

PRADIS

НАПИСАНИЕ ПЛАГИН-ОБЪЕКТОВ НА ЯЗЫКЕ ФОРТРАН

**ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ АВТОМАТИЗАЦИИ
МОДЕЛИРОВАНИЯ НЕСТАЦИОНАРНЫХ ПРОЦЕССОВ В
МЕХАНИЧЕСКИХ СИСТЕМАХ И СИСТЕМАХ ИНОЙ
ФИЗИЧЕСКОЙ ПРИРОДЫ**

ВЕРСИЯ 4.3

Содержание

Содержание.....	2
1. Описание технологии.....	3
1.1 Цели.....	3
1.2 Загрузка репозитория.....	3
1.3 Инициализация plugin.....	3
1.4 Использование plugin при расчете.....	3
2. Репозиторий plugin.....	4
2.1 Формат репозитория.....	4
2.2 Дерево репозитория.....	4
2.3 Разбор узлов дерева “plugin”.....	5
2.4 Разбор узлов дерева “model”.....	5
2.5 Разбор узлов дерева “pgo”.....	7
2.6 Разбор узлов дерева “prvp”.....	7
2.7 Добавление новых свойств и типов компонент.....	8
3. Изменения в решателе при реализации технологии.....	9
3.1 Глобальные функции обработки репозитория.....	9
3.2 Подключение репозитория plugin в решатель.....	9
4 Интерфейсы plugin.....	10
4.1 Унифицированные вызовы plugin.....	10
4.2 Соглашения о вызовах и декорирование.....	10
4.3 Аргументы вызова инициализации.....	10
4.4 Аргументы вызова модели элемента.....	11
4.5 Аргументы вызова ПГО.....	11
4.6 Аргументы вызова ПРВП.....	12
5 Добавление plugin в сборку решателя.....	14

1. Описание технологии.

1.1 Цели.

Технология динамического встраивания в решатель PRADIS разработана для возможности добавлять новые модели элементов, ПГО, ПРВП без изменения существующего кода решателя, и без его перекомпиляции.

1.2 Загрузка репозитария.

Перед стартом расчета решатель загружает конфигурационный файл (**репозитарий**), в котором описан набор встраиваемых динамических библиотек (plugin), и компонент (моделей элементов, ПГО, ПРВП), которые в них реализованы. Далее для моделей, ПГО и ПРВП строятся таблицы соответствия числовых идентификаторов (из armcltg), и структур, описывающих компоненты. Наиболее важный член в каждой структуре – адрес глобальной процедуры, реализующей компоненту (модель элемента, ПГО или ПРВП). Перед построением таблиц соответствия идентификаторам все динамические библиотеки загружаются в адресное пространство процесса решателя. Далее, при чтении из репозитария идентификатора и имени функции компоненты, адрес соответствующей функции ищется в модуле загруженной библиотеки, и регистрируется в таблице нужного типа (модель/ПГО/ПРВП), со своим идентификатором.

1.3 Инициализация plugin.

Кроме процедур, реализующих компоненты, каждая динамическая библиотека plugin обязана содержать функцию инициализации. В нее перед расчетом решатель передаст адреса членов своих common областей (неименованной, NOTAT, GRCONF), которые могут понадобиться компонентам в ходе расчета. Функция инициализации должна сохранить переданные адреса в глобальных переменных, которые будут доступны функциям-компонентам.

1.4 Использование plugin при расчете.

В процессе решения перед обращением к модели элемента, ПГО, или ПРВП выполняется поиск в соответствующей таблице загруженных компонент по идентификатору. Если соответствующая идентификатору компонента найдена – вызывается она, если не найдена – делается попытка вызвать компоненту, обращение к которой жестко закодировано в решателе.

2. Репозиторий plugin.

2.1 Формат репозитария.

После инсталляции PRADIS репозиторий plugin должен располагаться в файле %DINSYS%\dinama\sysarm\plugin_repository.xml. Формат файла является упрощенным XML. В начале файла обязательно должен присутствовать тэг <?xml version="1.0"?>, затем один корневой тэг <root></root>. В нем может находиться произвольный набор поддеревьев XML тэгов. Из них анализируются только поддеревья с верхним тэгом “plugin” (см. ниже). Остальные поддеревья тэгов могут быть произвольными, формат не накладывает на них ограничений. Код обработки репозитария содержится в файле pradis\src\pradis32\pradis\solver\itgdl\plugin_repository.cpp.

2.2 Дерево репозитария.

