояние выполнения игры – *true*, игра запущена (активна); *false*, игра остановлена. Тип данных: bool (логический тип данных – false (0) или true (1)).
- StartTime – временной строб начала игры. Тип данных: DateTime (C# JSON). При использовании средств Javascript для восстановления объекта, необходимо привести данную строку к синтаксису типа Date Javascript JSON.

ModelTree

Возвращает текущую структуру дерева модели. Метод GET. Запрос выполняется без параметров. Результат запроса – text/json: **{"ElementsOK":false,"Lamp1val":0,"Lamp2val":0, "RootNode":{"GeneratedPower":0,"Lines":[{"Childs":null,"GeneratedPower":0, "ID":"П1","IsON":false,"ObjectType":1,"Power":0,"RequiredPower":0,"SocketNum":-1}, {"Childs":null,"GeneratedPower":0,"ID":"П2","IsON":false,"ObjectType":1,"Power":0, "RequiredPower":0,"SocketNum":-1},{"Childs":null,"GeneratedPower":0, "ID":"П3","IsON":false,"ObjectType":1,"Power":0,"RequiredPower":0,"SocketNum":-1}], "RequiredPower":0,"Stations":[null,null,null,null,null,null]},"TreeOK":false,"Windval":0}**. Вид запроса: http://localhost:8004/JSONGreenCity/ModelTree.

Ниже приведены структуры данных в сокращенном виде синтаксиса C#.

Узел начала дерева:

```
public sealed class ModelTreeResult
{
    [DataMember]
    public bool TreeOK { get; set; }

    [DataMember]
    public bool ElementsOK { get; set; }

    [DataMember]
    public ushort Lamp1val { get; set; }

    [DataMember]
    public ushort Lamp2val { get; set; }

    [DataMember]
```

```
        public ushort Windval { get; set; }

        [DataMember]
        public PowerDistributionStation RootNode { get; set; }
    }
```

- TreeOK – успешное выполнение взаимодействия с ПЛК. Тип данных: bool (логический тип данных – false (0) или true (1)).
- ElementsOK – успешное взаимодействие с подчиненными ПЛК. Тип данных: bool (логический тип данных – false (0) или true (1)).
- Lamp1val – значение мощности в процентах, управления первой лампой. Тип данных: беззнаковое 16-ти разрядное целое число.
- Lamp2val – значение мощности в процентах, управления второй лампой. Тип данных: беззнаковое 16-ти разрядное целое число.
- Windval – значение мощности в процентах, управления вентилятором. Тип данных: беззнаковое 16-ти разрядное целое число.
- RootNode – силовая распределительная станция. Тип данных: объект C# *PowerDistributionStation*.

```
    public sealed class PowerDistributionStation
    {
        [DataMember]
        public double GeneratedPower

        [DataMember]
        public double RequiredPower

        [DataMember]
        public PowerLinkObject[] Stations

        [DataMember]
        public PowerLinkObject[] Lines
    }
```

- GeneratedPower – генерируемая мощность в условных единицах. Тип данных: число двойной точности с плавающей точкой.
- RequiredPower – требуемая мощность в условных единицах. Тип данных: число двойной точности с плавающей точкой.
- Stations – подключенные объекты генерации. Тип данных: массив объектов C# *PowerLinkObject*.
- Lines – подключенные объекты транспорта энергии. Тип данных: массив объектов C# *PowerLinkObject*.

```
    public sealed class PowerLinkObject
    {
        [DataMember]
        public bool IsON

        [DataMember]
        public string ID
```

```
        [DataMember]
        public PowerLinkObject[] Childs

        [DataMember]
        public PowerObjectTypes ObjectType

        [DataMember]
        public int SocketNum

        [DataMember]
        public double GeneratedPower

        [DataMember]
        public double RequiredPower

        [DataMember]
        public double Power
    }
```

