Вычислительная Инженерная Платформа

Механизм подключения пользовательских модулей к Вычислительной Инженерной Платформе

РУКОВОДСТВО ПРОГРАММИСТА

СОДЕРЖАНИЕ

1. ПОЛЬЗОВАТЕЛЬСКИЕ МОДУЛИ

1.1 ОБЩЕЕ ОПИСАНИЕ

Пользовательский модуль – это библиотека, написанная пользователем на любом языке высокого уровня, имеющем интерфейс с языком C, и откомпилированная любым компилятором для одной или нескольких платформ. Вычислительная инженерная платформа (ВИП) динамически загружает пользовательские модули и вызывает определённые функции. Пользовательские модули в процессе работы имеют доступ к API ВИП и могут вызывать те или иные функции.

Интерфейс C был выбран по той причине, что большинство операционных систем предоставляют API ядра в виде интерфейса C, в связи с чем большинство языков высокого уровня его поддерживают.

1.2 СТРУКТУРА ПОЛЬЗОВАТЕЛЬСКОГО МОДУЛЯ

Пользовательский модуль представляет собой zip-архив, содержащий в себе:

- описание модуля в файле **main.xml**
- описание входных параметров модуля в файле **params.xml**
- динамически загружаемые библиотеки, расположенные в директориях вида *система_архитектура* (например, Linux_x64)

Файл zip-архива должен иметь расширение **.fvdll**

При подключении пользовательского модуля к проекту FlowVision он автоматически копируется в клиентскую директорию этого проекта, а затем, при синхронизации с решателем, передаётся по сетевому соединению в серверную директорию проекта.

В ней, в момент загрузки проекта, решатель создаёт временную поддиректорию для каждого пользовательского модуля, подключённого к этому проекту. Туда из zip-архива распаковываются все файлы исполняемого кода модуля, соответствующие платформе текущей машины. Например, если это 64-битная Windows, из архива будут распакованы файлы Windows_x64/*.*, рекурсивно, включая возможные поддиректории. Таким образом, код пользовательского модуля может состоять более чем из одного файла (например, если им используются сторонние библиотеки). При закрытии проекта солвер удаляет временную поддиректорию со всем содержимым.

Файл **main.xml** содержит общую информацию о модуле:

- тип модуля
- название модуля
- краткое описание модуля (опционально)
- информацию об авторских правах (опционально)
- глобально уникальный идентификатор (GUID) модуля
- версию модуля (опционально)
- язык текстовой информации модуля для выбора по умолчанию (опционально)

- требование к версии FlowVision для подключения данного модуля
- поддерживаемые платформы (системы и архитектуры)

Текстовая информация (название, описание, информация об авторских правах) может быть представлена на нескольких языках. Язык задаётся значением XML-атрибута **xml:lang**. В соответствии с выбранным языком Препроцессор отображает соответствующую текстовую информацию пользовательского модуля. Приоритет (от высшего к низшему) при выборе многоязычных элементов:

- элемент, у которого атрибут **xm:lang** соответствует текущему языку интерфейса Препроцессора (на данный момент это может быть только русский "ru" или английский "en");
- элемент, у которого атрибут **xml:lang** не задан;
- первый элемент по порядку.

Поддерживаемые Солвером комбинации систем и архитектур:

- Windows/x64
- Linux/x64

Шаблон файла **main.xml** (значения элементов приведены в качестве примера):

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<FVDLL>
  <Type>Binder</Type>
  <Name xml:lang="en">Binder template</Name>
  <Name xml:lang="ru">Шаблон связки</Name>
  <Description xml:lang ="en">Template of user module for custom BC binder</Description>
  <Description xml:lang ="ru">Шаблон пользовательского модуля для нестандартной связки
ГУ</Description>
  <Copyright xml:lang ="en">«NEP» Ltd. 2016-2019</Copyright>
  <Copyright xml:lang ="ru">ООО «ВИП» 2016-2019</Copyright>
  <Version>1.00</Version>
  <GUID>CC59A330-6F69-411C-BE2D-A999E44E491D</GUID>
  <SolverVersion>3.10.01</SolverVersion>
  <Platforms>
    <Windows>
      <x64>BinderTemplate_x64.dll</x64>
    </Windows>
    <Linux>
      <x64>BinderTemplate_x64.so</x64>
    </Linux>
  </Platforms>
</FVDLL>
```

Файл **params.xml** содержит описание параметров пользовательского модуля, настраиваемых для каждой конкретной задачи. Эти параметры будут отображены в редакторе свойств в Препроцессоре.

Общая структура файла **params.xml**:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

```
<FVDLL_PARAMETERS>
  <!-- описания именованных структур и перечислений -->
  <ParameterSet>
    <!-- описания параметров -->
  </ParameterSet>
</FVDLL_PARAMETERS>
```

Параметры задаются xml-элементом **<Parameter>**. Параметры могут быть простыми, структурированными, а также перечислениями. Каждый параметр должен иметь следующие обязательные атрибуты:

- **type**, задающий его тип;
- **id**, задающий его уникальный в контексте родительского элемента идентификатор.

Каждый параметр должен иметь обязательный дочерний элемент **<Name>**, который задаёт отображаемое в графическом пользовательском интерфейсе название параметра и может быть повторён несколько раз с разными значениями атрибута **xml:lang**. Каждый параметр может иметь необязательный дочерний элемент **<Description>**, который задаёт отображаемое в графическом пользовательском интерфейсе краткое описание параметра и может быть повторён несколько раз с разными значениями атрибута **xml:lang**.

Пример задания простого вещественного параметра с идентификатором Ratio, значением по умолчанию 0.5 и разрешённым диапазоном (0.0; 1.0):

```
<Parameter type="Real" id="Ratio" default="0.5" min="0.0" max="1.0" strictMin="1"
strictMax="1">
  <Name xml:lang="en">Base ratio</Name>
  <Name xml:lang="ru">Отношение оснований</Name>
  <Description xml:lang="en">Truncated cone base ratio</Description>
  <Description xml:lang="ru">Отношение оснований усечённого конуса</Description>
</Parameter>
```

Возможны следующие типы параметров:

1. **Вещественное число**

   type="**Real**"

   Соответствующий тип C: **double**

   Необязательные атрибуты:

   - **default**: вещественное значение по умолчанию (по умолчанию 0.0)
   - **min**: минимальное вещественное значение (по умолчанию -DBL_MAX)
   - **max**: максимальное вещественное значение (по умолчанию DBL_MAX)
   - **strictMin**: "0" означает, что минимум включён в допустимый диапазон (параметр $\geq min$), "1" означает, что минимум не включён в допустимый диапазон (параметр $> min$)
   - **strictMax**: "0" означает, что максимум включён в допустимый диапазон (параметр $\leq max$), "1" означает, что максимум не включён в допустимый диапазон (параметр $< max$)

2. **Целое число**

   type="**Integer**"

   Соответствующий тип C: **int**

   Необязательные атрибуты:

   - **default**: знаковое 32-битное значение по умолчанию (по умолчанию 0)
   - **min**: минимальное знаковое 32-битное значение (по умолчанию INT_MIN)
   - **max**: максимальное знаковое 32-битное значение (по умолчанию INT_MAX)

3. **Байт**

   type="**Byte**"

   Соответствующий тип C: **unsigned char**

   Необязательные атрибуты:

   - **default**: 8-битное беззнаковое значение по умолчанию (по умолчанию 0)
   - **min**: минимальное 8-битное беззнаковое значение (по умолчанию 0)
   - **max**: максимальное 8-битное беззнаковое значение (по умолчанию UCHAR_MAX)

4. **Булево значение**

   type="**Boolean**"

   Соответствующий тип C: **bool**

   Необязательные атрибуты:

   - **default**: "0" (false) или "1" (true) (по умолчанию "0")

5. **Строка**

   type="**String**"

   Соответствующий тип C: **char[]**

   Необязательные атрибуты:

   - **default**: строковое значение (по умолчанию пустая строка)

6. **Перечисление**

   type="**Enumeration**"

   Соответствующий тип C: **int**

   Обязательный атрибут:

   - **class**: имя ранее описанного класса перечислений

   Необязательные атрибуты:

   - **default**: значение по умолчанию – одно из строковых значений идентификаторов констант соответствующего перечисления
     (по умолчанию первое значение из перечисления)

7. **Структура**

   type="**Structure**"

   Соответствующий тип C: **struct**

   Обязательный атрибут:

   - **class**: имя ранее описанного класса структур

8. **Файл**

   type="**File**"

   Соответствующий тип C: **FILE**

Для использования структур и перечислений в качестве параметров, их классы должны быть описаны до элемента **<ParameterSet>** при помощи элементов **<Structure>** и **<Enumeration>** соответственно.

Структуры могут быть вложенными, ссылаясь на ранее описанный класс структуры.

Общая структура элемента **<Structure>**, где “описания параметров” – последовательность элементов **<Parameter>**:

```
<Structure class="уникальный идентификатор класса структур">
  <!-- описания параметров -->
</Structure>
```

Пример описания класса структуры двумерного вектора:

```
<Structure class="Vector2d">
  <Parameter type="Real" id="X" default="1.0">
    <Name>X</Name>
    <Description xml:lang="en">X coordinate</Description>
    <Description xml:lang="ru">Координата X</Description>
  </Parameter>
  <Parameter type="Real" id="Y">
    <Name>Y</Name>
    <Description xml:lang="en">Y coordinate</Description>
    <Description xml:lang="ru">Координата Y</Description>
  </Parameter>
</Structure>
```

Общая структура элемента **<Enumeration>**:

```
<Enumeration class="уникальный идентификатор класса перечислений">
  <!-- описания констант -->
</Enumeration>
```

Константы перечислений задаются xml-элементом <**Item**>. Каждая константа должна иметь обязательный атрибут **id**, задающая её уникальный в контексте данного перечисления строковый идентификатор.

Каждая константа должна иметь обязательный дочерний элемент **<Name>**, который задаёт отображаемое в графическом пользовательском интерфейсе название константы и может быть повторён несколько раз с разными значениями атрибута **xml:lang**. Каждая константа может иметь необязательный дочерний элемент **<Description>**, который задаёт отображаемое в графическом

пользовательском интерфейсе краткое описание константы и может быть повторён несколько раз с разными значениями атрибута **xml:lang**.

Пример описания класса перечисления типовых направлений:

```
<Enumeration class="DirectionType">
    <Item id="AlongX">
      <Name xml:lang="en">Along X</Name>
      <Name xml:lang="ru">Вдоль X</Name>
      <Description xml:lang="en">Direction along X axis</Description>
      <Description xml:lang="ru">Направление вдоль оси X</Description>
    </Item>
    <Item id="AlongY">
      <Name xml:lang="en">Along Y</Name>
      <Name xml:lang="ru">Вдоль Y</Name>
      <Description xml:lang="en">Direction along Y axis</Description>
      <Description xml:lang="ru">Направление вдоль оси Y</Description>
    </Item>
    <Item id="AlongZ">
      <Name xml:lang="en">Along Z</Name>
      <Name xml:lang="ru">Вдоль Z</Name>
      <Description xml:lang="en">Direction along Z axis</Description>
      <Description xml:lang="ru">Направление вдоль оси Z</Description>
    </Item>
    <Item id="Custom">
      <Name xml:lang="en">Along vector</Name>
      <Name xml:lang="ru">Вдоль вектора</Name>
      <Description xml:lang="en">Direction along custom vector</Description>
      <Description xml:lang="ru">Направление вдоль заданного вектора</Description>
    </Item>
</Enumeration>
```

В случае описания именованных классов структур и перечислений они впоследствии могут быть использованы любое количество раз.

Однако если структура или перечисление используется только один единственный раз, то описание класса может быть опущено, а структура или перечисление описаны прямо в блоке параметров. При этом параметр-структура и параметр-перечисление задаются уже не элементом **<Parameter>**, а элементами **<Structure>** и **<Enumeration>** соответственно. Их формат совпадает с форматом при описании класса с двумя отличиями:

- нет атрибута **class**, но есть атрибут **id** (идентификатор параметра)
- среди дочерних элементов должны быть заданы обязательные **<Name>** и могут быть заданы опциональные **<Description>**

Пример описания поступательного движения с местной структурой:

```
<ParameterSet>
  <Parameter type="Real" id="Speed">
    <Name xml:lang="en">Speed</ Name>
    <Name xml:lang="ru">Скорость</ Name>
    <Description xml:lang="en">Movement speed</Description>
    <Description xml:lang="ru">Скорость движения</Description>
```

```
  </Parameter>
  <Structure id="Direction">
    <Name xml:lang="en">Направление</ Name>
    <Name xml:lang="ru"> Direction</ Name>
    <Description xml:lang="en">Movement direction</Description>
    <Description xml:lang="ru">Направление движения</Description>
    <Parameter type="Real" id="X" default="1.0">
      <Name>X</Name>
      <Description xml:lang="en">X coordinate</Description>
      <Description xml:lang="ru">Координата X</Description>
    </Parameter>
    <Parameter type="Real" id="Y">
      <Name>Y</Name>
      <Description xml:lang="en">Y coordinate</Description>
      <Description xml:lang="ru">Координата Y</Description>
    </Parameter>
    <Parameter type="Real" id="Z">
      <Name>Z</Name>
      <Description xml:lang="en">Z coordinate</Description>
      <Description xml:lang="ru">Координата Z</Description>
    </Parameter>
  </Structure>
</ParameterSet>
```

Свойство типа файл позволяет выбирать в окне свойств файл. Выбранный файл считывается с диска и сохраняется в проекте FlowVision. Для контроля загруженного в проект файла существует возможность сохранить файл из проекта на диск.

Пример параметра типа файл:

```
<Parameter type="File" id="TestFile">
    <Name xml:lang="en">File test</Name>
    <Name xml:lang="ru">Проба файла</Name>
    <Description xml:lang="en">File test description</Description>
    <Description xml:lang="ru">Дескриптор пробы файла</Description>
</Parameter>
```

1.3 СХЕМА РАБОТЫ ПОЛЬЗОВАТЕЛЬСКОГО МОДУЛЯ

Рассмотрим схему работы пользовательского модуля (Рисунок 1.3.1).

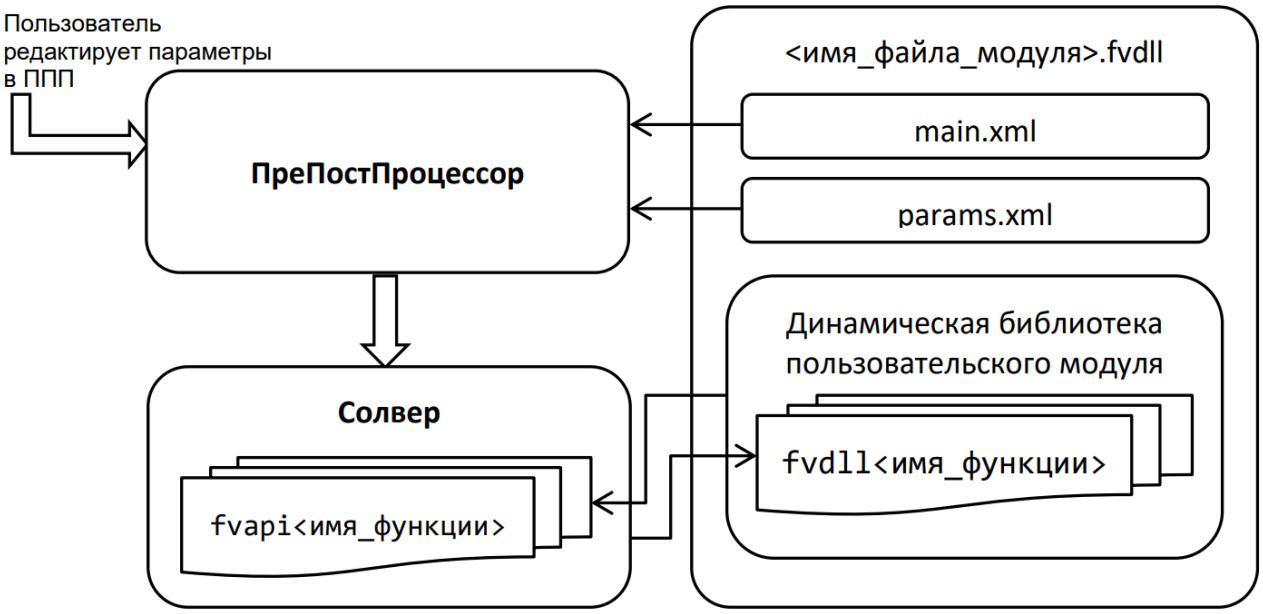

Рисунок 1.3.1 - Схема работы пользовательского модуля

Подключение модуля к проекту выполняется в ПреПостПроцессоре (ППП). При этом ППП считывает тип модуля из файла описания модуля *main.xml*. В зависимости от типа, в соответствующих местах проекта появляется возможность выбора данного модуля. Если модуль был выбран, ППП отображает список его параметров, считанный из файла *params.xml*. В каждом месте выбора модуля создаётся свой экземпляр набора параметров. Затем, файл модуля и значения параметров передаются на Солвер. Солвер подключает динамическую библиотеку модуля, передаёт ему параметры и, в необходимые моменты расчёта, вызывает соответствующие функции интерфейса пользовательского модуля. Данные функции имеют префикс *fvdll*. При этом модуль получает все необходимые данные, вызывая функции API ВИП. Такие функции имеют префикс *fvapi*.

1.4 ОБЩИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЬСКОГО МОДУЛЯ

Список обязательных функций, которые должны быть реализованы в пользовательском модуле, независимо от его типа:

```
1. FVDLL_API fvdllInitModule(FVDLL_INIT_MODULE* _pInitData)
2. FVDLL_API fvdllInitContext(FVDLL_INIT_CONTEXT* _pInitData,
   FVDLL_CONTEXT* _outContext)
3. FVDLL_API fvdllReleaseContexts()
```

Кроме обязательных функций существуют также опциональные функции, которые вызываются только в том случае, если пользовательский модуль их экспортирует. Данные функции рассматриваются в следующих пунктах руководства.

Все функции (как пользовательского модуля, так и API) имеют тип возвращаемого значения `INTEROP_RESULT` (часть определения `FVDLL_API`) и при успешном завершении обязаны возвращать `INTEROP_OK`.

Для работы с API ВИП применяется концепция *дескрипторов*: с помощью API-функций пользовательский модуль может получать дескрипторы сущностей Солвера (например, дескрипторы ячейки, грани, переменной и др.). Значения дескрипторов не несут смысловой нагрузки для

пользовательского модуля и применяются им для передачи в API-функции, чтобы получить другие дескрипторы или значения параметров. Исключением является *дескриптор контекста выполнения*, который порождается пользовательским модулем и передаётся в Солвер. Солвер хранит данный дескриптор и передаёт его при вызове функций пользовательского модуля.

Функция `fvdllInitModule` вызывается один раз для всего модуля при его инициализации. Функции передается указатель `_pInitData` на структуру `FVDLL_INIT_MODULE`, в которой содержится параметр `numContexts` – количество контекстов выполнения. Под контекстом выполнения подразумевается место в проекте, куда подключается пользовательский модуль. Каждый контекст выполнения имеет свой экземпляр набора параметров. Затем `numContexts` раз вызывается функция `fvdllInitContext`. Функции передается указатель `_pInitData` на структуру `FVDLL_INIT_CONTEXT`, которая содержит число параметров, задаваемых пользователем, и указатель на массив этих параметров (каждый параметр – структура типа `FVDLL_PARAMETER`). В данной функции должно быть реализовано создание *контекста выполнения* и возвращение его *дескриптора* (в качестве выходного аргумента `_outContext`, это может быть указатель или индекс). Впоследствии, при вызове расчётных функций пользовательского модуля Солвер передаст в них этот дескриптор. Кроме того, в данных функциях инициализации рекомендуется проводить затратные по времени операции, выполнение которых требуется однократно, например:

- аллокация памяти,
- необходимые предрасчеты,
- запрос *дескрипторов* переменных, значения которых необходимо получать в процессе работы модуля.

Структура `FVDLL_PARAMETER` представлена в таблице **Ошибка! Источник ссылки не найден.**.

Таблица 1.4.1 - Структура FDLL_PARAMETER

Тип данных	**Имя поля**	**Содержимое**
`INTEROP_STRING`	`id`	уникальный идентификатор параметра
`FVENUM_PARAMETER_TYPES`	`type`	тип параметра
`FVDLL_PARAMETER_VALUE`	`value`	значение параметра

Уникальный идентификатор параметра формируется из имён структур, в которые он вложен, разделённых косой чертой, и имени параметра. Возьмём для примера параметр с именем “X”, вложенный в структуру с именем “Vector”, которая, в свою очередь, вложена в структуру с именем “AdditionalParameters”. Тогда его `id = ”AdditionalParameters/Vector/X”`.

Таблица 1.4.2 – Перечисление `FVENUM_PARAMETER_TYPES`

Значение перечисления	**Соответствующий тип**
`FVDLL_PARAM_REAL`	вещественное число
`FVDLL_PARAM_INTEGER`	целое число

FVDLL_PARAM_BYTE	байт
FVDLL_PARAM_BOOLEAN	булево значение
FVDLL_PARAM_STRING	строка
FVDLL_PARAM_ENUMERATION	строковое значение перечисления
FVDLL_PARAM_FILE	данные из файла

Таблица 1.4.3 – Объединение FVDLL_PARAMETER_VALUE

Тип данных	Имя поля	Содержимое
INTEROP_DOUBLE	real	вещественное число двойной точности
INTEROP_INT32	integer	32-битное знаковое целое
INTEROP_UINT8	byte	8-битное беззнаковое целое
INTEROP_BOOL	boolean	8-битное булево целое
INTEROP_STRING	pszString	указатель на строку, заканчивающуюся нулём
INTEROP_STRING	pszEnum	указатель на строковое значение перечисления, заканчивающееся нулём
FVDLL_PARAMETER_FILE*	pFile	указатель на структуру параметра, содержащего данные из файла

Свойство, содержащее данные из файла, рассмотрено подробнее в п. 1.6.

Функция fvdllReleaseContexts должна уничтожить все созданные ранее контексты выполнения и освободить захваченные пользовательским модулем ресурсы.

1.5 ФУНКЦИИ, ВЫЗЫВАЕМЫЕ ДО/ПОСЛЕ ШАГА ПО ВРЕМЕНИ

Для выполнения необходимых действий до/после шага по времени, в пользовательском модуле могут быть реализованы следующие функции:

```
1. FVDLL_API fvdllBeforeTimeStep(FVDLL_CONTEXT _hContext)
2. FVDLL_API fvdllAfterTimeStep(FVDLL_CONTEXT _hContext)
```

Наличие данных функций не обязательно, их вызов будет осуществляться только в том случае, если они экспортируются библиотекой пользовательского модуля.

Функция fvdllBeforeTimeStep вызывается перед началом каждого расчетного шага в Солвере. В нее передаётся *дескриптор контекста выполнения.*

Функция fvdllAfterTimeStep вызывается после каждого расчетного шага, после загрузки проекта с диска, при запуске проекта на расчет с нуля. Список аргументов функции аналогичен функциии fvdllBeforeTimeStep.

1.6 СВОЙСТВО ТИПА ФАЙЛ

Свойство типа файл позволяет пользователю при редактировании проекта в параметрах модуля выбрать произвольный файл (1.6.1). Данные из выбранного файла загружаются и сохраняются в проекте. Файл может быть произвольного типа, поэтому в Препроцессоре нет средств просмотра содержимого файла. Вместо этого присутствует функция сохранения файла на диск, откуда его можно открыть соответствующей программой просмотра.

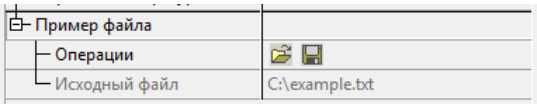

Рисунок 1.6.1 – Свойство типа файл

Для удобства, путь к загруженному файлу доступен для просмотра в окне свойств в поле «Исходный файл».

Свойство, представленное на рисунке, определяется в файле **params.xml** следующим кодом:

```
<Parameter type="File" id="FileExample">
    <Name xml:lang="en">Example of file</Name>
    <Name xml:lang="ru">Пример файла</Name>
</Parameter>
```

В пользовательский модуль свойство типа файл передаётся в виде указателя на структуру `FVDLL_PARAMETER_FILE`, которая представлена в таблице 1.6.1.

Таблица 1.6.1– Структура `FVDLL_PARAMETER_FILE`

Тип данных	Имя поля	Содержимое
`INTEROP_BYTE*`	`data`	указатель на массив с данными
`INTEROP_COUNT`	`size`	размер данных в байтах

Гарантируется, что в массиве с данными после данных находится нулевое значение. В случае текстового файла это означает, что `data` является указателем на нуль-терминированную строку. При использовании языка Си чтение данных возможно с помощью функции `sscanf`. Если разработка пользовательского модуля ведётся на языке C++, возможно сконструировать объект класса `stringstream` стандартной библиотеки C++. Чтение данных из полученного объекта выполняется аналогично чтению из файлового потока.

1.7 СОХРАНЕНИЕ СОСТОЯНИЯ ПОЛЬЗОВАТЕЛЬСКОГО МОДУЛЯ НА ДИСК

В процессе счета Солвер сохраняет данные расчета на диск в заданные пользователем моменты. Пользовательский модуль тоже может предоставить свои данные для записи на диск (и получить их при чтении с диска). Данные могут быть в виде набора «ключ-значение» и/или произвольного блока бинарных данных.

Для использования механизма сохранения параметров в виде набора «ключ-значение» в модуле должны быть реализованы следующие функции:

```
1. FVDLL_API fvdllLoadParameters(FVDLL_CONTEXT _hContext,
   FVDLL_SAVED_PARAMETERS* _pParams);
2. FVDLL_API fvdllSaveParameters(FVDLL_CONTEXT _hContext);
```

Наличие данных функций не обязательно, их вызов будет осуществляться только в том случае, если они экспортируются библиотекой пользовательского модуля.

Солвер, считав из файла набор параметров экземпляра пользовательского модуля, вызывает функцию `fvdllLoadParameters`, в которую передаёт пары «ключ-значение». При этом `FVDLL_SAVED_PARAMETERS` является псевдонимом типа `FVDLL_INIT_CONTEXT`, то есть параметры передаются тем же механизмом, что определён в п.1.4 в таблицах **Ошибка! Источник ссылки не найден.** REF _Ref124440307 \h Таблица 1.4.1Таблица 1.4.2Таблица 1.4.3

При сохранении параметров в файл, Солвер вызывает функцию `fvdllSaveParameters` пользовательского модуля. В этой функции необходимое число раз следует вызвать следующие функции API:

```
1. FV_API fvapiSaveParamReal(INTEROP_STRING _id, INTEROP_DOUBLE
   _value);
2. FV_API fvapiSaveParamInteger(INTEROP_STRING _id, INTEROP_INT32
   _value);
3. FV_API fvapiSaveParamByte(INTEROP_STRING _id, INTEROP_UINT8 _value);
4. FV_API fvapiSaveParamBoolean(INTEROP_STRING _id, INTEROP_BOOL
   _value);
5. FV_API fvapiSaveParamString(INTEROP_STRING _id, INTEROP_STRING
   _value);
```

Для использования механизма сохранения бинарных данных в модуле должны быть реализованы следующие функции:

```
1. FVDLL_API fvdllLoadBinaryData(FVDLL_CONTEXT _hContext,
   FVDLL_BINARY_DATA* _pBinData);
2. FVDLL_API fvdllSaveBinaryData(FVDLL_CONTEXT _hContext,
   FVDLL_BINARY_DATA** _outBinData);
3. FVDLL_API fvdllFreeBinaryDataBuffer();
```

Наличие данных функций не обязательно, их вызов будет осуществляться только в том случае, если они экспортируются библиотекой пользовательского модуля.

Солвер, считав из файла бинарные данные экземпляра пользовательского модуля, вызывает функцию `fvdllLoadBinaryData`, в которую передаёт указатель на `FVDLL_BINARY_DATA`. Тип `FVDLL_BINARY_DATA` является псевдонимом структуры `FVDLL_PARAMETER_FILE` (см п. 1.6) и содержит размер бинарных данных в байтах и указатель на бинарный блок.

При сохранении параметров в файл, Солвер вызывает функцию `fvdllSaveBinaryData` пользовательского модуля. В этой функции необходимо выделить нужное количество памяти, записать в неё данные и вернуть в Солвер число байт и указатель на область памяти.

После сохранения будет вызвана функция `fvdllFreeBinaryDataBuffer` пользовательского модуля. В ней нужно очистить ранее выделенную память.

1.8 ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЬСКОГО МОДУЛЯ ТИПА EVALUATOR

В пользовательском модуле типа Evaluator должны быть реализованы все функции общего интерфейса, рассмотренные в п. 1.4. Кроме этого, должна быть реализована функция вычисления значения:

```
1. FVDLL_API fvdllEvaluateScalar(FVDLL_CONTEXT _hContext, FVAPI_CONTEXT
   _hVarContext, INTEROP_DOUBLE* _outValue)
```

Функция вычисления значения `fvdllEvaluateScalar` – главная функция пользовательского модуля типа *Evaluator*. Солвер, при запросе значения, передаёт в функцию *дескриптор контекста выполнения* `_hContext` (по нему пользовательский модуль определяет, какой именно экземпляр вычислителя из своего хранилища нужно использовать). В качестве второго аргумента, Солвер передаёт дескриптор *контекста вычисления значения* `_hVarContext`, содержащий информацию о том, где именно вычисляется значение. Например, если значение вычисляется на граничном условии, в контексте содержится информация о грани, на которой запрашивается значение, и ячейки (для однозначного определения стороны грани). Функция должна вернуть вещественное значение `_outValue`.

1.9 ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЬСКОГО МОДУЛЯ ТИПА BINDER

Список обязательных функций, которые должны быть реализованы в пользовательском модуле типа Binder (в дополнение к функциям, рассмотренным в п. 1.4):

```
1. FVDLL_API fvdllInitBinder(FVDLL_CONTEXT _hContext, FVAPI_BCONDITION
   _hBCond1, FVAPI_BCONDITION _hBCond2)
2. FVDLL_API fvdllGetValueInContext(FVDLL_CONTEXT _hContext,
   FVAPI_VARIABLE _hVariable, FVAPI_CONTEXT _hVarContext,
   INTEROP_DOUBLE* _outValue)
```

Все функции (как пользовательского модуля, так и API) имеют тип возвращаемого значения `INTEROP_RESULT` (часть определения `FVDLL_API`) и при успешном завершении обязаны возвращать `INTEROP_OK`.

Функция `fvdllInitBinder` вызывается при инициализации модуля `numContexts` раз. В функцию передаётся *дескриптор контекста выполнения* и два дескриптора связываемых граничных условий.

С помощью функции `fvdllGetValueInContext` Солвер запрашивает значения переменных на границах, которые связывает модуль. В функцию передаются дескриптор контекста выполнения, дескриптор переменной, значение которой необходимо вернуть, дескриптор контекста вычисления значения, содержащий информацию о том, где именно вычисляется значение. Функция должна вернуть вещественное значение `_outValue`.

1.10 ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЬСКОГО МОДУЛЯ ТИПА BOUNDARYCONDITION

Список обязательных функций, которые должны быть реализованы в пользовательском модуле типа BoundaryCondition (в дополнение к функциям, рассмотренным в п. 1.4):