Принцип обработки репозитария следующий. Весь xml файл считывается в дерево узлов, соответствующих тэгам. Дерево представлено шаблоном `ctm::cxx::Tree`, определенном в `pradis\include\ctmstd\cxx_tree.h`. В качестве узла дерева используется структура `ctm::pradis::plugin::Node`, определенная в `plugin_repository.cpp`. Узел хранит строковое имя, а также двусвязный список значений простых типов: `char`, `unsigned char`, `bool`, `short`, `unsigned short`, `long`, `unsigned long`, `int`, `unsigned int`, `float`, `double`, `std::string`. Если в XML встречается любой тэг за исключением одного из специальных (см. ниже), в текущий узел дерева добавляется дочерний узел с таким же именем, как встреченный тэг. Атрибуты и текст тэга игнорируются. Все дочерние XML тэги будут добавляться в дерево как дочерние узлы уже этого добавленного узла. Если встречается специальный тэг, то в текущий узел дерева добавляется не дочерний узел, а очередное значение (интерпретация текста тэга), в соответствии с таблицей:

<code><c></c></code>	<code>char</code> ,
<code><uc></uc></code>	<code>unsigned char</code> ,
<code></code>	<code>bool</code> ,
<code><h></h></code>	<code>short</code> ,
<code><uh></uh></code>	<code>unsigned short</code> ,
<code><l></l></code>	<code>long</code> ,
<code></code>	<code>unsigned long</code> ,
<code><i></i></code>	<code>int</code> ,
<code><ui></ui></code>	<code>unsigned int</code> ,
<code><f></f></code>	<code>float</code> ,
<code><d></d></code>	<code>double</code> ,
<code><s></s></code>	<code>std::string</code> .

Добавлять значение в узел можно шаблонным методом `ctm::pradis::plugin::Node::push_value()`. После того, как узел сформирован можно перебирать его значения с помощью итераторов списка, возвращаемых `ctm::pradis::plugin::Node::begin()` и `ctm::pradis::plugin::Node::end()`. Определить тип значения по итератору можно с помощью `ctm::pradis::plugin::Node::get_tag()`, а получить значение по итератору - с помощью шаблонного метода `ctm::pradis::plugin::Node::get_value()`.

Таким образом, на основе репозитария строится дерево узлов с именами тэгов, и массивами значений простых типов, полученных из специальных подтэгов. Построение дерева осуществляет метод `ctm::pradis::plugin::Repository::LoadTree()` из `plugin_repository.cpp`. Класс `ctm::pradis::plugin::Repository` представляет в решателе репозиторий plugin, а его публичный метод `Load()` выполняет всю загрузку репозитария (вместе с построением таблиц компонент), из входного потока к XML.

2.3 Разбор узлов дерева “plugin”.

После построения дерева, метод `ctm::pradis::plugin::Repository::Load()` перебирает узлы дерева первого уровня после `root`. Из них анализируются узлы с именем “plugin”, каждый из которых представляет одну динамически загружаемую библиотеку. Остальные узлы первого уровня игнорируются. В каждом узле “plugin” должен находиться один узел “library”, первым значением в котором должна быть строка (тэг `<s></s>`) – имя динамической библиотеки `plugin`, которая должна находиться в `%DINSYS%\dinama\pradis32`. Кроме этого, в каждом узле “plugin”, может присутствовать один подузел “init” и один подузел “cleanup”. Первыми значениями этих узлов должны быть строки – имена функций инициализации динамической библиотеки (в узле “init”) перед расчетом, и ее очистки (в узле “cleanup”), после окончания расчета. Если эти узлы присутствуют, функции с соответствующими именами обязательно должны экспортироваться динамической библиотекой `plugin`. Функция инициализации обязательно должна присутствовать в библиотеке `plugin`. Если узла “init” нет – считается, что функция инициализации называется “INIT”. Об аргументах функции инициализации см. ниже. Функции о чистки может не быть в библиотеке. Если узла “cleanup” нет – считается, что нет и функции, и она не вызывается. Функция очистки не имеет аргументов. Кроме узлов “library”, “init” и “cleanup”, в узле “plugin” может содержаться произвольное количество узлов “model”, “pgo” или “prvp”, (описывают компоненты, реализованные в `plugin` библиотеке), а также любые другие узлы, которые не анализируются при разборе дерева, построенного из XML репозитория.