- IsON – *true* объект/узел включен, *false* объект/узел отключен. Тип данных: bool (логический тип данных – false (0) или true (1)).
- ID – уникальный идентификатор объекта. Тип данных: строка.
- Childs – дочерние объекты узла. Тип данных: массив объектов C# *PowerLinkObject*.
- ObjectType – тип объекта. Тип данных: перечисление – “0” – потребитель, “1” – линия, “2” – генератор, “3” – подстанция.
- SocketNum – номер ввода. Тип данных: целое число.
- GeneratedPower – генерируемая мощность в условных единицах. Тип данных: число двойной точности с плавающей точкой.
- RequiredPower – требуемая мощность в условных единицах. Тип данных: число двойной точности с плавающей точкой.
- Power – статическая мощность в условных единицах. Тип данных: число двойной точности с плавающей точкой.

GetPowerObjects

Возвращает массив доступных силовых объектов, зарегистрированных в системе. Метод GET. Запрос выполняется без параметров. Результат запроса – text/json: **[{"__type": "PowerStation:#Novator.GreenCity.Common","ID":"СБ1","Power":100, "PowerTable":null,"StationType":2,"UseTable":false},{"__type": "PowerStation:#Novator.GreenCity.Common","ID":"СБ2","Power":100, "PowerTable":null,"StationType":2,"UseTable":false},{"__type": "PowerStation:#Novator.GreenCity.Common","ID":"ВГ1","Power":100, "PowerTable":null,"StationType":3,"UseTable":false},{"__type": "PowerStation:#Novator.GreenCity.Common","ID":"ДГ1","Power":100, "PowerTable":null,"StationType":1,"UseTable":false},{"__type": "PowerStation:#Novator.GreenCity.Common","ID":"ДГ2","Power":100, "PowerTable":null,"StationType":1,"UseTable":false},{"__type": "PowerStation:#Novator.GreenCity.Common","ID":"ДГ3","Power":100, "PowerTable":null,"StationType":1,"UseTable":false},{"__type":**

**"PowerStation:#Novator.GreenCity.Common","ID":"АБ1","Power":100,
"PowerTable":null,"StationType":0,"UseTable":false},{"__type":
"PowerStation:#Novator.GreenCity.Common","ID":"АБ2","Power":100,
"PowerTable":null,"StationType":0,"UseTable":false},{"__type":
"PowerStation:#Novator.GreenCity.Common","ID":"АБ3","Power":100,
"PowerTable":null,"StationType":0,"UseTable":false},{"__type":
"PowerStation:#Novator.GreenCity.Common","ID":"АИ1","Power":100,
"PowerTable":null,"StationType":4,"UseTable":false},{"__type":
"PowerStation:#Novator.GreenCity.Common","ID":"АИ2","Power":100,
"PowerTable":null,"StationType":4,"UseTable":false},{"__type":
"PowerConsumer:#Novator.GreenCity.Common","ID":"Микро район №1",
"Consumer":0,"Power":100,"PowerTable":null,"Sockets":1,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common","ID":
"Микро район №2","Consumer":0,"Power":100,"PowerTable":null,"Sockets":1,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common","ID":
"Микро район №3","Consumer":0,"Power":100,"PowerTable":null,"Sockets":1,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common","ID":
"Микро район №4","Consumer":0,"Power":100,"PowerTable":null,"Sockets":1,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common","ID":
"Микро район №5","Consumer":0,"Power":100,"PowerTable":null,"Sockets":1,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common","ID":
"Микро район №6","Consumer":0,"Power":100,"PowerTable":null,"Sockets":1,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common",
"ID":"Больница №1","Consumer":2,"Power":100,"PowerTable":null,"Sockets":2,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common",
"ID":"Больница №2","Consumer":2,"Power":100,"PowerTable":null,"Sockets":2,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common","ID":
"Завод №1","Consumer":1,"Power":100,"PowerTable":null,"Sockets":2,
"UseTable":false},{"__type":"PowerConsumer:#Novator.GreenCity.Common","ID":
"Завод №2","Consumer":1,"Power":100,"PowerTable":null,"Sockets":2,
"UseTable":false},{"__type":"PowerSubstation:#Novator.GreenCity.Common",
"ID":"МП1","Lines":2},{"__type":"PowerSubstation:#Novator.GreenCity.Common",
"ID":"МП2","Lines":2}]**. Вид запроса: http://localhost:8004/JSONGreenCity/GetPowerObjects.