```
1. FVDLL_API fvdllInitParameterContext(FVDLL_CONTEXT _hContext,
   FVAPI_VARIABLE _hVariable, INTEROP_STRING _idParam, FVDLL_CONTEXT*
   _outParameterContext)
```

2. `FVDLL_API fvdllGetValueInContext(FVDLL_CONTEXT _hContext, FVDLL_PARAMETER_CONTEXT _hParameterContext, FVAPI_CONTEXT _hVarContext, INTEROP_DOUBLE* _outValue)`

Все функции (как пользовательского модуля, так и API) имеют тип возвращаемого значения `INTEROP_RESULT` (часть определения `FVDLL_API`) и при успешном завершении обязаны возвращать `INTEROP_OK`.

Функция `fvdllInitParameterContext` вызывается при инициализации модуля. Число вызовов данной функции равно произведению количества использований модуля в проекте на суммарное число параметров граничных условий, задаваемых модулем. В функцию передается *дескриптор контекста выполнения*, дескриптор переменной, для которой требуется параметр граничного условия, строковой идентификатор параметра. Модуль должен сгенерировать и возвратить *дескриптор контекста запроса параметра*. Подробнее параметры, в зависимости от используемых граничных условий, будут рассмотрены в п.2.4.

С помощью функции `fvdllGetValueInContext` Солвер запрашивает значения параметров граничных условий, которые заданы модулем.

1.11 ОТЛАДКА МОДУЛЯ

Чтобы выполнять отладку пользовательского модуля из среды MS Visual Studio, необходимо следовать следующему алгоритму:

1. Выполнить построение пользовательского модуля в конфигурации *Debug*
2. Добавить в нужную директорию *.fvdll* архива полученные *.dll* и *.pdb* файлы
3. Запустить новый Солвер с помощью Терминала или ПреПостПроцессора
4. Из проекта пользовательского модуля выполнить команду Отладка → Присоединиться к процессу… → FvSolver.exe / FvSolver64.exe
5. Расставить точки останова
6. Загрузить проект на Солвер
7. Выполнять отладку

2. ИНТЕРФЕЙС API ВЫЧИСЛИТЕЛЬНОЙ ИНЖЕНЕРНОЙ ПЛАТФОРМЫ

2.1 ОБЩЕЕ ОПИСАНИЕ

Вычислительная инженерная платформа реализует интерфейс API, функции которого могут быть вызваны из пользовательского модуля. Типы аргументов функций также определены в API. Имена функций имеют префикс *fvapi* и возвращают значение типа `INTEROP_RESULT`. При успешном завершении функции возвращают `INTEROP_OK`, в противном случае возвращается код ошибки.

2.2 ТИПЫ ДАННЫХ

Заголовочный файл *interop_types.h*, входящий в интерфейс API ВИП, определяет ряд независимых от компилятора типов фиксированного размера для обмена данными между Солвером и пользовательскими модулями.

- `INTEROP_BYTE`, `INTEROP_INT8` – знаковое 8-битное целое
- `INTEROP_UBYTE`, `INTEROP_UINT8` – беззнаковое 8-битное целое
- `INTEROP_INT16` – знаковое 16-битное целое
- `INTEROP_UINT16` – беззнаковое 16-битное целое
- `INTEROP_INT32` – знаковое 32-битное целое
- `INTEROP_UINT32` – беззнаковое 32-битное целое
- `INTEROP_INT64` – знаковое 64-битное целое
- `INTEROP_UINT64` – беззнаковое 64-битное целое
- `INTEROP_ENUM` – знаковое 32-битное целое для хранения значений перечислений
- `INTEROP_RESULT` – беззнаковое 32-битное целое для хранения кодов результатов
- `INTEROP_BOOL` – беззнаковое 8-битное целое для хранения булевых значений
- `INTEROP_STRING` – указатель на строку в кодировке UTF-8, заканчивающуюся нулём
- `INTEROP_DOUBLE` – вещественное число двойной точности
- `INTEROP_HANDLE` – указатель на сущность

2.3 ФУНКЦИИ API ВИП

Интерфейс API ВИП включает функции для:

- протоколирования сообщений,
- получения информации из статуса расчета,

позволяет работать с

- ячейками расчетной сетки,

- гранями ячеек,
- подобластями и граничными условиями,
- фазами, моделями физических процессов и переменными,

получать

- интегральные характеристики,
- значения из контекста вычисления значения переменной.

Список функций API ВИП с подробным описанием доступен в формате *Microsoft Compiled HTML Help* (.chm).

2.4 ТИПЫ ГРАНИЧНЫХ УСЛОВИЙ

2.4.1 Общая информация

При создании пользовательского модуля типа *BoundaryCondition* в файле описания модуля необходимо для каждой переменной, определяемой модулем, задать базовый тип граничного условия FlowVision. Параметры данного граничного условия при этом становятся недоступны для редактирования в интерфейсе и запрашиваются у модуля в процессе расчета. Данный раздел представляет собой справочник с описанием возможных типов базовых граничных условий и набора параметров каждого из них.

2.4.2 CBVarAblationTempCarcass

Название в интерфейсе: Абляция

Описание в интерфейсе: Граничное условие, учитывающее абляцию материала стенки

Данное граничное условие не требует задания параметров.

2.4.3 CBVarAutoEOTURB

Название в интерфейсе: Авто-условие для турб. диссипации

Описание в интерфейсе: Пользователю ничего задавать не нужно

Данное граничное условие не требует задания параметров.

2.4.4 CBVarAutoETETA

Название в интерфейсе: Авто-условие для переменной Eteta

Описание в интерфейсе: Пользователю ничего задавать не нужно

Данное граничное условие не требует задания параметров.

2.4.5 CBVarAutoRETETA

Название в интерфейсе: Авто-условие для TR_RETETA

Описание в интерфейсе: Данное ГУ автоматически вычисляет (по интенсивности турбулентности) и устанавливает фиксированное значение переменной TR_RETETA на входе в расчётную область

Данное граничное условие не требует задания параметров.

2.4.6 CBVarConstant

Название в интерфейсе: Значение

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.1- Набор параметров CBVarConstant

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.7 CBVarConstantAblation

Название в интерфейсе: Абляция

Описание в интерфейсе: Граничное условие, учитывающее абляцию материала стенки

Данное граничное условие не требует задания параметров.

2.4.8 CBVarConstantInlet

Название в интерфейсе: Значение на входе

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.2- Набор параметров CBVarConstantInlet

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.9 CBVarConstantPotential

Название в интерфейсе: Значение эл потенциала

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.3- Набор параметров CBVarConstantPotential

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.10 CBVarConstantPrt

Название в интерфейсе: Значение

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.4- Набор параметров CBVarConstantPrt

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.11 CBVarConstantPrtCoalDiam

Название в интерфейсе: Уголь

Описание в интерфейсе: Модель горения угля

Данное граничное условие не требует задания параметров.

2.4.12 CBVarConstantPrtCoalMFractions

Название в интерфейсе: Уголь

Описание в интерфейсе: Модель горения угля

Данное граничное условие не требует задания параметров.

2.4.13 CBVarConstantSlow

Название в интерфейсе: Значение

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.5- Набор параметров 2.4.13.CBVarConstantSlow

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.14 CBVarConstantTemp

Название в интерфейсе: Температура

Описание в интерфейсе: Значение температуры, задаваемое на данной поверхности

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.6- Набор параметров CBVarConstantTemp

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.15 CBVarConstantTempCarcass

Название в интерфейсе: Значение

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.7- Набор параметров CBVarConstantTempCarcass

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.16 CBVarConstantTempPrt

Название в интерфейсе: Значение

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.8- Набор параметров CBVarConstantTempPrt

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.17 CBVarConstantTempSlow

Название в интерфейсе: Температура при медленном течении

Описание в интерфейсе: Значение температуры, задаваемое на данной поверхности при медленном вытекании жидкости

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.9- Набор параметров CBVarConstantTempSlow

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.18 CBVarConstantUnit

Название в интерфейсе: Единица для TR_GAMMA

Описание в интерфейсе: Данное ГУ автоматически устанавливает фиксированное значение "1" переменной TR_GAMMA на входе в расчётную область

Данное граничное условие не требует задания параметров.

2.4.19 CBVarConstantZero

Название в интерфейсе: Фиксированное нулевое значение значение

Данное граничное условие не требует задания параметров.

2.4.20 CBVarDORadiationOpaqueWall

Название в интерфейсе: Непрозрачная стенка

Данное граничное условие требует задания двух параметров, которые определены в следующей таблице:

Таблица 2.4.10- Набор параметров CBVelRiemann

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
Blackness	Скалярный	Степ. черноты	Степень черноты поверхности
Diffraction	Скалярный	Показатель диффузионоого отражения	Доля излучения, отражаемая диффузионно

2.4.21 CBVarDispDiamSpectrumPrt

Название в интерфейсе: Спектр размеров

Данное граничное условие не требует задания параметров.

2.4.22 CBVarExternalConjugate

Название в интерфейсе: Внешнее сопряжение

Данное граничное условие не требует задания параметров.

2.4.23 CBVarFlux

Название в интерфейсе: Поток

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.11- Набор параметров CBVarFlux

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
FluxValue	Скалярный	Значение	Устанавливаемое значение

2.4.24 CBVarFluxPot

Название в интерфейсе: Плотность электрического тока

Описание в интерфейсе: Плотность электрического тока А/м^2

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.12- Набор параметров CBVarFluxPot

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
FluxValue	Скалярный	Значение	Устанавливаемое значение

2.4.25 CBVarHeatEmission

Название в интерфейсе: Внешний теплообмен

Описание в интерфейсе: Переменная температура стенки вычисляется с использованием заданных коэффициента теплоотдачи и температуры окружающей среды

Данное граничное условие требует задания двух параметров, которые определены в следующей таблице:

Таблица 2.4.13- Набор параметров CBVarHeatEmission

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
Alpha	Скалярный	Коэф. теплоотдачи	Коэффициент внешней теплоотдачи
WallTemp	Скалярный	T внешней среды	Относительная температура внешней среды

2.4.26 CBVarInterCellVect

Название в интерфейсе: Неопределенный

Описание в интерфейсе: Неопределенный

Данное граничное условие не требует задания параметров.

2.4.27 CBVarPermeableEulerianPrt

Название в интерфейсе: Проницаемая поверхность

Описание в интерфейсе: Вход/выход для частиц, не возмущающий их поток

Данное граничное условие не требует задания параметров.

2.4.28 CBVarRadFlux

Название в интерфейсе: Поток радиации

Описание в интерфейсе: Поток радиации с поверхности

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.14- Набор параметров CBVarRadFlux

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.29 CBVarRadFluxCalc

Название в интерфейсе: Расчёт потока радиации

Описание в интерфейсе: Поток радиации, рассчитанный по T и ст.черноты

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.15- Набор параметров CBVarRadFluxCalc

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
Blackness	Скалярный	Степ. черноты	Степень черноты поверхности

2.4.30 CBVarRiemannTemp

Название в интерфейсе: Неотраж.

Описание в интерфейсе: Неотражающее граничное условие

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.16- Набор параметров CBVarRiemannTemp

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.31 CBVarScalarSoft

Название в интерфейсе: Значение (для мягкого ГУ)

Описание в интерфейсе: Используется в случае входа через поверхность, на которой задано мягкое граничное условие (нулевая 2-я производная) для данного скаляра

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.17- Набор параметров CBVarScalarSoft

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.32 CBVarSymmetry

Название в интерфейсе: Симметрия

Данное граничное условие не требует задания параметров.

2.4.33 CBVarSymmetryPrt

Название в интерфейсе: Симметрия

Данное граничное условие не требует задания параметров.

2.4.34 CBVarTemperatureAblation

Название в интерфейсе: Абляция

Описание в интерфейсе: Граничное условие, учитывающее абляцию материала стенки

Данное граничное условие требует задания трёх параметров, которые определены в следующей таблице:

Таблица 2.4.18- Набор параметров CBVarTemperatureAblation

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
Black	Скалярный	Степ. черноты	Степень черноты поверхности
AblWallTemp	Скалярный	T кипения	Температура кипения расплава
AblWallEnth	Скалярный	h_solid	Энтальпия твёрдого тела или расплава (в несопряжённой задаче)

2.4.35 CBVarTemperatureAblationCarcass

Название в интерфейсе: Абляция каркаса

Описание в интерфейсе: Граничное условие, учитывающее абляцию материала каркаса

Данное граничное условие требует задания пяти параметров, которые определены в следующей таблице:

Таблица 2.4.19- Набор параметров CBVarTemperatureAblationCarcass

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
BlacknessCarcass	Скалярный	Степ. черноты	Степень черноты поверхности
LambdaBlowCarcass	Скалярный	Эффективности	Эффективности «третьих тел»
TempInfCarcass	Скалярный	T_inf	Относительная температура, определяющая радиационный поток, падающий на твёрдую поверхность
EnthInfCarcass	Скалярный	h_solid	Энтальпия твёрдого тела или расплава (в несопряжённой задаче)

StRhoVCarcass	Скалярный	Значение	Устанавливаемое значение

2.4.36 CBVarTemperatureRadiation

Название в интерфейсе: Радиационное равновесие

Описание в интерфейсе: Тепловой поток к стенке, обусловленный теплопроводностью среды, равен радиационному потоку энергии от стенки.

Данное граничное условие требует задания трёх параметров, которые определены в следующей таблице:

Таблица 2.4.20- Набор параметров CBVarTemperatureRadiation

Строковый идентификатор параметра	**Тип параметра**	**Название параметра в интерфейсе**	**Описание параметра**
HeatFlux	Скалярный	Поток энергии	Удельный поток энергии с твёрдой поверхности в газ / жидкость
Blackness	Скалярный	Степ. черноты	Степень черноты поверхности
TempInf	Скалярный	T_inf	Относительная температура, определяющая радиационный поток, падающий на твёрдую поверхность

2.4.37 CBVarTotalTemperature

Название в интерфейсе: Полная температура

Описание в интерфейсе: Полная температура (относительная)

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.21- Набор параметров CBVarTotalTemperature

Строковый идентификатор параметра	**Тип параметра**	**Название параметра в интерфейсе**	**Описание параметра**
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.38 CBVarTurbLengthScale

Название в интерфейсе: Масштаб турбулентности

Описание в интерфейсе: Масштаб турбулентности - [м]

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.22- Набор параметров CBVarTurbLengthScale

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.39 CBVarTurbPulsation

Название в интерфейсе: Пульсации

Описание в интерфейсе: низкие (<0.03), средние (0.03<...<0.05), высокие (0.05<...<0.1)

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.23- Набор параметров CBVarTurbPulsation

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
WallValue	Скалярный	Значение	Устанавливаемое значение

2.4.40 CBVarValueNearWall

Название в интерфейсе: Значение в ячейке рядом со стенкой

Данное граничное условие не требует задания параметров.

2.4.41 CBVarVectorConstant

Название в интерфейсе: Значение

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.24- Набор параметров CBVarVectorConstant

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
MagPot	Векторный	Значение	Значение магнитного потенциала В*с/м

2.4.42 CBVarVectorSymmetry

Название в интерфейсе: Симметрия

Данное граничное условие не требует задания параметров.

2.4.43 CBVarVectorZeroFlux

Название в интерфейсе: Нулевой градиент

Данное граничное условие не требует задания параметров.

2.4.44 CBVarZeroFlux

Название в интерфейсе: Нулевой градиент

Данное граничное условие не требует задания параметров.

2.4.45 CBVarZeroValueZeroFlux

Название в интерфейсе: Поток

Данное граничное условие не требует задания параметров.

2.4.46 CBVelDirectionPres

Название в интерфейсе: Полное давление и направление скорости

Описание в интерфейсе: Граничное условие входа, на котором задаются полное давление и направление скорости

Данное граничное условие требует задания двух параметров, которые определены в следующей таблице:

Таблица 2.4.25- Набор параметров CBVelDirectionPres

Строковый идентификатор параметра	**Тип параметра**	**Название параметра в интерфейсе**	**Описание параметра**
TotalPres	Скалярный	Полное давление	Полное давление [Па]
DirVector	Векторный	Направление скорости	Направление скорости (не обязательно единичный вектор)

2.4.47 CBVelFreeOutlet

Название в интерфейсе: Сверхзвуковой выход

Данное граничное условие не требует задания параметров.

2.4.48 CBVelInlet

Название в интерфейсе: Давление на входе

Описание в интерфейсе: Относительное входное давление, Па

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.26- Набор параметров CBVelInlet

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
Pressure	Скалярный	Значение	Давление [Па]

2.4.49 CBVelInletMassEulerianPrt

Название в интерфейсе: Массовая скорость частиц

Описание в интерфейсе: Вход для частиц

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.27- Набор параметров CBVelInletMassEulerianPrt

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
NormMassVel_Prt	Скалярный	Значение	Устанавливаемое значение

2.4.50 CBVelInletVolumeEulerianPrt

Название в интерфейсе: Объёмная скорость частиц

Описание в интерфейсе: Вход для частиц

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.28- Набор параметров CBVelInletVolumeEulerianPrt

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
NormVolVel_Prt	Скалярный	Значение	Устанавливаемое значение

2.4.51 CBVelMassNormal

Название в интерфейсе: Нормальная массовая скорость

Описание в интерфейсе: Нормальная массовая скорость

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.29- Набор параметров CBVelMassNormal

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
NormMassVel	Скалярный	Массовая скорость	Массовая скорость [кг / (м^2 сек)]

2.4.52 CBVelNormalPres

Название в интерфейсе: Норм. скорость с давлением

Данное граничное условие требует задания двух параметров, которые определены в следующей таблице:

Таблица 2.4.30- Набор параметров CBVelNormalPres

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
NormVel	Скалярный	Скорость	Скорость [м/сек]
Pressure	Скалярный	Давление	Статическое давление [Па]

2.4.53 CBVelOutlet

Название в интерфейсе: Давление

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.31- Набор параметров CBVelOutlet

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
Pressure	Скалярный	Значение	Давление [Па]

2.4.54 CBVelOutletEulerianPrt

Название в интерфейсе: Скорость частиц

Описание в интерфейсе: Скорость частиц, втекающих в расчётную область, м/с

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.32- Набор параметров CBVelOutletEulerianPrt

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
vel_Prt	Векторный	Скорость частиц	Скорость частиц, втекающих в

			рассчётную область, м/с

2.4.55 CBVelOutletPorosity

Название в интерфейсе: Давление на пористой поверхности

Описание в интерфейсе: Давление на внешней части пористой поверхности

Данное граничное условие требует задания трёх параметров, которые определены в следующей таблице:

Таблица 2.4.33- Набор параметров CBVelOutletPorosity

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
Pressure	Скалярный	Значение	Давление [Па]
Darcy_Coef_D	Скалярный	Коэф. D	Толщина пористой границы / проницаемость [1/м]
Darcy_Coef_E	Скалярный	Коэф. E	Толщина пористой границы X инерционный коэффициент

2.4.56 CBVelPermeableEulerianPrt

Название в интерфейсе: Проницаемая поверхность

Описание в интерфейсе: Вход/выход для частиц, не возмущающий их поток

Данное граничное условие не требует задания параметров.

2.4.57 CBVelRiemann

Название в интерфейсе: Неотраж.

Описание в интерфейсе: Неотражающее граничное условие

Данное граничное условие требует задания двух параметров, которые определены в следующей таблице:

Таблица 2.4.34- Набор параметров CBVelRiemann

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
VelInf	Векторный	Скорость на беск.	Скорость на бесконечности, м/с
PresInf	Скалярный	Давление на беск.	Относительное давление на бесконечности, Па

2.4.58 CBVelSuperSonic

Название в интерфейсе: Сверхзвуковой вход

Описание в интерфейсе: На данной границе должны быть определены все переменные

Данное граничное условие требует задания пяти параметров, которые определены в следующей таблице:

Таблица 2.4.35- Набор параметров CBVelSuperSonic

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
EPressureType	Перечисление	P (ГУ) тип	Тип граничного условия для давления
Pressure	Скалярный	P статическое или полное	Здесь задаётся статическое или полное давление
EVelocityType	Перечисление	V (ГУ) тип	Тип граничного условия для скорости
NormVel	Скалярный	|V| или M	Здесь задаётся модуль скорости или число Маха
DirVector	Векторный	V или направление	Здесь задаётся вектор скорости или её направление

2.4.59 CBVelTotalPres

Название в интерфейсе: Полное давление

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.36- Набор параметров CBVelTotalPres

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
TotalPres	Скалярный	Полное давление	Полное давление [Па]

2.4.60 CBVelVectorPres

Название в интерфейсе: Скорость с давлением

Описание в интерфейсе: Задаются вектор скорости и статическое давление

Данное граничное условие требует задания двух параметров, которые определены в следующей таблице:

Таблица 2.4.37- Набор параметров CBVelVectorPres

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
VelVector	Векторный	Скорость	Скорость [м/сек]
VIS_Pres	Скалярный	Давление	Статическое давление [Па]

2.4.61 CBVelVelocityFixed

Название в интерфейсе: Фиксированная скорость

Описание в интерфейсе: Заданный в интерфейсе вектор скорости не меняется в процессе расчёта

Данное граничное условие требует задания одного параметра, который определён в следующей таблице:

Таблица 2.4.38- Набор параметров CBVelVelocityFixed

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
VelVectorFixed	Векторный	Скорость	Скорость [м/сек]

2.4.62 CBVelWallAblation

Название в интерфейсе: Абляция

Описание в интерфейсе: Граничное условие, учитывающее абляцию материала стенки

Данное граничное условие не требует задания параметров.

2.4.63 CBVelWallContactEulerianPrt

Название в интерфейсе: Контакт со стенкой

Описание в интерфейсе: Упругое/неупругое взаимодействие частиц со стенкой

Данное граничное условие требует задания двух параметров, которые определены в следующей таблице:

Таблица 2.4.39- Набор параметров CBVelWallContactEulerianPrt

Строковый идентификатор параметра	Тип параметра	Название параметра в интерфейсе	Описание параметра
RestCoef_n	Скалярный	Коэфф. норм.	Коэффициент восстановления нормальной составляющей импульса частицы после соударения со стенкой

RestCoef_t	Скалярный	Коэфф. танг.	Коэффициент восстановления касательной составляющей импульса частицы после соударения со стенкой

2.4.64 CBVelWallLogarithmLaw

Название в интерфейсе: Логарифмический закон

Данное граничное условие не требует задания параметров.

2.4.65 CBVelWallNonSlip

Название в интерфейсе: Прилипание

Данное граничное условие не требует задания параметров.

2.4.66 CBVelWallSlip

Название в интерфейсе: Проскальзывание

Данное граничное условие не требует задания параметров.

2.4.67 CBVelWallTrack

Название в интерфейсе: Гусеница

Описание в интерфейсе: Задание скорости движения гусеницы. Необходимо также на этом ГУ поставить локальную систему координат (ЛСК) с вращением. Вектор вращения будет задавать направление движения гусеницы - вдоль поверхности и в плоскости, перпендикулярной вектору вращения ЛСК.

Данное граничное условие требует задания двух параметров, которые определены в следующей таблице:

Таблица 2.4.40- Набор параметров CBVelWallTrack

Строковый идентификатор параметра	**Тип параметра**	**Название параметра в интерфейсе**	**Описание параметра**
Track_Speed	Скалярный	Скорость гусеницы	Задаёт скорость движения гусеницы.
Cf_Stress_Coefficient	Скалярный	Коэффициент трения	Сила трения задаётся через коэффициент $F = \rho C_f \frac{(V-V_t)^2}{2}$, где ρ - плотность жидкости, V - скорость жидкости около гусеницы, V_t - скорость гусеницы.

3. ЛИЦЕНЗИОННАЯ ЗАЩИТА МОДУЛЯ

3.1 ВАРИАНТЫ ЛИЦЕНЗИОННОЙ ЗАЩИТЫ

Существуют различные варианты лицензионной защиты модуля:

1. Отсутствие лицензионной защиты. Данный вариант применяется, если разработчик модуля распространяет свой модуль под свободной лицензией или использует его на собственных вычислительных ресурсах, без распространения.
2. Собственный механизм лицензионной защиты. Применяется, если разработчик модуля самостоятельно реализует защитный механизм или приобретает его у независимого разработчика технических средств защиты авторских прав. Используется модулем без связи с Вычислительной Инженерной Платформой.
3. Механизм защиты, предоставляемый Вычислительной инженерной платформой. Подробно рассматривается в следующем пункте.

3.2 МЕХАНИЗМ ЛИЦЕНЗИОННОЙ ЗАЩИТЫ, ПРЕДОСТАВЛЯЕМЫЙ ВЫЧИСЛИТЕЛЬНОЙ ИНЖЕНЕРНОЙ ПЛАТФОРМОЙ

Если разработчик модуля выбирает данный механизм защиты, ему необходимо заключить договор с ООО «ВИП» и передать по защищённому каналу GUID и ключ безопасности своего модуля. В качестве ключа безопасности выступает строка произвольной длины. Для модуля выделяется лицензионная опция, которая становится доступна для покупки пользователям Вычислительной инженерной платформы.

Для проверки того, что модуль подключен к Вычислительной инженерной платформе, а не к программе злоумышленника, используется механизм аутентификации, для которого в API предусмотрена следующая функция:

```
FV_API fvapiCheckAuthenticity(INTEROP_STRING _libGUID, INTEROP_UINT64
_randInt, INTEROP_STRING _hashOut);
```

При инициализации модуль вызывает эту функцию, передаёт в неё свой GUID в строковом представлении, случайно сгенерированное число `_randInt` и буфер с выделенной для хэша памятью `_hashOut`.

Далее модуль должен самостоятельно вычислить хэш и сравнить его с тем, который вернула функция аутентификации. Если они совпадают – аутентификация прошла успешно.

Хэш вычисляется по следующему алгоритму:

Шаг 1. Случайно сгенерированное число конвертируется в строку и конкатенируется с ключом безопасности (допустим, это “SecretPassword”):

_randInt = 12345678901234567890

randIntStr = “12345678901234567890”

key = “SecretPassword”

M = randIntStr + key = “12345678901234567890SecretPassword”

Шаг 2. Для полученной строки M вычисляется функция хэширования ГОСТ Р 34.11-2012 с длиной хэш-кода 512 бит:

> H(M) = “f3c093a2c98e25d39e028a46a037210acfbd6bf1a61a818e32038b542b369ec8f75890
> d48c52f9cb8468b811e3a2b2274f3ba76d676b451afe52931630aa0870”

ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хеширования» — действующий российский криптографический стандарт, определяющий алгоритм и процедуру вычисления хеш-функции.

4. ПРИМЕР МОДУЛЯ ТИПА EVALUATOR – ВЫЧИСЛИТЕЛЬ СИНУСА

4.1 ПОСТАНОВКА ЗАДАЧИ

В качестве примера пользовательского модуля типа Evaluator рассмотрим вычислитель значения синуса.

Данный модуль возвращает значение синуса (4.1) или косинуса (4.2), в соответствии с выбранной в параметрах модуля функцией, зависящих от расстояния до точки, в которой запрашивается значение, от начала координат.

$$A\left|\sin\left(v\sqrt{(xk_x)^2+\left(yk_y\right)^2+(zk_z)^2}\right)\right| \tag{4.1}$$

$$A\left|\cos\left(v\sqrt{(xk_x)^2+\left(yk_y\right)^2+(zk_z)^2}\right)\right| \tag{4.2}$$

где (x, y, z) – координаты точки, в которой запрашивается значение, а A, v, k_x, k_y, k_z – параметры модуля, редактируемые в интерфейсе:

- A – амплитуда
- v – частота
- (k_x, k_y, k_z) – коэффициенты масштабирования

4.2 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

Пользовательский модуль типа Evaluator подключается в проекте в те места, где задаётся значение (рис. 4.2.1).

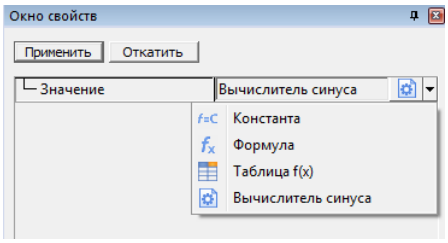

Рисунок 4.2.1 – Подключение Вычислителя синуса в качестве значения

В параметрах пользовательского модуля должна быть возможность выбора тригонометрической функции, которая будет вычисляться (рис. 4.2.2).

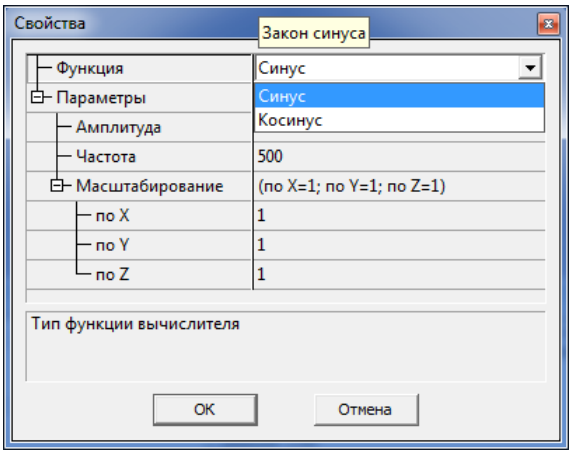

Рисунок 4.2.2 – Выбор функции, значение которой будет возвращено

Для реализации такого интерфейса потребуется описать в файле **params.xml** пользовательское перечисление:

```
<Enumeration class="FunctionType">
  <Item id="Sinus">
    <Name xml:lang="en">Sinus</Name>
    <Name xml:lang="ru">Синус</Name>
    <Description xml:lang="en">Sinus</Description>
    <Description xml:lang="ru">Закон синуса</Description>
  </Item>

  <Item id="Cosinus">
    <Name xml:lang="en">Cosinus</Name>
    <Name xml:lang="ru">Косинус</Name>
    <Description xml:lang="en">Cosinus</Description>
    <Description xml:lang="ru">Закон косинуса</Description>
  </Item>
</Enumeration>
```

Кроме этого, можно описать структуру с тремя полями *X*, *Y* и *Z* для редактирования параметров масштабирования:

```
<Structure class="StructXYZ">
  <Parameter type="Real" id="X" default="1.0">
    <Name xml:lang="en">X</Name>
    <Name xml:lang="ru">по X</Name>
    <Description xml:lang="en">Scaling on X</Description>
    <Description xml:lang="ru">Масштабирование по X</Description>
  </Parameter>

  <Parameter type="Real" id="Y" default="1.0">
    <Name xml:lang="en">Y</Name>
    <Name xml:lang="ru">по Y</Name>
    <Description xml:lang="en">Scaling on Y</Description>
    <Description xml:lang="ru">Масштабирование по Y</Description>
  </Parameter>

  <Parameter type="Real" id="Z" default="1.0">
```

```
      <Name xml:lang="en">Z</Name>
      <Name xml:lang="ru">по Z</Name>
      <Description xml:lang="en">Scaling on Z</Description>
      <Description xml:lang="ru">Масштабирование по Z</Description>
    </Parameter>
  </Structure>
```

С учётом определенных перечисления и структуры, разметка окна параметров, представленного на рисунке 4.2.3, осуществляется следующим xml-кодом:

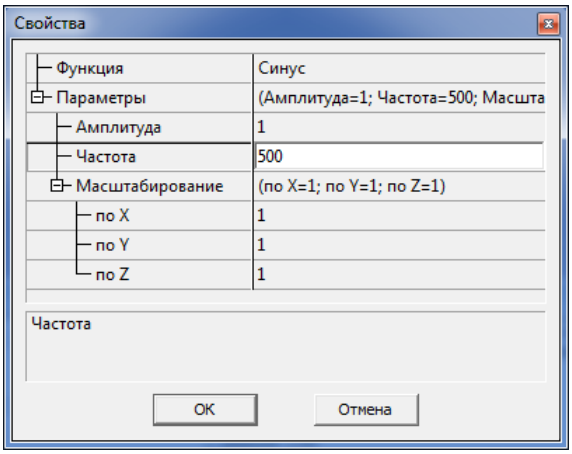

Рисунок 4.2.3 – Окно параметров Вычислителя синуса

```
<ParameterSet>
  <Parameter type="Enumeration" class="FunctionType" id="Function" default="Sinus">
    <Name xml:lang="en">Function</Name>
    <Name xml:lang="ru">Функция</Name>
    <Description xml:lang="en">Evaluator function type</Description>
    <Description xml:lang="ru">Тип функции вычислителя</Description>
  </Parameter>

  <Structure id="Parameters">
    <Name xml:lang="en">Parameters</Name>
    <Name xml:lang="ru">Параметры</Name>
    <Description xml:lang="en">Function parameters</Description>
    <Description xml:lang="ru">Параметры функции</Description>

    <Parameter type="Real" id="Amplitude" default="1.0">
      <Name xml:lang="en">Amplitude</Name>
      <Name xml:lang="ru">Амплитуда</Name>
      <Description xml:lang="en">Amplitude</Description>
      <Description xml:lang="ru">Амплитуда</Description>
    </Parameter>

    <Parameter type="Real" id="Frequency" default="1.0">
      <Name xml:lang="en">Frequency</Name>
      <Name xml:lang="ru">Частота</Name>
      <Description xml:lang="en">Frequency</Description>
      <Description xml:lang="ru">Частота</Description>
    </Parameter>

    <Parameter type="Structure" class="StructXYZ" id="Scale">
      <Name xml:lang="en">Scale</Name>
```

