Вычислительная Инженерная Платформа

Механизм подключения пользовательских модулей к Вычислительной Инженерной Платформе

РУКОВОДСТВО ПРОГРАММИСТА

СОДЕРЖАНИЕ

Содержание ... 1
1. Пользовательские модули ... 2
1.1. Общее описание ... 2
1.2. Структура пользовательского модуля ... 2
1.3. Схема работы пользовательского модуля ... 8
1.4. Общий интерфейс пользовательского модуля ... 9
1.5. Свойство типа файл ... 11
1.6. Сохранение состояния пользовательского модуля на диск ... 12
1.7. Интерфейс пользовательского модуля типа Evaluator ... 13
1.8. Интерфейс пользовательского модуля типа Binder ... 14
1.9. Отладка модуля ... 14
2. Интерфейс API Вычислительной Инженерной Платформы ... 16
2.1. Общее описание ... 16
2.2. Типы данных ... 16
2.3. Функции API ВИП ... 16
3. Лицензионная защита модуля ... 18
3.1. Варианты лицензионной защиты ... 18
3.2. Механизм лицензионной защиты, предоставляемый Вычислительной инженерной платформой ... 18
4. Пример модуля типа Evaluator ... 20
4.1. Постановка задачи ... 20
4.2. Пользовательский интерфейс ... 20
4.3. Программирование модуля ... 23
5. Пример модуля типа Binder ... 25
5.1. Постановка задачи ... 25
5.2. Пользовательский интерфейс ... 26
5.3. Программирование модуля ... 28
6. Информация о разработчике ... 30

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 32-битная Windows x86, из архива будут распакованы файлы Windows_x86/*.*, 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/x86
- 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>
      <x86>BinderTemplate_x86.dll</x86>
      <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 (рис. 1.3.1).

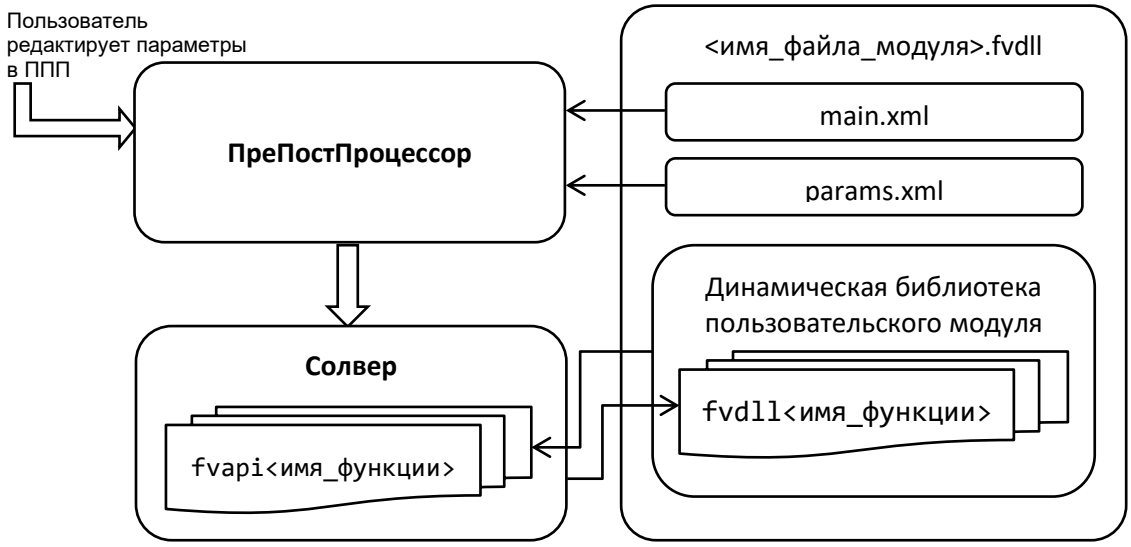

Рисунок 1.3.1 – Схема работы пользовательского модуля

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 места 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()
```

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 1.4.1.

Таблица 1.4.1 – Структура FVDLL_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. 1.5.

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

Кроме этого, существуют функции, общие для всех типов модулей, но реализуемые опционально. Такие функции описаны в п. 1.6.

1.5. СВОЙСТВО ТИПА ФАЙЛ

Свойство типа файл позволяет пользователю при редактировании проекта в параметрах модуля выбрать произвольный файл (рис. 1.5.1). 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.

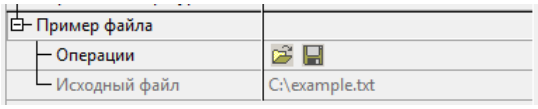

Рисунок 1.5.1 – Свойство типа файл

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 1.5.1.

Таблица 1.5.1– 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.

1.6. 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» необходимо включить эту опцию, добавив в **main.xml** следующий тэг:

```
<SaveParameters/>
```

В модуле, в этом случае, должны быть реализованы следующие функции:

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

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 1.4.1, 1.4.2 и 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 необходимо включить эту опцию, добавив в **main.xml** следующий тэг:

```
<SaveBinaryData/>
```

В модуле, в этом случае, 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();
```

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. 1.5) 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.