Формируемый список объектов, предоставляет результаты от общего предка. Поле *__type* генерируется системой WCF, для возможности последующего восстановления при работе сервиса (определяет класс восстанавливаемого объекта C#). При формировании данного объекта с последующей передачи его сервису JSON, необходимо сохранить установленное значение поля *__type*, чтобы запрос был обработан без ошибок. Ниже приведены структуры данных в сокращенном виде синтаксиса C#.

```
public abstract class PowerObjectIdentifier
{
    [DataMember]
    public abstract string ID { get; internal set; }
}
```

- ID – уникальный идентификатор объекта. Тип данных: строка.

Производные объекта.

```
public sealed class PowerStation : PowerObjectIdentifier
{
    [DataMember]
    public PowerTimeTable PowerTable

    [DataMember]
    public bool UseTable

    [DataMember]
    public PowerStationTypes StationType

    [DataMember]
    public double Power
}
```

- PowerTable – таблица распределения мощности по временным меткам. Тип данных: объект C# *PowerTimeTable*.
- UseTable – задействовать при моделировании таблицу распределения мощностей (*true* – использовать таблицу, *false* – использовать статическую мощность). Тип данных: bool (логический тип данных – false (0) или true (1)).
- StationType – тип объекта. Тип данных: перечисление – “0” – аккумуляторная, “1” – дизельная, “2” – солнечная, “3” – альтернативная.
- Power – статическая мощность генерации в условных единицах. Тип данных: число двойной точности с плавающей точкой.

```
public sealed class PowerSubstation : PowerObjectIdentifier
{
    [DataMember]
    public int Lines
}
```

- Lines – количество выходных линий. Тип данных: целое число.

```
public sealed class PowerConsumer : PowerObjectIdentifier
{
    [DataMember]
    public PowerTimeTable PowerTable

    [DataMember]
    public bool UseTable

    [DataMember]
    public PowerConsumerTypes Consumer

    [DataMember]
    public int Sockets
```

```
    [DataMember]
    public double Power
}
```

- PowerTable – таблица распределения мощности по временным меткам. Тип данных: объект C# *PowerTimeTable*.
- UseTable – задействовать при моделировании таблицу распределения мощностей (*true* – использовать таблицу, *false* – использовать статическую мощность). Тип данных: bool (логический тип данных – false (0) или true (1)).
- Consumer – тип объекта. Тип данных: перечисление – “0” – жилой район, “1” – производство, “2” – медицинские объекты.
- Sockets – количество подводимых линий. Тип данных: целое число.
- Power – статическая мощность потребления в условных единицах. Тип данных: число двойной точности с плавающей точкой.

```
public sealed class PowerTimeTable
{
    [DataMember]
    public PowerTimestamp[] EveryDay { get; set; }

    [DataMember]
    public PowerTimestamp[] Weekend { get; set; }
}
```

- EveryDay – нагрузка по временным отрезкам в рабочие дни. Тип данных: массив объектов C# *PowerTimestamp*.
- Weekend – нагрузка по временным отрезкам в выходные дни. Тип данных: массив объектов C# *PowerTimestamp*.

```
public sealed class PowerTimestamp
{
    [DataMember]
    public TimeSpan Timestamp

    [DataMember]
    public double Power
}
```

- Timestamp – временная метка начала действия установленного значения параметра *Power*. Тип данных: TimeSpan (C# JSON).
- Power – значение мощности в условных единицах. Тип данных: число двойной точности с плавающей точкой.