```
      <Name xml:lang="ru">Масштабирование</Name>
      <Description xml:lang="en">Scale</Description>
      <Description xml:lang="ru">Масштабирование</Description>
    </Parameter>
  </Structure>
</ParameterSet>
```

4.3 ПРОГРАММИРОВАНИЕ МОДУЛЯ

Как было рассмотрено в п. 1.4 и п. 1.8, в модуле типа Evaluator должны быть реализованы 4 функции, вызываемые Солвером. Две первые функции являются функциями инициализации. Для удобства, в примере определяется класс `CSinusEvaluator`. При инициализации, функция `fvdllInitContext` вызывается по количеству мест подключения модуля в проект. Соответственно, при каждом вызове в модуле создаётся объект класса `CSinusEvaluator`, которому в конструктор передаются параметры инициализации. Адрес созданного объекта передаётся в Солвер в качестве *дескриптора контекста выполнения*. **id** параметров, переданных в конструктор объекта, соответствуют именам в **params.xml** и их значения представлены в списке:

- `Function`
- `Parameters/Amplitude`
- `Parameters/Frequency`
- `Parameters/Scale/X`
- `Parameters/Scale/Y`
- `Parameters/Scale/Z`

Значения параметров-перечислений передаются в строковом виде, поэтому, в классе `CSinusEvaluator` определен `enum EFunctionType`, и при инициализации строковое значение вручную преобразуется в значение типа перечисление.

Рассмотрим алгоритм, реализованный в функции вычисления значения.

1. В функцию передаётся дескриптор контекста выполнения и дескриптор контекста вычисления значения.
2. По дескриптору контекста выполнения определяется объект, соответствующий конкретному экземпляру подключения модуля.
3. У найденного объекта вызывается соответствующий метод, в который передаётся дескриптор контекста вычисления значения.
4. Из дескриптора контекста вычисления значения с помощью функции `fvapiGetCellOfContext` запрашивается дескриптор ячейки.
5. По дескриптору ячейки с помощью функции `fvapiGetCellCenter` осуществляется получение координат центра ячейки.

6. Используя значения параметров, полученных при инициализации, по формулам, представленным в п. 4.1, происходит вычисление результирующего значения.

В функции `fvdllReleaseContexts` удаляются все хранимые объекты.

5. ПРИМЕР МОДУЛЯ ТИПА EVALUATOR – ВЫЧИСЛИТЕЛЬ GLO

5.1 ПОСТАНОВКА ЗАДАЧИ

В качестве примера пользовательского модуля типа Evaluator рассмотрим вычислитель значения по известным точкам («Вычислитель GLO»).

Модуль считывает данные из файла – сохранённых в формате .glo данных со скалярного слоя, а также из обычного текстового файла с описанием точек и значений в них в формате

```
# <point1.x><point1.y><point1.z><value1>
# <point2.x><point2.y><point2.z><value2>
# ...
# <pointN.x><pointN.y><pointN.z><valueN>
```

Данный модуль интерполирует значение в расчётной точке по известным окружающим точкам. Также модуль может экстраполировать значения вне указанного диапазона.

Для ускоренного поиска точек, которые находятся близко к расчётной, исходный массив из N точек ограничивается боксом $bBox$. Этот большой контейнер разбивается на боксы, размер которых подбирается таким образом, чтобы в одном боксе при равномерном распределении точек находилось около $\sqrt[3]{N}$ точек:

Количество точек вдоль грани единичного куба при равномерном распределении исходных точек

$$n_{unit} = \sqrt[3]{N} \tag{5.1}$$

Шаг разбиения исходного ограничивающего бокса со сторонами $sideX$, $sideY$ и $sideZ$:

$$splitStep = \max\{sideX, sideY, sideZ\} / n_{unit} \tag{5.2}$$

Таким образом, каждая сторона исходного бокса разбивается на следующее количество частей:

$$size_i = \max\left\{\frac{side_i}{splitStep}, 1\right\} \tag{5.3}$$

Для хранения точек, разделённых на sub-ячейки, создаётся массив размером $size_x * size_y * size_z$, элементы которого хранят указатели на точки, которые принадлежат соответствующей sub-ячейке.

Если расчётная точка попадает в ограничивающий бокс $bBox$ с учётом задаваемой пользователем погрешности, то для него находится соответствующая sub-ячейка, и анализируются точки из этой ячейки и окружающих её ячеек (всего 9 sub-ячеек), что позволяет избежать полного перебора всех точек для поиска ближайших.

Расчётная точка, попавшая в $bBox$, считается валидной если:

1) sub-ячейка, в которую попала точка, не пустая (ячейка может быть пустой, если исходное распределение было неравномерным).

2) Вокруг расчётной точки нашлись значащие точки со всех 6 сторон окружающего саму точку бокса со значениями. Такой бокс ($bValBox$) строится по точкам, которые находятся с разных сторон от расчётной точки, и расстояние до которых минимально

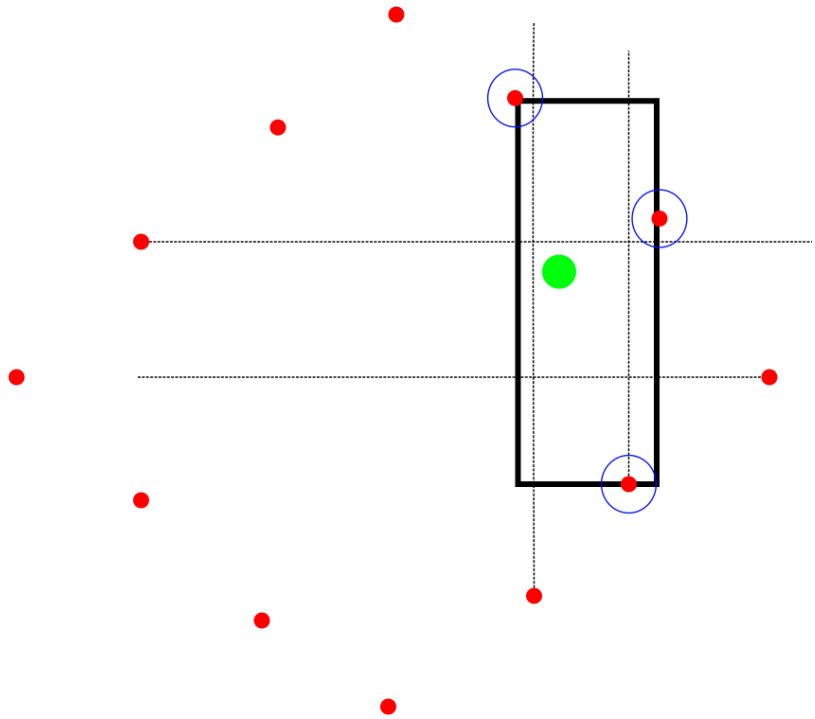

Рисунок 5.1.1 – Пример определения бокса со значениями в 2д постановке

По значениям из $bValBox$ (в общем случае до 6 различных точек) производится интерполяция путём вычисления среднего арифметического взвешенного по следующей формуле:

$$\overline{Val} = \frac{\sum_{i=1}^{n} w_i \cdot Val_i}{\sum_{i=1}^{n} w_i}, \tag{5.4}$$

где $w_i = \frac{1}{dist^2}$ – величина, обратно пропорциональная квадрату расстояния между точками.

Если расчётная точка считается не валидной, однако расстояние от неё до ближайшего объекта (точки, прямой, треугольника), образованного исходными точками меньше заданной пользователем погрешности, то значение в точке интерполируется аналогично валидной точке.

Если значение в точке не интерполировалось, то проверяется наличие включённого флага экстраполяции. Если экстраполяция включена, то в расчётной точке вычисляется взвешенное среднее по всем известным исходным точкам по формуле указанной выше. При большом количестве исходных и расчётных точек, данная операция может быть достаточно ресурсоёмкой.

Если значение не интерполировалось и не экстраполировалось, то возвращается пользовательское значение по умолчанию. Программный комплекс FlowVision умеет обрабатывать особые значения – FLT_MAX. Если пользователю требуется не визуализировать область, которая не попадает в bBox из файла с данными, в качестве пользовательского значения можно задать значение `3.402823466e+38` – числовая константа, соответствующая максимальному значению, которое вмещает в себя число с плавающей точкой одинарной точности (float). Это же значение возвращает функция `fvapiGetVariableValue`.

5.2 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

Пользовательский модуль загружается во вкладке «Препроцессор» ППП.

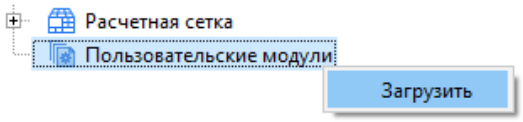

Рисунок 5.2.1 – Загрузка модуля

После загрузки модуля в ППП отображается информационное окно с описанием модуля

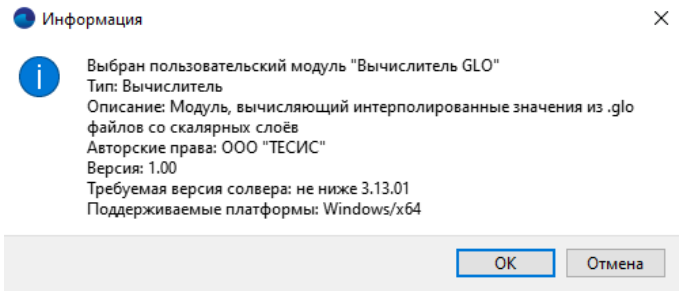

Рисунок 5.2.2 - Информационное окно

Модуль подключается в проекте в те места препроцессора, где можно задавать значения (пользовательские переменные, значения на ГУ и прочее).

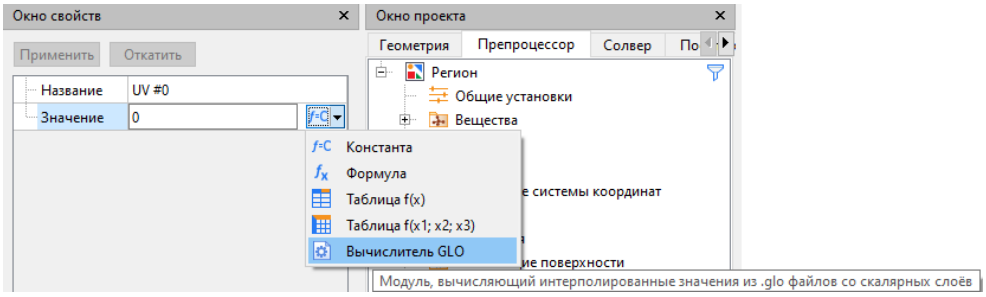

Рисунок 5.2.3 - Подключение модуля в качестве значения

Интерфейс DLL представлен на Рисунок 5.2.4 – Окно параметров модуля «Вычислитель GLO»Рисунок 5.2.4.

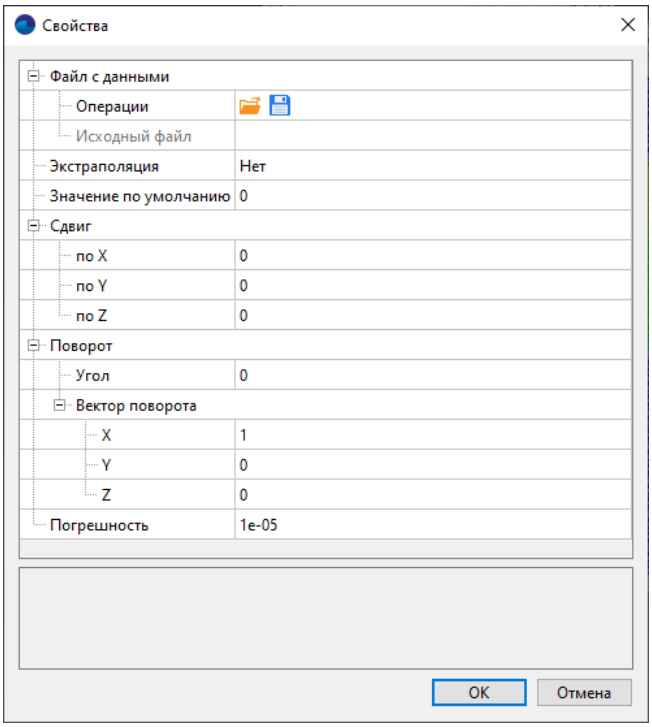

Рисунок 5.2.4 – Окно параметров модуля «Вычислитель GLO»

В параметрах пользовательского модуля необходимо задать файл с исходными данными о точках. Этот файл загружается в проект FlowVision и хранится в бинарном формате. Интерфейс Flowvision не предоставляет возможности открыть этот файл, однако его можно сохранить на диск в требуемом формате при помощи кнопки «Сохранить» для дальнейшего открытия сторонними программами. Подробнее о свойствах поля «Файл» можно прочитать в п. 1.6.

Дополнительно можно включить экстраполяцию (флаг «Экстраполяция»).

При отключённой экстраполяции, в точках, не попадающих в расчётный диапазон, возвращается пользовательское значение по умолчанию

Параметры «Сдвиг» и «Поворот» позволяют задать перемещение области из файла относительно системы координат проекта. Сдвиг задаётся смещением в метрах вдоль опорных осей. Поворот задаётся вектором поворота (с началом в нуле глобальной системы координат проекта) и углом вращения всех точек относительно этого вектора в градусах. При трансформации положения точек сначала производится поворот, а затем сдвиг.

Поле «Погрешность» задаёт допустимое отклонение расчётной точки от области, ограниченной исходными точками.

Для реализации интерфейса в файле **params.xml** задаётся разметка со следующими основными блоками xml-кода :

```
   . . .
<FVDLL_PARAMETERS>
  <Structure class="Translation">
    <Parameter type="Real" id="X" default="0.0">
    . . .
    <Parameter type="Real" id="Y" default="0.0">
    . . .
    <Parameter type="Real" id="Z" default="0.0">
```

```
      <Name xml:lang="en">Z</Name>
  </Structure>

  <Structure class="RotationVector">
    <Parameter type="Real" id="X" default="1.0">
    . . .
    <Parameter type="Real" id="Y" default="0.0">
    . . .
    <Parameter type="Real" id="Z" default="0.0">
    . . .
  </Structure>

  <Structure class="Rotation">
    <Parameter type="Real" id="Angle" default="0.0">
    . . .
    <Parameter type="Structure" class="RotationVector" id="RotationVector">
    . . .
  </Structure>