Пример XML для описания `plugin`:

```
<plugin>
  <library><s>balka</s></library>
  <init><s>INIT</s></init>
  <cleanup><s>CLEAN</s></cleanup>
  <model>
    .....
  </model>
  .....
  <pgo>
    .....
  </pgo>
  .....
  <prvp>
</plugin>
```

Код разбора узлов дерева “plugin” содержится в методе `ctm::pradis::plugin::Repository::ProcessPlugin()`. В этом методе выполняется обход подузлов узла “plugin” и вызов обработчиков для подузлов типа “model”, “pgo”, “prvp”. При необходимости добавить еще один тип `plugin` компоненты, этот метод несложно расширить.

2.4 Разбор узлов дерева “model”.

Код разбора узлов дерева “plugin”/”model” содержится в методе `ctm::pradis::plugin::Repository::ProcessModel()`. В нем анализируются подузлы следующих типов:

- “id”, обязательно должен присутствовать, и содержать в качестве первого значения `unsigned int` (XML тэг `<ui></ui>`) – идентификатор модели, соответствующий `armctlg`.

- “procedure”, обязательно должен присутствовать, и содержать в качестве первого значения std::string (XML тэг <s></s>) – имя процедуры модели элемента. О параметрах процедуры модели элемента см. ниже.
- “parameters”, обязательно должен присутствовать. Может содержать по одному узлу типа “ext”, “ent”, “adr”, “ign”. Узлы других типов могут содержаться в “parameters”, но не анализируются. Если присутствует, каждый из узлов “ext”, “ent”, “adr” или “ign” должен содержать одно значение int (XML тэг <i></i>). Эти значения имеют тот же смысл, что и аналогичные параметры паспорта модели элемента, и должны совпадать со значениями паспорта данной модели в armctlg. Если какое-либо из значений отсутствует, принимается, что оно равно значению по умолчанию, по тем же правилам, по которым выставляются значения по умолчанию в паспорте модели элемента при добавлении в armctlg.
- “aliases”, обязательно должен присутствовать и содержать 0 или более значений std::string (XML тэг <s></s>) – альтернативных имен для процедуры модели элемента, для будущего использования в трансляторе.
- “classes”, если присутствует, может содержать 0 или более подузлов типа “system” (физическая система). Узлы других типов могут содержаться в “classes”, но не анализируются. Каждый из узлов “system” может содержать по одному узлу “name” (имя физической системы) и “defaultPGO” (ПГО по умолчанию для данной физической системы). Узлы других типов могут содержаться в узлах “system”, но не анализируются. Из узлов “name” и “defaultPGO” извлекаются и запоминаются первые строковые значения, для будущего использования в трансляторе.
- “nodes”, если присутствует, может содержать 0 или более подузлов “node”. Узлы других типов могут содержаться в “nodes”, но не анализируются. В каждом узле типа “node” анализируется один узел типа “system” (имя физической системы для узла с номером, соответствующим порядковому номеру узла “node” в узле “nodes”). Если узел “system” присутствует, из него извлекается первое строковое значение, и запоминается, для будущего использования в трансляторе.

Пример XML для описания model:

```
<model>
  <id><ui>75</ui></id>
  <procedure><s>MODEL</s></procedure>
  <parameters>
    <ext><i>6</i></ext>
    <ent><i>0</i></ent>
    <adr><i>1</i></adr>
    <ign><i>2</i></ign>
  </parameters>
  <aliases>
    <s>BALKKA</s>
    <s>balka</s>
  </aliases>
  <classes>
    <system>
      <name><s>mechanics</s></name>
      <defaultPGO><s>PGO1</s></defaultPGO>
    </system>
    <system>
      <name><s>hydraulics</s></name>
      <defaultPGO><s>PGO2</s></defaultPGO>
    </system>
  </classes>
```

```

<nodes>
  <node>
    <system><s>mechanics</s></system>
  </node>
  <node>
    <system><s>hydraulics</s></system>
  </node>
</nodes>
</model>

```

Любые другие узлы могут содержаться в дереве, но не анализируются. При необходимости добавить еще одну характеристику модели элемента, метод `ctm::pradis::plugin::Repository::ProcessModel()` несложно расширить.

2.5 Разбор узлов дерева “pgo”.

Код разбора узлов дерева “plugin”/”pgo” содержится в методе `ctm::pradis::plugin::Repository::ProcessPgo()`. В нем анализируются подузлы следующих типов:

- “id”, обязательно должен присутствовать, и содержать в качестве первого значения `unsigned int` (XML тэг `<ui></ui>`) – идентификатор ПГО, соответствующий `armctlg`.
- “procedure”, обязательно должен присутствовать, и содержать в качестве первого значения `std::string` (XML тэг `<s></s>`) – имя процедуры ПГО. О параметрах процедуры ПГО см. ниже.
- “aliases”, обязательно должен присутствовать и содержать 0 или более значений `std::string` (XML тэг `<s></s>`) – альтернативных имен для процедуры ПГО, для будущего использования в трансляторе.