UpdatePowerObject

Обновляет параметры выбранного объекта. Вид передаваемого JSON-объекта:

```
{
    "__type": "PowerConsumer:#Novator.GreenCity.Common",
    "ID": "Завод №2",
    "Consumer": 1,
    "Power": 100,
    "PowerTable": null,
    "Sockets": 2,
    "UseTable": false
},
```


Метод запроса POST. JSON-объект передается как параметр *chobj*. При успешном завершении операции, возвращается *true*, в противном случае – *false*.

GetPowerObject

Возвращает объект по его идентификатору. Метод GET. Запрос выполняется со строковым параметром key, определяющим идентификатор объекта. Результат запроса – text/json: **{"__type": "PowerStation:#Novator.GreenCity.Common","ID": "СБ2","Power": 100, "PowerTable": null,"StationType": 2,"UseTable": false}**. Вид запроса: http://localhost:8004/JSONGreenCity/GetPowerObject?key=АБ2.

TurnOn

Подключить ввод объекта. Метод GET. Запрос выполняется с параметрами по следующему шаблону: *TurnOn?key={key}&soketnum={soketnum}*. Запрос ничего не возвращает клиенту (отправили и забыли).

- key – идентификатор объекта. Тип данных: строка.
- soketnum – номер ввода. Тип данных: целое число.

TurnOff

Отключить ввод объекта. Метод GET. Запрос выполняется с параметрами по следующему шаблону: *TurnOff?key={key}&soketnum={soketnum}*. Запрос ничего не возвращает клиенту (отправили и забыли).

- key – идентификатор объекта. Тип данных: строка.
- soketnum – номер ввода. Тип данных: целое число.

GetPlcDevices

Возвращает настройки ПЛК-устройств. Метод GET.

- SunUse – подключить имитатор солнца. Тип данных: bool (логический тип данных – false (0) или true (1)).
- SunValue – статическая мощность в процентах имитатора солнца. Тип данных: беззнаковое 16-ти битное целое число.
- SunModeling – использовать моделирование имитатора солнца. Тип данных: bool (логический тип данных – false (0) или true (1)).
- Lamp1Table – мощность первой лампы имитатора солнца по временным отрезкам. Тип данных: массив объектов C# *PowerTimestamp*.
- Lamp2Table – мощность второй лампы имитатора солнца по временным отрезкам. Тип данных: массив объектов C# *PowerTimestamp*.
- WindUse – задействовать имитатор ветра. Тип данных: bool (логический тип данных – false (0) или true (1)).
- WindValue – статическая мощность в процентах имитатора ветра. Тип данных: беззнаковое 16-ти битное целое число.
- WindModeling – использовать моделирование имитатора ветра. Тип данных: bool (логический тип данных – false (0) или true (1)).
- WindTable– мощность имитатора ветра по временным отрезкам. Тип данных: массив объектов C# *PowerTimestamp*.

SetPlcDevices

Установить настройки ПЛК-устройств. Метод POST. В качестве параметра *data* передается объект аналогичный результату работы *GetPlcDevices*. Запрос ничего не возвращает клиенту (отправили и забыли).

GetPlcServiceCfg

Настройки сервиса взаимодействия с ПЛК. Метод GET.

- TreeSettings – настройки подключения к основному ПЛК. Тип данных: массив объектов C# *DeviceConnectionSettings*.
- ElementsSettings – настройки подключения к вспомогательным ПЛК. Тип данных: массив объектов C# *DeviceConnectionSettings*.
- WaitTask – время паузы между опросами в мс. Тип данных: целое число. Минимальное значение 300 мс, максимальное – 1500 мс.
- RestartTask – время перезапуска сервиса в сек., в случае некорректируемого сбоя. Тип данных: целое число. Минимальное значение 1 сек., максимальное – 25 сек.

```
public sealed class DeviceConnectionSettings
{
    [DataMember]
    public int BaudRate
```

```
        [DataMember]
        public System.IO.Ports.Parity Parity

        [DataMember]
        public int DataBits

        [DataMember]
        public System.IO.Ports.StopBits StopBits

        [DataMember]
        public int ReadTimeout

        [DataMember]
        public int WriteTimeout

        [DataMember]
        public string PortName
    }
```