  <ParameterSet>
    <Parameter type="File" id="Filename">
    <Parameter type="Boolean" id="IsExtapolationOn" default="false">
    . . .
    <Parameter type="Real" id="DefaultValue" default="0.0">
    . . .
    <Parameter type="Structure" class="Translation" id="Translation">
    . . .
    <Parameter type="Structure" class="Rotation" id="Rotation">
    . . .
    <Parameter type="Real" id="Tolerance" default="0.00001">
    . . .
  </ParameterSet>
</FVDLL_PARAMETERS>
```

5.3 ПРОГРАММИРОВАНИЕ МОДУЛЯ

В модуле используются классы и функции из открытой библиотеки Geometric Tools Library (https://www.geometrictools.com/). Для сборки проекта необходимо указать путь к заголовочным файлам библиотеки вручную или создать переменную среды GTE_LIB, которая будет ссылаться на папку с этими заголовочными файлами. В настройках проекта для VisualStudio в качестве пути к дополнительным заголовочным файлам используется переменная GTE_LIB.

Как было рассмотрено в п. 1.4 и п. 1.8, в модуле типа Evaluator должны быть реализованы 4 функции, вызываемые Солвером. Две первые функции являются функциями инициализации. Для удобства, в примере определяется класс `EvaluatorGLO`. При инициализации, функция `fvdllInitContext` вызывается по количеству мест подключения модуля в проект. Соответственно, при каждом вызове в модуле создаётся объект класса `EvaluatorGLO`, которому в конструктор передаются параметры инициализации. Адрес созданного объекта передаётся в Солвер в качестве дескриптора контекста выполнения. id параметров, переданных в конструктор объекта, соответствуют именам в params.xml и их значения представлены в списке:

- `Filename`
- `IsExtapolationOn`
- `DefaultValue`
- `Translation/X`

- `Translation/Y`
- `Translation/Z`
- `Rotation/Angle`
- `Rotation/RotationVector/X`
- `Rotation/RotationVector/Y`
- `Rotation/RotationVector/Z`
- `Tolerance`

Рассмотрим алгоритм, реализованный в функции вычисления значения.

1. В функцию `fvdllEvaluateScalar` передаётся дескриптор контекста выполнения и дескриптор контекста вычисления значения.

2. По дескриптору контекста выполнения определяется объект, соответствующий конкретному экземпляру подключения модуля.

3. У найденного объекта вызывается соответствующий метод, в который передаётся дескриптор контекста вычисления значения.

4. Из дескриптора контекста вычисления значения с помощью функции `fvapiGetContextMask` запрашивается маска контекста, определяющая, какие данные пришли от Солвера.

5. Если расчёт производится в ячейке, то при помощи функции `fvapiGetFaceCenter` получаем координаты центра расчётного чипа (срабатывает, если модуль используется в качестве значения на ГУ). Если в контексте вычисления ячейка не задана, то при помощи функции `fvapiGetPointOfContext` получаем координаты расчётной точки.

6. Используя значения параметров, полученных при инициализации, по представленным формулам происходит вычисление итогового значения. В функции `fvdllReleaseContexts` удаляются все хранимые объекты

7. Модуль может кэшировать вычисленные значения, если расчёт производится в ячейке. Для своевременной очистки и инициализации кэша используются функции `fvdllBeforeTimeStep` и `fvdllAfterTimeStep`, которые вызываются до и после основных вычислений соответственно.

6. ПРИМЕР МОДУЛЯ ТИПА BINDER – МОДЕЛЬ АКТИВНОГО ДИСКА

6.1 ПОСТАНОВКА ЗАДАЧИ

В качестве примера пользовательского модуля типа Binder рассмотрим Модель активного диска. Данный модуль предназначен для моделирования вентиляторных (винтовых) устройств, для которых известна расходно-напорная характеристика, зависящая от частоты вращения.

Входными параметрами модуля являются частота вращения диска ω_{RPM} (об/мин) и таблица зависимости коэффициента упора K_T от относительной поступи J. Для удобства расчетов пересчитаем частоту вращения диска в оборотах в секунду (6.1).

$$\omega = \frac{\omega_{RPM}}{60} \tag{6.1}$$

Модуль запрашивает у Солвера площадь границы S, и по формуле (6.2) вычисляется диаметр активного диска.

$$D = \sqrt{\frac{4S}{\pi}} \tag{6.2}$$

Рассмотрим формулы для вычисления относительной скорости винта (6.3) и упора (6.4).

$$v_A = J\omega D \tag{6.3}$$

$$T = K_T \rho \omega^2 D^4, \tag{6.4}$$

где ρ – плотность, значение которой запрашивается у Солвера. Так как $K_T(J)$ задано таблично, получаем табличную зависимость $T(v_A)$. При запросе значения T осуществляется линейная интерполяция табличных значений.

Также, на первой границе запрашивается нормальная массовая скорость $(\rho V_n)_1$. Значение скорости на первой границе вычисляется по формуле (6.5).

$$v_A = \frac{(\rho V_n)_1}{\rho} \tag{6.5}$$

На первой границе установлено граничное условие «Давление на входе». Значение давления p_1 вычисляется по формуле (6.7).

$$T = (p_2 - p_1)S \tag{6.6}$$

$$p_1 = p_2 - \frac{T(v_A)}{S} \tag{6.7}$$

Значение давления на второй границе p_2 запрашивается у Солвера. На второй границе установлено граничное условие «Нормальная массовая скорость». Значение $(\rho V_n)_2$ переносится с первой границы (6.8).

$$(\rho V_n)_2 = (\rho V_n)_1 \tag{6.8}$$

6.2 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

В интерфейсе пользовательский модуль типа Binder позволяет создавать связку своего типа (рисунок 6.2.1).

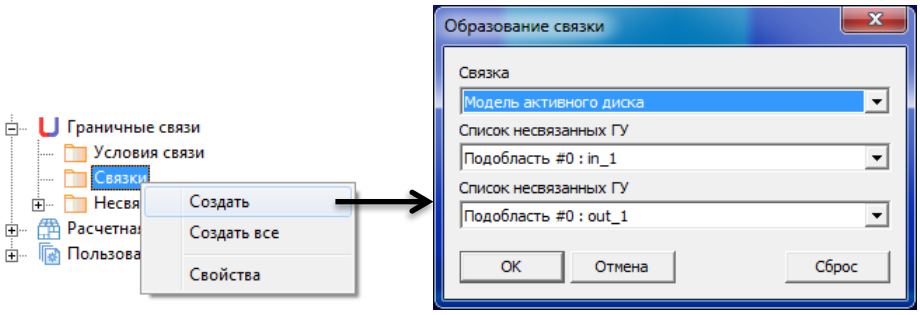

Рисунок 6.2.1 – Создание связки типа Модель активного диска

Выбор образованной связки в дереве препроцессора отображает окно свойств данной связки, в котором возможно редактирование параметров (рисунок 6.2.2).

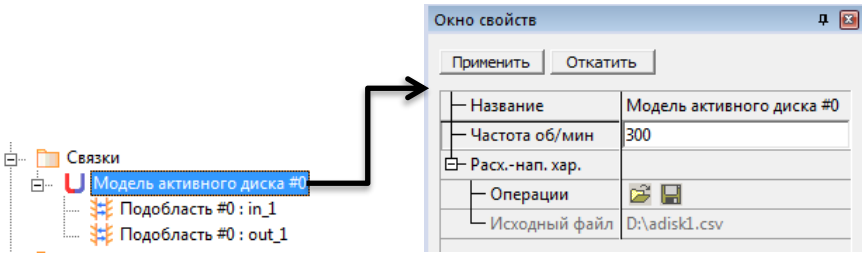

Рисунок 6.2.2 – Окно свойств связки

Первым параметром является название связки. Связку можно переименовывать. Следующие параметры – это параметры модуля, описание которых загружается из файла параметров модуля **param.xml**. В данном случае есть всего один параметр типа вещественное число, задающий частоту вращения активного диска, и параметр типа файл, позволяющий загрузить расходно-напорную характеристику из файла. Рассмотрим xml-код, определяющий такой набор параметров.

```
<FVDLL_PARAMETERS>
  <ParameterSet>
    <Parameter type="Real" id="Frequency" default="1.0">
      <Name xml:lang="en">Frequency rpm</Name>
      <Name xml:lang="ru">Частота об/мин</Name>
      <Description xml:lang="en">Rotation frequency, revolution per minute</Description>
      <Description xml:lang="ru">Частота вращения, оборотов в минуту</Description>
    </Parameter>
    <Parameter type="File" id="FlowPress">
```

```
        <Name xml:lang="en">Flow-press. char.</Name>
        <Name xml:lang="ru">Расх.-нап. хар.</Name>
        <Description xml:lang="en">Flow-pressure characteristic</Description>
        <Description xml:lang="ru">Расходно-напорная характеристика</Description>
    </Parameter>
  </ParameterSet>
</FVDLL_PARAMETERS>
```

Расходно-напорная характеристика представляет собой таблицу в csv-формате. Таблица имеет два столбца (рисунок 6.2.3):

- J – относительная поступь;
- K_T – коэффициент упора.

	A	B	C
1	J	Kt	
2	0	0.448763	
3	0.1	0.420739	
4	0.2	0.388007	
5	0.3	0.35115	
6	0.4	0.310747	
7	0.5	0.267377	
8	0.6	0.22162	
9	0.7	0.174053	
10	0.8	0.125257	
11	0.9	0.075811	
12	1	0.026293	
13	1.1	-0.02272	
14			

Рисунок 6.2.3 – Расходно-напорная характеристика

Также возможно создание условия связи типа пользовательского модуля (рисунок 6.2.4).

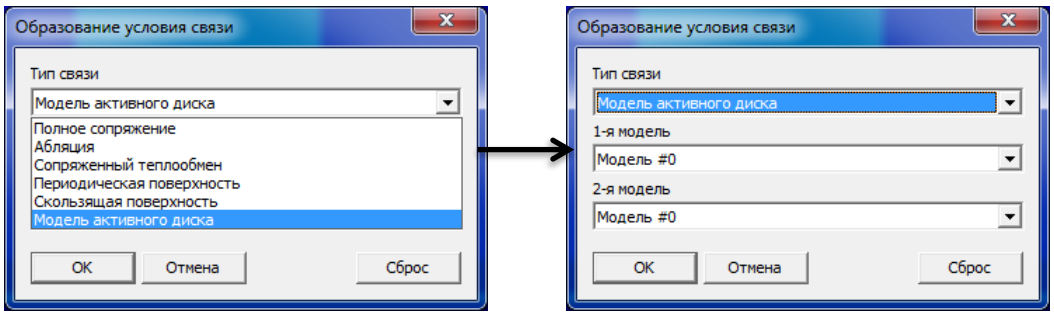

Рисунок 6.2.4 – Образование условия связи типа Модель активного диска

При этом в интерфейсе отображаются переменные, которые связывает данный пользовательский модуль. Модель активного диска может связывать следующие переменные:

- скорость;
- температура.

Список связываемых переменных задаётся в файле описания модуля **main.xml** в разделе специальных параметров, зависящих от типа модуля. Для модуля активного диска специальные параметры определяются следующим xml-кодом:

```
  <SpecParams>
    <ConnectedVariable>TEMPERATURE</ConnectedVariable>
    <ConnectedVariable>VELOCITY</ConnectedVariable>
```

```
    <BoundaryCondition1>
        <SuperType>FREE_OUTLET</SuperType>
        <TEMPERATURE>ZEROGRAD</TEMPERATURE>
        <VELOCITY>OUTLET</VELOCITY>
    </BoundaryCondition1>
    <BoundaryCondition2>
        <SuperType>INLET_OUTLET</SuperType>
        <TEMPERATURE>CONST</TEMPERATURE>
        <VELOCITY>MASSNORMAL</VELOCITY>
    </BoundaryCondition2>
  </SpecParams>
```

Связываемые переменные также отображаются в интерфейсе (рисунок 6.2.5).

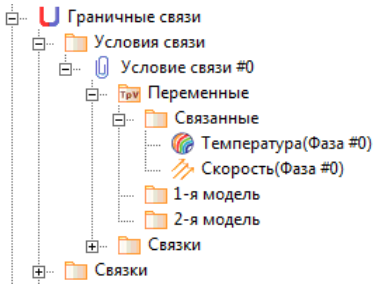

Рисунок 6.2.5 – Связанные переменные в интерфейсе

6.3 ПРОГРАММИРОВАНИЕ МОДУЛЯ

В модуле типа Binder, в соответствии с пунктами 1.4 и 1.9, должны быть реализованы 5 функций, вызываемых Солвером. Функции `fvdllInitModule`, `fvdllInitContext` и `fvdllInitBinder` являются функциями инициализации. Для удобства, в примере определяется класс `CActiveDisk`. При инициализации, функция `fvdllInitContext` вызывается по количеству созданных в проекте связок типа Модель активного диска. Соответственно, при каждом вызове в модуле создаётся объект класса `CActiveDisk`, которому в конструктор передаются параметры инициализации. Адрес созданного объекта передаётся в Солвер в качестве *дескриптора контекста выполнения*. **id** параметров, переданных в конструктор объекта, соответствуют именам в **params.xml** и их значения представлены в списке:

- `Frequency`
- `FlowPress`

Значение частоты передаётся в виде вещественного числа, а расходно-напорная характеристика – в виде указателя на структуру типа `FVDLL_PARAMETER_FILE`. Так как csv-файл имеет текстовый формат, из поля `data` данной структуры возможно сконструировать объект класса `stringstream` стандартной библиотеки C++. Чтение данных из полученного объекта выполняется аналогично чтению из файлового потока.

Так же, как и функция `fvdllInitContext`, по количеству созданных в проекте связок типа Модель активного диска вызывается фунция `fvdllInitBinder`. В неё передаются дескриптор контекста выполнения и два дескриптора связываемых граничных условий. По дескриптору контекста выполнения осуществляется поиск соответствующего объекта класса `CActiveDisk`, у которого вызывается метод `InitDisk`. В данном методе объект запрашивает через API ВИП дескрипторы подобластей, которых он связывает, и дескрипторы переменных скорости, давления и плотности.

В функции 2-4 интерфейса Binder (п. 1.9) так же, как и в `fvdllInitContext`, передаётся дескриптор контекста выполнения, и у соответствующего объекта класса `CActiveDisk` вызываются методы `BeforeTimeStep, AfterTimeStep, GetValueInContext`.

Метод `BeforeTimeStep` в данном примере ничего не делает, так как весь алгоритм выполняется в `AfterTimeStep`.

Метод `AfterTimeStep` реализует основной алгоритм модуля, который был подробно рассмотрен в п. 6.1. Искомые нормальная массовая скорость на выходе из активного диска и давление на входе в активный диск сохраняются в полях объекта.

Метод `GetValueInContext` принимает в аргументах дескриптор переменной и по нему определяет, какое из вычисленных ранее в методе `AfterTimeStep` значений необходимо вернуть: нормальную массовую скорость на выходе из активного диска или давление на входе в активный диск.

В функции `fvdllReleaseContexts` удаляются все хранимые объекты.

7. ПРИМЕР МОДУЛЯ ТИПА BOUNDARYCONDITION – ГЕНЕРАТОР ВОЛН

7.1 ПОСТАНОВКА ЗАДАЧИ

В качестве примера пользовательского модуля типа *BoundaryCondition* рассмотрим генератор волн.

Высота свободной поверхности η_h определяется формулой (7.1), горизонтальная скорость u_h – формулой (7.2), а вертикальная w_h – формулой (7.3).

$$\eta_h = \frac{H}{2}\cos(kx - \sigma t) + \frac{H^2 k}{16}\frac{\cosh kh}{\sinh^3 kh}(2 + \cosh 2kh)\cos 2(kx - \sigma t) \tag{7.1}$$

$$u_\mathrm{h} = \frac{\mathrm{H}}{2}\frac{\mathrm{gk}}{\sigma}\frac{\cosh \mathrm{k(h+z)}}{\cos \mathrm{kh}}\cos(\mathrm{kx} - \sigma\mathrm{t}) + \frac{3}{16}\frac{\mathrm{H}^2\sigma\mathrm{k}\cosh 2\mathrm{k(h+z)}}{\sinh^4 \mathrm{kh}}\cos 2(\mathrm{kx} - \sigma\mathrm{t}) \tag{7.2}$$

$$w_h = \frac{H}{2}\frac{gk}{\sigma}\frac{\sinh k(h+z)}{\cos kh}\sin(kx - \sigma t) + \frac{3}{16}\frac{H^2\sigma k \sinh 2k(h+z)}{\sinh^4 kh}\sin 2(kx - \sigma t) \tag{7.3}$$

В данных формулах применяются следующие обозначения:

- H – высота волны,
- k – волновое число,
- x – горизонтальная координата,
- z – вертикальная координата,
- t – время,
- σ – угловая частота, $\sigma = \frac{2\pi}{T}$, где T – период,
- g – ускорение свободного падения,
- h – глубина.

Для глубокой воды найдём пределы определённых выше величин при $h \to \infty$:

$$\eta = \lim_{h\to\infty}\eta_h = \frac{H}{2}\cos(kx - \sigma t) + \frac{H^2 k}{16}2\cos 2(kx - \sigma t) \tag{7.4}$$

$$u = \lim_{h\to\infty}u_h = \frac{H}{2}\frac{gk}{\sigma}e^{kz}\cos(kx - \sigma t) \tag{7.5}$$

$$w = \lim_{h\to\infty}w_h = \frac{H}{2}\frac{gk}{\sigma}e^{kz}\sin(kx - \sigma t) \tag{7.6}$$

Пользователь в интерфейсе задаёт период колебания T, высоту волны H, направление $\vec{d}$, опорную точку с координатами $(r_x, r_y, r_z) = \vec{r}$, постоянную скорость $\vec{v}$.

В процессе расчета у пользовательского модуля запрашиваются значения VOF и вектора скорости. В свою очередь, модуль получает через API координаты центра текущей грани $(c_x, c_y, c_z) = \vec{c}$, вектор ускорения свободного падения $\vec{g}$, текущее время t.

Для координат центра грани применяются поправки в соответствии с постоянной скоростью:

$$\vec{c} \leftarrow \vec{c} - \vec{v}t \tag{7.7}$$

Находим горизонтальную координату x как расстояние между текущей точкой и плоскостью, для которой вектор направления распространения волны является нормалью (7.8).

$$x = \frac{\langle \vec{d} | \vec{c} \rangle - \langle \vec{d} | \vec{r} \rangle}{|\vec{d}|} \tag{7.8}$$

Находим вертикальную координату z как расстояние между текущей точкой и плоскостью, для которой вектор гравитации является нормалью (7.9).

$$z = -\frac{\langle \vec{g} | \vec{c} \rangle - \langle \vec{g} | \vec{r} \rangle}{|\vec{g}|} = \frac{\langle \vec{g} | \vec{r} \rangle - \langle \vec{g} | \vec{c} \rangle}{|\vec{g}|} \tag{7.9}$$

Волновое число k находим по формуле (7.10).

$$k = \frac{\sigma^2}{g} \tag{7.10}$$

Вычисляем η, u, w по формулам (7.4) – (7.6).

Значение VOF и $\overrightarrow{Velocity}$, запрошенные у модуля, будут рассчитаны по формулам (7.11) и (7.12).

$$VOF = \begin{cases} 1, \text{если } z \le \eta, \\ 0, \text{если } z > \eta. \end{cases} \tag{7.11}$$

$$\overrightarrow{Velocity} = \begin{cases} u\vec{d} - w\vec{g} + \vec{v}, \text{если } z \le \eta, \\ 0, \qquad\qquad \text{если } z > \eta. \end{cases} \tag{7.12}$$

7.2 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

В интерфейсе пользовательский модуль типа *BoundaryCondition* позволяет создавать ГУ своего типа (рисунок 7.2.1).

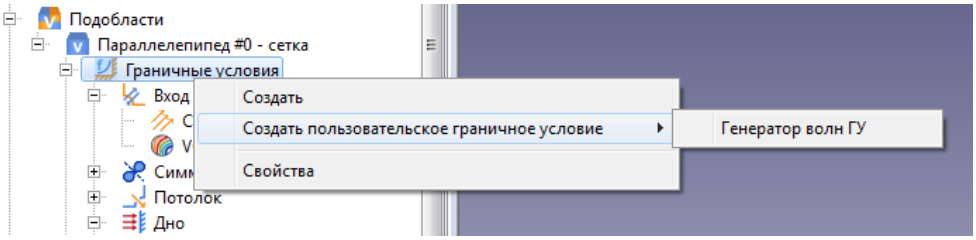

Рисунок 7.2.1 – Создание пользовательского ГУ

Выбор созданного ГУ в дереве препроцессора отображает окно свойств данного ГУ, в котором возможно редактирование параметров (рисунок 7.2.2).

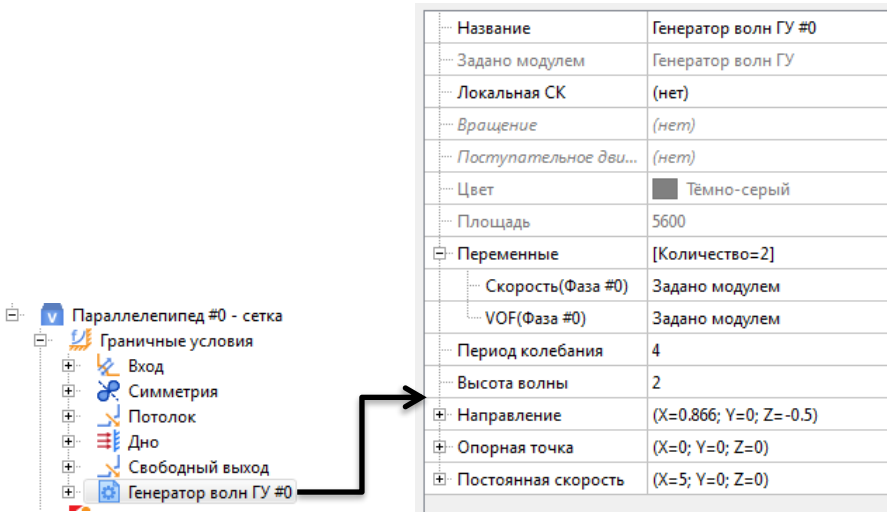

Рисунок 7.2.2 – Окно свойств Генератора волн

Параметры «Период колебания», «Высота волны», «Направление», «Опорная точка» и «Постоянная скорость» определены в файле **params.xml** и выводятся в интерфейс для редактирования пользователем. Файл **params.xml**, в свою очередь, имеет следующее содержимое:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<FVDLL_PARAMETERS>
  <Structure class="StructXYZ">
    <Parameter type="Real" id="X" default="1.0">
      <Name xml:lang="en">X</Name>
      <Name xml:lang="ru">X</Name>
      <Description xml:lang="en">X</Description>
      <Description xml:lang="ru">X</Description>
    </Parameter>
    <Parameter type="Real" id="Y" default="1.0">
      <Name xml:lang="en">Y</Name>
      <Name xml:lang="ru">Y</Name>
      <Description xml:lang="en">Y</Description>
      <Description xml:lang="ru">Y</Description>
    </Parameter>
    <Parameter type="Real" id="Z" default="1.0">
      <Name xml:lang="en">Z</Name>
      <Name xml:lang="ru">Z</Name>
      <Description xml:lang="en">Z</Description>
      <Description xml:lang="ru">Z</Description>
    </Parameter>
  </Structure>
  <ParameterSet>
    <Parameter type="Real" id="Period" default="2.0">
      <Name xml:lang="en">Period</Name>
      <Name xml:lang="ru">Период колебания</Name>
      <Description xml:lang="en">Period</Description>
      <Description xml:lang="ru">Период колебаний</Description>
    </Parameter>
    <Parameter type="Real" id="High" default="2.0">
      <Name xml:lang="en">High</Name>
```

```
      <Name xml:lang="ru">Высота волны</Name>
      <Description xml:lang="en">High</Description>
      <Description xml:lang="ru">Высота волны</Description>
    </Parameter>
    <Parameter type="Structure" class="StructXYZ" id="Direction">
      <Name xml:lang="en">Direction</Name>
      <Name xml:lang="ru">Направление</Name>
      <Description xml:lang="en">Wave direction</Description>
      <Description xml:lang="ru">Направление распространения волны</Description>
    </Parameter>
    <Parameter type="Structure" class="StructXYZ" id="RefPoint">
      <Name xml:lang="en">Reference point</Name>
      <Name xml:lang="ru">Опорная точка</Name>
      <Description xml:lang="en">Reference point</Description>
      <Description xml:lang="ru">Опорная точка</Description>
    </Parameter>
    <Parameter type="Structure" class="StructXYZ" id="ConstVel">
      <Name xml:lang="en">Constant velocity</Name>
      <Name xml:lang="ru">Постоянная скорость</Name>
      <Description xml:lang="en">Constant velocity</Description>
      <Description xml:lang="ru">Постоянная скорость</Description>
    </Parameter>
  </ParameterSet>
</FVDLL_PARAMETERS>
```