Пример XML для описания pgo:

```

<pgo>
  <id><ui>5</ui></id>
  <procedure><s>AKLAB</s></procedure>
  <aliases>
    <s>Pgo1</s>
    <s>pgo01</s>
  </aliases>
</pgo>

```

Любые другие узлы могут содержаться в дереве, но не анализируются. При необходимости добавить еще одну характеристику ПГО, метод `ctm::pradis::plugin::Repository::ProcessPgo()` несложно расширить.

2.6 Разбор узлов дерева “prvp”.

Код разбора узлов дерева “plugin”/”prvp” содержится в методе `ctm::pradis::plugin::Repository::ProcessPRVP()`. В нем анализируются подузлы следующих типов:

- “id”, обязательно должен присутствовать, и содержать в качестве первого значения `unsigned int` (XML тэг `<ui></ui>`) – идентификатор ПРВП, соответствующий `armctlg`.
- “procedure”, обязательно должен присутствовать, и содержать в качестве первого значения `std::string` (XML тэг `<s></s>`) – имя процедуры ПРВП. О параметрах процедуры ПРВП см. ниже.

- “aliases”, обязательно должен присутствовать и содержать 0 или более значений std::string (XML тэг <s></s>) – альтернативных имен для процедуры ПРВП, для будущего использования в трансляторе.

Пример XML для описания prvp:

```
<prvp>
  <id><ui>59</ui></id>
  <procedure><s>X</s></procedure>
  <aliases>
    <s>PRVP1</s>
    <s>prvp01</s>
  </aliases>
</prvp>
```

Любые другие узлы могут содержаться в дереве, но не анализируются. При необходимости добавить еще одну характеристику ПРВП, метод ctm::pradis::plugin::Repository::ProcessPRVP() несложно расширить.

2.7 Добавление новых свойств и типов компонент.

Преимуществом описанного в предыдущих пунктах подхода с построением по XML файлу дерева узлов, и их последующим анализом, является сравнительная легкость добавления новых элементов в конфигурацию plugin. В самом деле, если необходимо добавить новый тип компонент, или новое свойство к уже существующему типу компонент, всегда можно добавить в соответствующее место XML файла новое поддерево именованных тэгов, содержащих значения простых типов (строк, чисел и т.п.). После этого, файл по-прежнему будет корректно считываться в дерево, просто узлы новых типов еще не будут анализироваться. Затем, следует добавить код анализирующий узлы новых типов в ctm::pradis::plugin::Repository::ProcessPlugin(), ctm::pradis::plugin::Repository::ProcessModel(), ctm::pradis::plugin::Repository::ProcessPGO() или ctm::pradis::plugin::Repository::ProcessPRVP(). Здесь значения из узлов нужно будет сохранить, определив для них соответствующие структуры, или расширив существующие.

3. Изменения в решателе при реализации технологии.

3.1 Глобальные функции обработки репозитория.

Как говорилось выше, класс, представляющий в решателе репозиторий plugin (ctm::pradis::plugin::Repository), определен в новом исходном файле модуля itgdlл pradis\src\pradis32\pradis\solver\itgdlл\plugin_repository.cpp. В этом же файле определены глобальные функции для взаимодействия с репозитарием:

ITGDLL_INIT_REPOSITORY() – инициализировать (загрузить) репозитарий,

ITGDLL_CLEAN_REPOSITORY() – очистить репозитарий,

ITGDLL_EXISTS_PLUGIN() – проверить, существует ли в репозитарии компонента с заданным типом и идентификатором,

ITGDLL_GET_MODEL_PARAM() – вернуть параметр модели элемента с заданным идентификатором,

ITGDLL_INVOKE_MODEL() – вызвать модель элемента с заданным идентификатором,

ITGDLL_INVOKE_PGO() – вызвать ПГО с заданным идентификатором,

ITGDLL_INVOKE_PRVP() – вызвать ПРВП с заданным идентификатором.

Все функции декорированы в стиле C. При компиляции под Windows в них используется соглашение о вызовах stdcall. Благодаря этому, функции могут быть вызваны из FORTRAN кода, скомпилированного с установками компилятора DIGITAL по умолчанию.

3.2 Подключение репозитория plugin в решатель.