1.7. 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`.

1.8. 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 fvdllBeforeTimeStep(FVDLL_CONTEXT _hContext)
3. FVDLL_API fvdllAfterTimeStep(FVDLL_CONTEXT _hContext)
4. 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.

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`.

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`.

1.9.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).

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

4.1. ПОСТАНОВКА ЗАДАЧИ

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

Данный модуль возвращает значение синуса (4.1.1) или косинуса (4.1.2), в соответствии с выбранной в параметрах модуля функцией, зависящих от расстояния до точки, в которой запрашивается значение, от начала координат.

$$A\left|\sin\left(v\sqrt{(xk_x)^2+\left(yk_y\right)^2+(zk_z)^2}\right)\right| \tag{4.1.1}$$

$$A\left|\cos\left(v\sqrt{(xk_x)^2+\left(yk_y\right)^2+(zk_z)^2}\right)\right| \tag{4.1.2}$$

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).

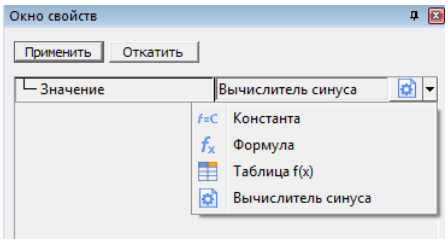

Рисунок 4.2.1 – Подключение Вычислителя синуса в качестве значения

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

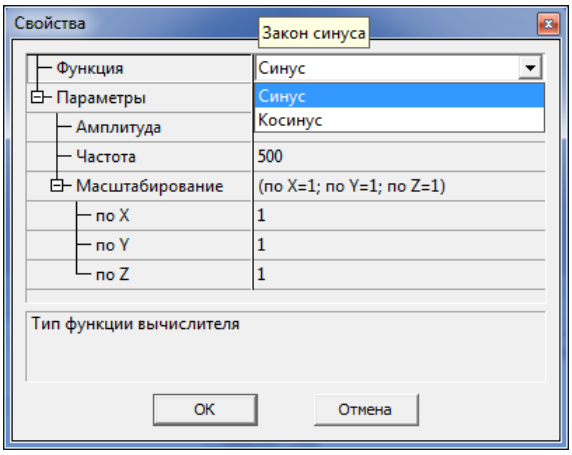

Рисунок 4.2.2 – Выбор функции, значение которой будет возвращено

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:

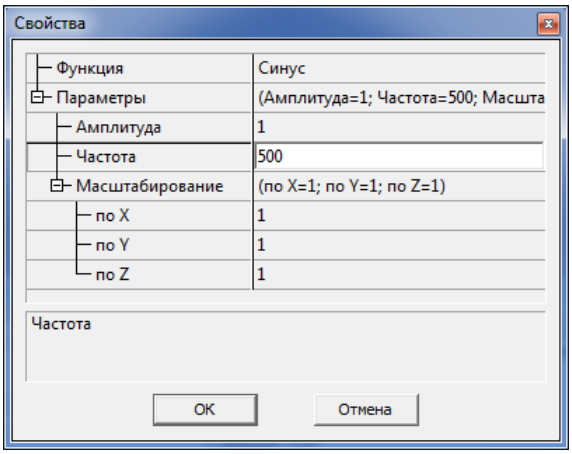

Рисунок 4.2.3 – Окно параметров Вычислителя синуса

```
<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. 1.7, 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 BINDER

5.1.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 (5.1.1).

$$\omega = \frac{\omega_{RPM}}{60} \tag{5.1.1}$$

Модуль запрашивает у Солвера площадь границы S, и по формуле (5.1.2) вычисляется диаметр активного диска.

$$D = \sqrt{\frac{4S}{\pi}} \tag{5.1.2}$$

Рассмотрим формулы для вычисления относительной скорости винта (5.1.3) и упора (5.1.4).

$$v_A = J\omega D \tag{5.1.3}$$

$$T = K_T \rho \omega^2 D^4, \tag{5.1.4}$$

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 (5.1.5).

$$v_A = \frac{(\rho V_n)_1}{\rho} \tag{5.1.5}$$

На первой границе установлено граничное условие «Давление на входе». Значение давления p_1 вычисляется по формуле (5.1.7).

$$T = (p_2 - p_1)S \tag{5.1.6}$$

$$p_1 = p_2 - \frac{T(v_A)}{S} \tag{5.1.7}$$

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 (5.1.8).

$$(\rho V_n)_2 = (\rho V_n)_1 \tag{5.1.8}$$

5.2.ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

В интерфейсе пользовательский модуль типа Binder позволяет создавать связку своего типа (рисунок 5.2.1).

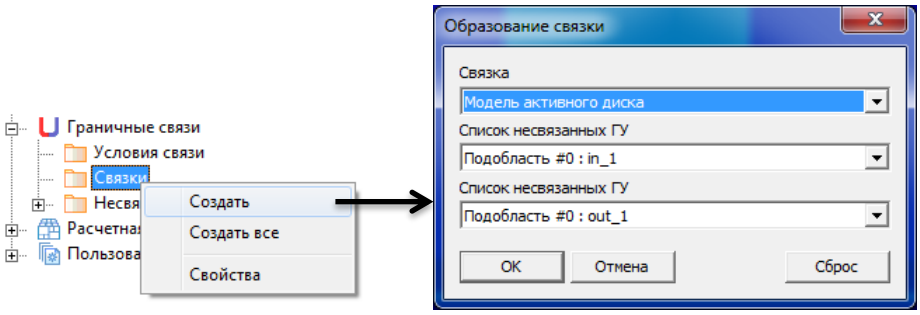

Рисунок 5.2.1 – Создание связки типа Модель активного диска

Выбор образованной связки в дереве препроцессора отображает окно свойств данной связки, в котором возможно редактирование параметров (рисунок 5.2.2).

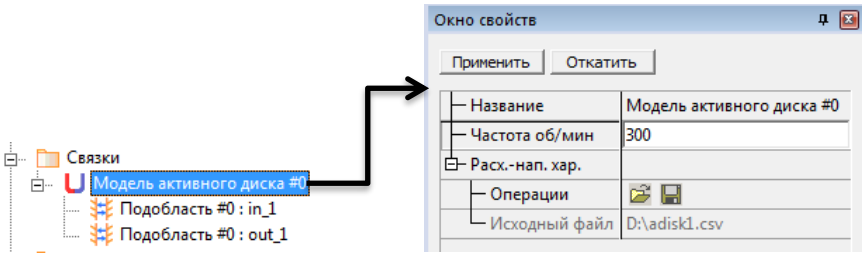

Рисунок 5.2.2 – Окно свойств связки

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 5.2.3):

- 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 5.2.3 – Расходно-напорная характеристика

Также возможно создание условия связи типа пользовательского модуля (рисунок 5.2.4).

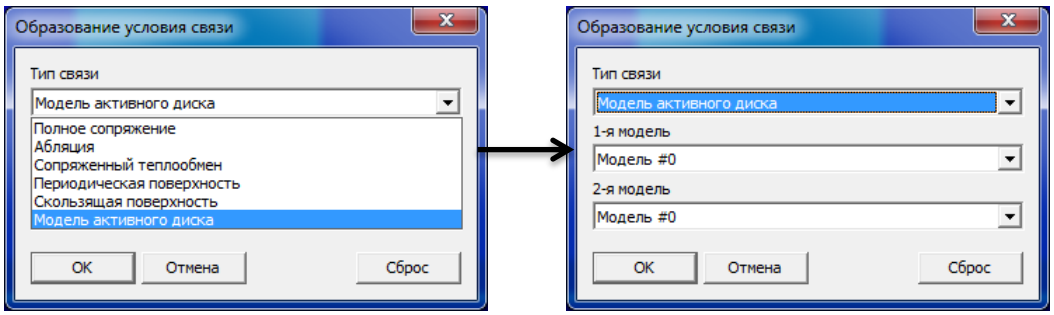

Рисунок 5.2.4 – Образование условия связи типа Модель активного диска

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 5.2.5).

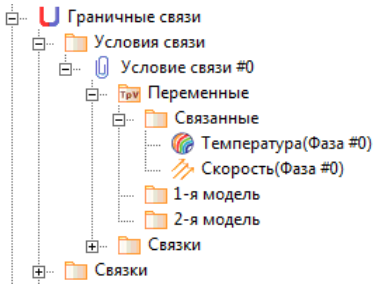

Рисунок 5.2.5 – Связанные переменные в интерфейсе

5.3.ПРОГРАММИРОВАНИЕ МОДУЛЯ

В модуле типа Binder, в соответствии с пунктами 1.4 и 1.8, должны быть реализованы 7 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. 1.8) 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. 5.1. 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.

6. 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 703

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

Частью Вычислительной инженерной платформы является ПО FlowVision 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

Телефоны и факсы: +7(495)612-4422; +7(495)612-4262; +7(495)612-8109