- BaudRate – скорость передачи данных в бодах по последовательному порту. Тип данных: целое число. Параметр может принимать одно из следующих значений: 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200.
- Parity – задает бит четности для последовательного порта. Тип данных: перечисление (“0” – контроль четности не осуществляется; “1” – устанавливает бит четности так, чтобы число установленных битов всегда было нечетным; “2” – устанавливает бит четности так, чтобы число установленных битов всегда было четным; “3” – оставляет бит четности равным 1; “4” – оставляет бит четности равным 0).
- DataBits – число битов данных в байте. Тип данных: целое число. Параметр может принимать одно из следующих значений: `5, 6, 7, 8`.
- StopBits – задает число стоповых битов для последовательного порта. Тип данных: перечисление (“0” – стоповые биты не используются; “1” – используется один стоповый бит; “2” – используется два стоповых бита; “3” – используется 1,5 стоповых бита).
- ReadTimeout – максимальное допустимое время в мс получения данных по последовательному порту. Тип данных: целое число. Минимальное значение 0 мс, максимальное – 2000 мс.
- WriteTimeout – максимальное допустимое время в мс записи данных в последовательный порт. Тип данных: целое число. Минимальное значение 0 мс, максимальное – 500 мс.
- PortName – имя последовательного порта, подключенного к сервису.

SetPlcServiceCfg

Установить настройки сервиса взаимодействия с ПЛК. Метод POST. В качестве параметра *cfg* передается объект аналогичный результату работы *GetPlcServiceCfg*. Возвращаемое значение всегда *true*.

PlcPorts

Возвращает массив строк, содержащий имен портов, доступных системе. Метод GET. Запрос выполняется без параметров.

Парсинг JSON на Питоне

Людям, работающим с веб часто приходится сталкиваться с необходимостью взаимодействовать с объектами JSON (JavaScript Object Notation). Это удобный способ структурированно хранить необходимые данные. Понятное дело, что и на Python должна быть реализация взаимодействия со строками JSON.

Разработчики Python позаботились об этом, поэтому в состав стандартной библиотеки Python входит модуль json. Почитать подробнее о нем можно в официальной документации, а пока мы сконцентрируемся на конкретной задаче:

Распарсить строку json средствами Python.

Представим, что у нас есть следующая строка json, представляющая заказ товара в интернет-магазине:

```
{
 "orderID": 42,
 "customerName": "John Smith",
 "customerPhoneN": "555-1234",
 "orderContents": [
  {
   "productID": 23,
   "productName": "keyboard",
   "quantity": 1
  },
  {
   "productID": 13,
   "productName": "mouse",
   "quantity": 1
```