Также в интерфейсе (рисунок 7.2.2) показано, что переменные *VOF* и *Скорость* заданы модулем. Фрагмент файла **main.xml**, определяющий это, представлен ниже:

```
<SpecParams>
  <Template>INLET_OUTLET</Template>
  <Variable>
    <IDVar>VEL</IDVar>
    <BaseClass>CBVelVelocityFixed</BaseClass>
  </Variable>
  <Variable>
    <IDVar>VOF</IDVar>
    <BaseClass>CBVarConstant</BaseClass>
  </Variable>
</SpecParams>
```

Для данного пользовательского модуля установлен шаблон «Вход/Выход». Скорость задаётся с помощью граничного условия «Фиксированная скорость» (CBVelVelocityFixed), рассмотренного в п. 2.4.61. У данного условия есть один векторный параметр – «Скорость», значение которого будет запрашиваться у пользовательского модуля. VOF задаётся с помощью граничного условия «Значение» (CBVarConstant), рассмотренного в п. 2.4.6. У данного условия есть скалярный параметр – «Значение», который запрашивается у пользовательского модуля.

7.3 ПРОГРАММИРОВАНИЕ МОДУЛЯ

В модуле типа *BoundaryCondition*, в соответствии с пунктами 1.4 и 1.10, должны быть реализованы 5 функций, вызываемых Солвером. Функции `fvdllInitModule`, `fvdllInitContext` и `fvdllInitParameterContext` являются функциями инициализации. Для удобства, в примере определяется класс `CWaveGeneratorBC`. При инициализации, функция `fvdllInitContext` вызывается по количеству созданных в проекте ГУ типа Генератор волн ГУ. Соответственно, при

каждом вызове в модуле создаётся объект класса `CWaveGeneratorBC`, которому в конструктор передаются параметры инициализации. Адрес созданного объекта передаётся в Солвер в качестве *дескриптора контекста выполнения*. **id** параметров, переданных в конструктор объекта, соответствуют именам в **params.xml**, и их значения представлены в списке:

- `Period`
- `High`
- `Direction/X`
- `Direction/Y`
- `Direction/Z`
- `RefPoint/X`
- `RefPoint/Y`
- `RefPoint/Z`
- `ConstVel/X`
- `ConstVel/Y`
- `ConstVel/Z`

Метод `fvdllInitParameterContext` вызывается по количеству параметров граничных условий, которыми управляет пользовательский модуль. В аргументах передаётся дескриптор переменной и строковой идентификатор параметра. Если переменная VOF и параметр «WallValue», то в качестве дескриптора метод возвращает указатель на функцию `GetVOFWrapper`. Если переменная скорость и параметр имеет строковой идентификатор «VelVectorFixed», возвращается указатель на функцию `GetVelocityFixedWrapper`.

Метод `fvdllGetValueInContext` принимает в аргументах дескриптор контекста выполнения, дескриптор контекста запроса параметра и дескриптор запроса значения. Дескриптор контекста выполнения в модуле преобразуется в указатель на соответствующий объект класса `CWaveGeneratorBC`. Дескриптор контекста запроса параметра – в указатель на соответствующую функцию. А дескриптор запроса значения нужен для получения с помощью API ВИП функций информации о грани, на которой значение было запрошено.

В функции `fvdllReleaseContexts` удаляются все хранимые объекты.

8. ОПИСАНИЕ ПРОТОКОЛА СВЯЗИ С СОЛВЕР-АГЕНТОМ

8.1 ОБЩАЯ ИНФОРМАЦИЯ

Протокол связи с солвер-агентом предназначен для передачи солвер-агенту команд, позволяющих создавать и модифицировать проекты FlowVision, запускать их на счет и получать результаты. Структурно протокол состоит из двух уровней – коммуникационного и прикладного.

8.2 КОММУНИКАЦИОННЫЙ УРОВЕНЬ ПРОТОКОЛА

Коммуникационный уровень позволяет передавать по сетевому каналу TCP/IP команды и файлы на солвер-агент и получать их с солвер-агента.

На коммуникационном уровне данные передаются через сетевой канал TCP/IP в следующем формате:

[*type***][***b_size***]** *text_command* **[***UUID***] LF** *binary_data*

- *type* – тип передаваемых данных.
  - **C** – текстовая команда
  - **D** – текстовая команда, передаваемая в бинарном виде
  - **F** – файл
- *b_size* – размер бинарных данных (*binary_data*), передаваемых вместе с командой (в байтах). Если бинарные данные не передаются, *b_size*=0.
- *text_command* – текстовая команда.
- *UUID* – уникальный идентификатор команды.
- *LF* – символ перевода строки (код 10).
- *binary_data* – бинарные данные, передаваемые вместе с командой (присутствуют, если *b_size*>0 или если *type*='D')

При передаче команды в бинарном виде (*type*=**D**) в поле *text_command* указывается размер команды в бинарном виде (*binary_data*) в байтах

При передаче файла (*type*=**F**) поле *text_command* имеет следующий вид: *<file_name><block_ofs>*
где *file_name* - имя файла (без пути), под которым файл будет сохранен на принимающей стороне *block_ofs* – смещение данного блока файла относительно начала (при пересылке файл может быть разбит на блоки и посылаться несколькими командами, при этом в первой команде *block_ofs* будет равняться 0).

При приеме команды коммуникационный уровень передает её на прикладной для распознавания и выполнения соответствующего действия вместе с аргументами и бинарными данными.

При приеме файла коммуникационный уровень самостоятельно сохраняет его в заранее определенную директорию для принятых файлов, не извещая прикладной уровень.

8.3 ПРИКЛАДНОЙ УРОВЕНЬ ПРОТОКОЛА

Прикладной уровень протокола представлен командами, которые посылаются солвер-агенту для выполнения тех или иных действий. В ответ на каждую команду солвер-агент присылает ответ-подтверждение с кодом результата, по которому можно понять, выполнена ли команда успешно или с ошибкой. После посылки команды на солвер-агент нужно обязательно дождаться ответа и только после этого посылать следующую команду.

Существуют также команды, посылаемые солвер-агентом на клиент самостоятельно. Они как правило не требуют ответа. Такие команды описаны ниже в отдельном разделе.

Текст принятой команды солвер-агент записывает в свой лог-файл (если у него включена запись отладочных сообщений). Это нужно иметь в виду при передаче на солвер-агент конфиденциальной информации.

Ниже описаны форматы текстовых команд, указываемых в поле *text_command* вышеописанного коммуникационного уровня. Бинарные данные во всех перечисленных ниже командах отсутствуют.

8.3.1 Команды общего назначения

8.3.1.1 КОМАНДА АВТОРИЗАЦИИ НА СОЛВЕР-АГЕНТЕ

Данная команда посылается самой первой после установления соединения с солвер-агентом и необходима для авторизации на солвер-агенте под конкретным пользователем FlowVision. Все посылаемые далее команды будут выполняться для этого пользователя (если имя пользователя явно не указано в команде).

- Формат команды:
  SA_CONNECT <USERNAME><PASSWORD>
- Формат ответа:
  SA_CONNECTED <RESULT><EnabledSolverModes><SolverCmdLines>
  <SolverCmdLinesMPI><SolverProxyHost><SolverProxyPort>
  RESULT – результат:

    0 – все нормально

    1 – ошибка (неверный логин или пароль)

  EnabledSolverModes – возможные режимы запуска солвера (битовая маска):

    Бит 0 – возможен однопроцессорный запуск

    Бит 1 – возможен многопроцессорный запуск через MPI

  SolverCmdLines – Список описаний командных строк солвера (для однопроцессорного запуска), разделенных точкой с запятой
  SolverCmdLinesMPI – Список описаний командных строк солвера (для многопроцессорного запуска), разделенных точкой с запятой
  SolverProxyHost – IP адрес ретранслятора, используемого для доступа к солверам данного солвер-агента. Если данный параметр пуст, ретранслятор не используется

SolverProxyPort – номер порта ретранслятора, используемого для доступа к солверам данного солвер-агента. Если данный параметр пуст, ретранслятор не используется.

8.3.1.2 КОМАНДА ПОЛУЧЕНИЯ СЕРВЕРНЫХ ДИРЕКТОРИЙ

Команда позволяет получить с солвер-агента список серверных директорий для указанного пользователя, разделенные точкой с запятой.

- Формат команды:
  SA_GETUSERDIRS <USERNAME><PASSWORD>
- Формат ответа:
  SA_ USERDIRSGETTED <RESULT><WORKING_DIR>
  RESULT – результат:

  0 – все успешно

  1 – если такого пользователя нет или неверный пароль

8.3.2 Команды для работы с проектами

8.3.2.1 КОМАНДА СОЗДАНИЯ ДИРЕКТОРИИ ПРОЕКТА

Данная команда позволяет создать новую проектную директорию в серверной директории указанного пользователя. Если такая директория уже есть, на конце её имени добавляется суффикс «_1», «_2», «_3» и так далее. Окончательное имя созданной директории возвращается в ответе на команду.

- Формат команды:
  SA_PROJECTDIRCREATE <USERNAME ><Project dir name>

  USERNAME – имя пользователя
  Project dir name – имя создаваемой директории

- Формат ответа:
  SA_ PROJECTDIRCREATED <RESULT><Project dir name>
  RESULT – результат операции
  0 – операция выполнена успешно
  1 – ошибка получения серверной директории пользователя
  2 – ошибка при создании директории
  Project dir name – имя созданной директории с полным путем (если все прошло успешно).

8.3.2.2 КОМАНДА ПОМЕЩЕНИЯ ФАЙЛА В ПРОЕКТ

Данная команда позволяет поместить указанный файл в указанный проект. Файл должен быть переслан на солвер-агент на коммуникационном уровне протокола до посылки данной команды. В команде указывается имя файла, под которым он был переслан на солвер-агент, и имя файла, под которым он должен быть помещен в проект.

- Формат команды:
  SA_PROJECTFILESET <PROJECT_ID><Project path><Sent file name>
  <File name>

PROJECT_ID – идентификатор проекта, в который помещается файл
File path – директория проекта (указывается, если не указан PROJECT_ID)
Sent file name – имя, под которым файл был переслан на солвер-агент
File name – имя, под которым файл должен быть сохранен в проекте

- Формат ответа:
  SA_PROJECTFILESETTED <RESULT>
  RESULT – результат операции
  0 – операция выполнена успешно
  1 – неправильный идентификатор проекта (нет такого проекта)
  2 – не найден пересланный файл
  3 – ошибка копирования файла в директорию проекта

8.3.2.3 КОМАНДА ПОЛУЧЕНИЯ ФАЙЛА ИЗ ПРОЕКТА

Данная команда позволяет получить указанный файл из указанного проекта. Собственно передача файла выполняется коммуникационным уровнем протокола до прихода ответа на команду, при этом имя файла, под которым он был переслан с солвер-агента, сообщается в ответе на команду.

Данная команда может также использоваться для проверки наличия в директории проекта определенного файла.

Если необходимо скачать одновременно несколько файлов из проекта, целесообразнее воспользоваться командой **SA_**PROJECTGET с соответствующей маской.

- Формат команды:
  SA_PROJECTFILEGET <PROJECT_ID><File name>
  PROJECT_ID – идентификатор проекта
  File name – имя запрашиваемого файла

- Формат ответа:
  SA_PROJECTFILEGETTED <RESULT><File name>
  RESULT – результат операции
  0 – операция выполнена успешно
  1 – неправильный идентификатор проекта (нет такого проекта)
  2 – не найден указанный файл
  3 – ошибка посылки файла
  File name – имя принятого файла

8.3.2.4 КОМАНДА УДАЛЕНИЯ ПРОЕКТА

Команда позволяет удалить проект с указанным идентификатором (файлы и директория проекта физически удаляются с диска).