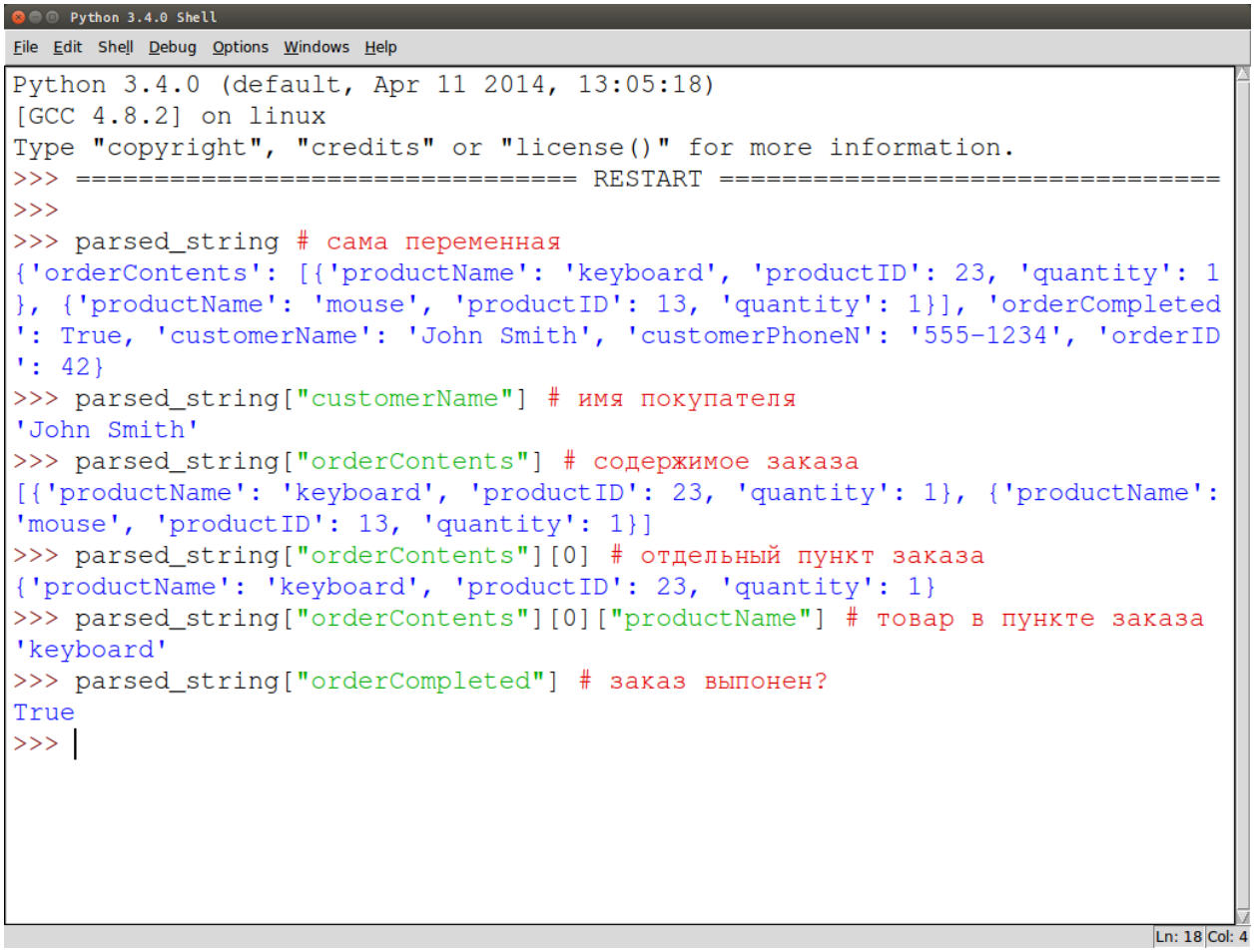
```
    }
  ],
  "orderCompleted": true
}
```

Тогда код на Python должен быть следующим:

```
# строка которую будем парсить
json_string = """ {
  "orderID": 42,
  "customerName": "John Smith",
  "customerPhoneN": "555-1234",
  "orderContents": [
    {
      "productID": 23,
      "productName": "keyboard",
      "quantity": 1
    },
    {
      "productID": 13,
      "productName": "mouse",
      "quantity": 1
    }
  ],
  "orderCompleted": true
} """

# распарсенная строка
parsed_string = json.loads(json_string)
```

Переменная parsed_string будет иметь тип данных словарь, следовательно любой элемент строки json можно будет получить через необходимый ключ:



Как видите, для того чтобы распарсить строку json средствами Python достаточно воспользоваться встроенной библиотекой этого прекрасного языка программирования.

Список литературы

https://python-scripts.com/json

https://pythonru.com/uroki/modul-json-uroki-po-python-dlja-nachinajushhih

http://pythonicway.com/education/basics/17-python-json-parse