- Формат команды:
  SA_PROJECTDELETE <PROJID>
  PROJID – уникальный идентификатор проекта
- Формат ответа:
  SA_PROJECTDELETED <RESULT>
  RESULT – результат операции
  0 – удаление прошло успешно
  1 – проект с таким ID отсутствует в списке проектов
  2 – файлы удалены, директория не удалена

3 – не все файлы удалены
4 – директория проекта отсутствует или нет прав на ее модификацию
5 – некорректное состояние проекта (должно быть default)

8.3.2.5 КОМАНДА ПОЛУЧЕНИЯ ИНФОРМАЦИИ О ПРОЕКТЕ

Команда позволяет информацию об указанном проекте.

- Формат команды:
  SA_PROJECTINFOGET <ProjectID>
- Формат ответа:
  SA_PROJECTINFOGETTED <RESULT><PROJ_INFO>
  RESULT - результат операции (0 – все прошло успешно, 1 – не найден проект с указанным ID в списке проектов)
  PROJ_INFO – параметры проекта, разделенные точкой с запятой:
  - имя проектного файла с полным путем
  - состояние проекта проекта (см. описание команды SA_PROJECTLISTGET)
  - дополнительное состояние проекта (см. описание команды SA_ PROJECTLISTGET)
  - ID солвера, если проект загружен или считается
  - Plugin ID (идентификатор плагина, с помощью которого был создан проект)
  - Дата и время модификации файла fvproj
  - Количество шагов нестационарной записи (если нестационарная запись выключена, передается -1)
  - Объем проекта в Мб
  - Название коннектора, если проект используется для совместного расчета

8.3.2.6 КОМАНДА ПОЛУЧЕНИЯ СОСТОЯНИЯ ПРОЕКТА

Команда позволяет получить информацию о состоянии указанного проекта.

- Формат команды:
  SA_PROJECTSTATEGET <PROJID>
  PROJID – уникальный идентификатор проекта
- Формат ответа:
  SA_PROJECTSTATEGETTED <RESULT><PROJECT_STATE><PROJECT_SUBSTATE><SOLVER _ID>
  RESULT – результат операции (0 – все прошло успешно, 1 – не найдено проекта с таким ID в списке проектов)
  PROJECT_STATE – состояние проекта (см. описание команды SA_ PROJECTLISTGET)
  PROJECT_SUBSTATE – дополнительное состояние проекта (см. описание команды SA_ PROJECTLISTGET)
  SOLVER _ID – ID солвера, если проект загружен или считается, иначе пустая строка

8.3.2.7 КОМАНДА ПОЛУЧЕНИЯ СПИСКА ПРОЕКТОВ

Команда позволяет получить список проектов для указанного пользователя.

- Формат команды:
  SA_PROJECTLISTGET <USERNAME>
- Формат ответа:
  SA_PROJECTLISTGETTED <RESULT><PROJ_INFO><PROJ_INFO>…<PROJ_INFO>
  RESULT - результат операции (0 – все прошло успешно, 1 – не найдено проектов для

данного пользователя)
PROJ_INFO – параметры проекта, разделенные точкой с запятой:
- Project ID
- имя проектного файла с полным путем
- состояние проекта
- дополнительное состояние проекта
- ID солвера, если проект загружен или считается
- Plugin ID (идентификатор плагина, с помощью которого был создан проект)
- Дата и время модификации файла fvproj
- Количество шагов нестационарной записи (если нестационарная запись выключена, передается -1)
- Объем проекта в Мб
- Название коннектора, если проект используется для совместного расчета

Возможные состояния проекта:
- 0 – default
- 1 – загружен
- 2 – считается
- 3 – в очереди

Возможные дополнительные состояния проекта (уточняют основное состояние):
- 0 – default
- 1 – проект не сохранен (только для статуса «загружен»)
- 2 – проект сохранен (только для статуса «загружен»)

8.3.2.8 КОМАНДА СКАЧИВАНИЯ ПРОЕКТА С СОЛВЕР-АГЕНТА

Команда позволяет получить с солвер-агента файлы указанного проекта по заданной маске. Прием файлов выполняется на коммуникационном уровне, ответ на команду приходит по окончании передачи последнего файла.

Так как скачивание проекта может занимать продолжительное время, клиентскому приложению дается возможность вывода на экран индикатора прогресса, для чего солвер-агентом в процессе скачивания посылаются специальные команды: SA_PROGRESSBARSHOW с указанием максимального и минимального значений индикатора прогресса – перед началом пересылки файлов; SA_PROGRESSBARMOVE с указанием текущего значения прогресса – после пересылки каждого файла (для обновления индикатора прогресса); SA_PROGRESSBARHIDE – по окончании пересылки файлов для закрытия индикатора прогресса. Также клиентскому приложению дается возможность прерывать процесс скачивания путем посылки команды SA_PROJECTGETCANCEL. Скачивание будет прервано после передачи очередного файла.

- Формат команды:
  Формат команды:
  SA_PROJECTGET <PROJID><Files_Mask>
  PROJID – уникальный идентификатор проекта.
  Files_Mask – список масок файлов, подлежащих передаче с солвер-агента, разделенных точкой с запятой. Если список масок пуст, возвращается стандартный набор файлов (*.fvproj;*.fvinp;*.fvctrl;*.fvgeom;*.fvgobj;*.fvbcs;*.fvstat;*.fvview;*.log)
- Формат ответа:
  SA_PROJECTGETTED <RESULT>
  RESULT – результат операции
  0 – операция прошла успешно
  1 –указан некорректный project ID

2 – директория проекта отсутствует
3 – директория проекта недоступна для чтения

8.3.2.9 КОМАНДА ПРЕРЫВАНИЯ СКАЧИВАНИЯ ПРОЕКТА

Команда посылается на совер-агент для того, чтобы прервать процесс скачивания проекта, начатый командой SA_PROJECTGET. Обычно эта команда посылается в случае нажатия пользователем кнопки «Отмена» в индикаторе прогресса, который он видит при скачивании проекта.

- Формат команды:
  SA_PROJECTGETCANCEL

- Формат ответа:
  ответ не предусмотрен

8.3.2.10 КОМАНДА ДОБАВЛЕНИЯ ПРОЕКТА В ОЧЕРЕДЬ ПРОЕКТОВ

Данная команда добавляет указанный проект в очередь для последующего его запуска на расчет с указанными параметрами..

- Формат команды:
  SA_PROJECTQUEUEAPPEND < PROJECT_ID
  ><ProcNum><ThreadNum><CmdLineIndex><CommandFile>
  PROJECT_ID – идентификатор проекта
  ProcNum - количество процессоров для запуска на расчет (значение 0 означает однопроцессорный запуск без MPI). Для запуска через очередь в эксклюзивном режиме, перед числом, обозначающим количество процессоров, ставится звездочка (символ *).
  ThreadNum - количество нитей солвера для запуска на расчет (значение 0 означает количество нитей, равное количеству ядер процессора)
  CmdLineIndex – индекс командной строки. Должен соответствовать номеру командной строки (начиная с 0) из списка, полученного в качестве ответа на команду SA_CONNECT для соответствующего типа запуска (одно- или многопроцессорного)
  CommandFile – имя командного файла (файл должен быть предварительно переслан на солвер). Если данный параметр пуст – будет использован командный файл по умолчанию из директории проекта (*command.txt*)

- Формат ответа:
  SA_ PROJECTQUEUEAPPENDED<RESULT>
  RESULT – результат операции
  0 – операция выполнена успешно
  1 – некорректное состояние проекта (должно быть default)
  2 – не удалось скопировать командный файл
  3 – отсутствует командный файл по умолчанию
  4 – указанное количество процессоров больше максимально допустимого
  5 – проект с указанным ID отсутствует

8.3.2.11 КОМАНДА ПОЛУЧЕНИЕ СПИСКА ПРОЕКТОВ В ОЧЕРЕДИ

Данная команда позволяет получить список проектов, находящихся в очереди.

- Формат команды:
  SA_PROJECTQUEUEGET

- Формат ответа:
  SA_PROJECTQUEUEGETTED <Count><PI 1><PI 2>....<PI n>
  Count – количество проектов в очереди
  PI – параметры проекта, разделенные точкой с запятой
  a. Project ID
  b. Имя проекта с полным путем
  c. Дата и время помещения в очередь
  d. Имя пользователя
  e. Тип солвера для запуска
  f. Количество процессоров

8.3.2.12 КОМАНДА УДАЛЕНИЯ ПРОЕКТА ИЗ ОЧЕРЕДИ

Данная команда позволяет удалить указанный проект из очереди проектов.

- Формат команды:
  SA_PROJECTQUEUEDELETE < PROJECT_ID >
  PROJECT_ID – идентификатор проекта

- Формат ответа:
  SA_ PROJECTQUEUEDELETED <RESULT>
  RESULT – результат операции
  0 – операция выполнена успешно
  1 – нет проекта с таким ID в очереди проектов
  2 – проект был помещен в очередь другим пользователем

8.3.2.13 КОМАНДА ОЧИСТКИ ОЧЕРЕДИ ПРОЕКТОВ

Данная команда позволяет удалить из очереди все проекты, помещенные туда текущим пользователем.

- Формат команды:
  SA_PROJECTQUEUECLEAR

- Формат ответа:
  SA_ PROJECTQUEUECLEARED <RESULT>
  RESULT – результат операции
  0 – операция выполнена успешно

8.3.3 Команды для работы с солверами

8.3.3.1 КОМАНДА ЗАПУСКА СОЛВЕРА

Команда запускает солвер с указанными параметрами.

- Формат команды:
  SA_SOLVERRUN <количество процессоров**><**количество нитей**><CmdLineIndex>**
  Если количество процессоров указано равным 0, солвер будет запущен в однопроцессорном режиме без использования MPI
  Если указано количество нитей 0, их количество будет выбрано равным количеству ядер процессора

CmdLineIndex – индекс командной строки. Должен соответствовать номеру командной строки (начиная с 0) из списка, полученного в качестве ответа на команду SA_CONNECT для соответствующего типа запуска (одно- или многопроцессорного)

- Формат ответа:
  SA_SOLVERRUNNED <RESULT><FV_SOLVERID><IP-адрес солвера><N-порта для связи с демоном пре-пост процессора>
  RESULT – результат запуска
  0 – запуск прошел успешно
  1 – не удалось запустить солвер
  2 – нет командной строки с указанным индексом

8.3.3.2 КОМАНДА ПРОВЕРКИ СТАТУСА СОЛВЕРА

Команда позволяет проверить, существует ли солвер с указанным идентификатором на связи с солвер-агентом.

- Формат команды:
  SA_SOLVERCHECK <FV_SOLVERID>
- Формат ответа:
  SA_SOLVERCHECKED <RESULT>
  RESULT – результат проверки
  0 – солвер с данным ID существует на связи с солвер агентом
  1 – нет такого солвера на связи с солвер агентом

8.3.3.3 КОМАНДА ПОЛУЧЕНИЯ ПАРАМЕТРОВ СОЛВЕРА

Команда позволяет получить параметры запущенного ранее солвера с указанным идентификатором.

- Формат команды:
  SA_SOLVERPARAMGET <FV_SOLVERID>
  FV_SOLVERID – идентификатор запущенного солвера
- Формат ответа:
  SA_SOLVERPARAMGETED <RESULT><IP-адрес ><N-порта><ProcNum>
  RESULT – результат операции
  0 – солвер с данным ID существует, команда возвращает его параметры
  1 – не существует солвера с таким ID
  IP-адрес – IP адрес компьютера, на котором запущен солвер
  N-порта – номер порта для связи с солвером
  ProcNum – количество процессоров, на которых запущен солвер

8.3.3.4 КОМАНДА ПОЛУЧЕНИЯ СПИСКА СОЛВЕРОВ

Команда позволяет получить список солверов, запущенных указанным пользователем.

- Формат команды:
  SA_SOLVERSLISTGET <USERNAME>
- Формат ответа:
  SA_SOLVERSLISTGETTED <Count><SI 1><SI 2>….<SI n>
  Count – количество активных солверов для данного пользователя
  SI – параметры солвера, разделенные точкой с запятой
  - Solver ID
  - IP адрес для связи с солвером
  - Номер порта для связи с солвером

- Количество процессоров, на которых запущен солвер и количество нитей солвера (разделенные двоеточием)
- Состояние проекта (если проект загружен на данный солвер)
- Дополнительное состояние проекта (если проект загружен на данный солвер)
- Имя проекта, если он загружен/считается на солвере
- ID процесса солвера
- Версия солвера

8.3.3.5 КОМАНДА ЗАВЕРШЕНИЯ РАБОТЫ СОЛВЕРА

Завершает работу солвера с указанным ID.

- Формат команды:
  Формат команды:
  SA_KILLSOLVER <FV_SOLVERID>
  FV_SOLVERID – идентификатор запущенного солвера

- Формат ответа:
  SA_KILLSOLVERSTARTED <RESULT>
  RESULT – результат операции
  0 – команда успешно передана солверу
  1 – нет солвера с таким ID или отсутствует связь с ним
  2 – солвер с указанным ID является автономным (для завершения работы такого солвера необходимо использовать команду SA_FORCEKILLSOLVER)

8.3.3.6 КОМАНДА ЗАВЕРШЕНИЯ РАБОТЫ АВТОНОМНОГО СОЛВЕРА

Завершает работу автономного (запущенного командным файлом) солвера с указанным ID.

- Формат команды:
  SA_FORCEKILLSOLVER <FV_SOLVERID>
  FV_SOLVERID – идентификатор запущенного солвера

  Формат ответа:
  SA_FORCEKILLSOLVERSTARTED <RESULT>
  RESULT – результат операции
  0 – команда успешно передана солверу
  1 – нет солвера с таким ID или отсутствует связь с ним

8.3.4 Команды, посылаемые на клиент солвер-агентом

8.3.4.1 КОМАНДА ПОКАЗА ИНДИКАТОРА ПРОГРЕССА

Команда посылается солвер-агентом перед началом выполнения длительной операции, в ходе которой пользователю на экране должен отображаться индикатор прогресса. Примером такой операции является скачивание проекта на клиент. Клиентское приложение, получив эту команду от солвер-агента, должно вывести на экран индикатор прогресса.

- Формат команды:
  SA_PROGRESSBARSHOW <StartVal><EndVal>
  StartVal – начальное значение прогресс-бара

EndVal – конечное значение прогресс-бара

- Формат ответа:
  ответ не предусмотрен

8.3.4.2 КОМАНДА ОБНОВЛЕНИЯ ИНДИКАТОРА ПРОГРЕССА

Команда посылается солвер-агентом для обновления индикатора прогресса. В команде указывается текущее значение, которое нужно установить в индикаторе.

- Формат команды:
  SA_PROGRESSBARMOVE <NewVal>
  NewVal – новое значение прогресс-бара

- Формат ответа:
  ответ не предусмотрен

8.3.4.3 КОМАНДА ЗАКРЫТИЯ ИНДИКАТОРА ПРОГРЕССА

Команда посылается солвер-агентом для закрытия индикатора прогресса по окончании выполнения длительной операции.

- Формат команды:
  SA_PROGRESSBARHIDE

- Формат ответа:
  ответ не предусмотрен

8.3.4.4 ОТВЕТ НА НЕРАСПОЗНАННУЮ КОМАНДУ

Данный ответ посылается солвер-агентом, если он не смог распознать пришедшую ему команду (команда имеет неверный префикс или состав аргументов).

- Формат команды:
  SA_BADCOMMAND

- Формат ответа:
  Ответ не предусмотрен

9. ИНФОРМАЦИЯ О РАЗРАБОТЧИКЕ

API Вычислительная инженерная платформа (API ВИП) разработан Обществом с ограниченной ответственностью «Вычислительная инженерная платформа».

Контакты

+7(495) 612-8109

ООО "Вычислительная инженерная платформа"

Офис: Россия, Москва, ул. Юннатов, д. 18, офис 905

Почтовый адрес: 143026, Москва, территория Сколково инновационного центра, ул. Нобеля, д. 7

Email: info@cfd-platform.ru

Исследования осуществляются ООО "ВИП" при грантовой поддержке фонда "Сколково"

FlowVision

Вычислительная инженерная платформа представляет собой набор документации, программных библиотек и кодов, реализующих API для ПО FlowVision. FlowVision предназначен для моделирования гидро-аэродинамики и теплопередачи. По вопросам приобретения FlowVision обращайтесь по адресу info@flowvision.ru , ООО «ТЕСИС», Адрес: 127083, Россия, Москва, ул.Юннатов, дом 18, 7-й этаж, оф.705

Телефон: +7(495)612-4422;