Загрузка и очистка репозитария добавлены до и после расчета в исходном файле решателя pradis\src\pradis32\pradis\solver\run\integr.for. Обращение к моделям элементов, ПГО и ПРВП выполняется в решателе в исходном файле pradis\src\pradis32\pradis\solver\itg\integr.for (процедуры FORMY, FORMO, FORMI). Для подключения plugin компонент в этом файле, во все три функции добавлен оператор IF, проверяющий наличие в репозитарии компоненты с идентификатором, поступившим от решателя (вызов ITGDLL_EXISTS_PLUGIN()). Если компонента найдена, выполняется формирование списка параметров и вызов компоненты (с помощью ITGDLL_INVOKE_MODEL(), ITGDLL_INVOKE_PGO(), или ITGDLL_INVOKE_PRVP()). Если plugin компонента не найдена, как и раньше вызывается оператор-переключатель GOTO, выполняющий переход по идентификатору к вызову компоненты, жестко записанному в код integrs.for. Поскольку файл integrs.for автоматически генерируется с помощью модуля bridge.exe, изменения в коде integrs.for внесены в код его генерации в файлах gformi.for, gformo.for, gformy.for, расположенных в pradis\src\pradis32\pradis\solver\bridge\.

4 Интерфейсы plugin.

4.1 Унифицированные вызовы plugin.

Для того, чтобы обеспечить возможность динамического встраивания компонент в решатель, без его перекомпиляции было необходимо разработать для всех plugin процедуры единые, унифицированные интерфейсы. Под интерфейсом процедуры понимается соглашение об ее вызове, а также набор ее аргументов. Эти соглашения жестко записываются в код решателя перед его компиляцией. Имена функций динамически ищутся в модуле plugin библиотеки после ее загрузки. Решатель использует 5 видов вызовов библиотек, декларированных в файле `pradis\src\pradis32\pradis\solver\itgddl\plugin_repository.cpp`:

- `ctm::pradis::plugin::Repository::Library::FN_INIT` – инициализация библиотеки
- `ctm::pradis::plugin::Repository::Library::FN_CLEAN` – очистка библиотеки
- `ctm::pradis::plugin::Repository::FN_MODEL` – вызов модели элемента
- `ctm::pradis::plugin::Repository::FN_PGO` – вызов ПГО
- `ctm::pradis::plugin::Repository::FN_PRVP` – вызов ПРВП

Все процедуры не возвращают значение (тип возврата `void`). Функция `ctm::pradis::plugin::Repository::Library::FN_CLEAN` не имеет аргументов, описание аргументов остальных функций см. в подпунктах ниже.

4.2 Соглашения о вызовах и декорирование.

При компиляции под Windows все функции, экспортируемые из plugin, должны быть декорированы в стиле C и соответствовать соглашению о вызовах `stdcall`, что позволяет вызывать и реализовывать их в FORTRAN коде, скомпилированном с помощью DIGITAL FORTRAN с установками по умолчанию. В C коде аргументы этих функций описываются и передаются как указатели, при вызове или реализации в FORTRAN коде, аргументы функций обычным образом описываются как `REAL * 8`, `INTEGER * 4` и т.п. Следует обратить внимание, что при реализации plugin библиотеки под Windows на C/C++ экспортируемые функции в стиле C, `stdcall`, будут иметь декорацию имен `_function_name@N`, где N – количество байт на стеке, используемое под передачу аргументов. При регистрации в XML файле репозитория следует или указывать такие же имена (а не просто `function_name`), или, например, пользоваться `def` файлами для более удобного декорирования. Так же следует обратить внимание на то, что при реализации plugin библиотеки на DIGITAL FORTRAN под Windows, в соответствии с рекомендациями в следующем пункте имена всех экспортируемых функций будут переведены в верхний регистр. Во избежание ошибок рекомендуется после сборки plugin библиотеки просмотреть декорированные имена процедур с помощью утилиты `dumpbin` под Windows, из дистрибутива MSVC (`dumpbin /exports name.dll`). Unix аналог – утилита `nm`. В XML файле репозитория имена процедур для plugin следует прописывать в соответствии с выводом этих утилит.

4.3 Аргументы вызова инициализации.

Вызов инициализации передает в plugin библиотеку адреса членов COMMON областей решателя: непоименованной, `/NOTAT/`, `/GRCONF/`. Таким образом, список аргументов функции инициализации следующий:
(непоименованная)

TIME, STEP, STEP01, STEP02, SMIN, DABSI, DRLTI, STEPMD, TIMEND, NAME, NSTEP, SYSPRN, NITER, ITR, CODE, NUMINT, NUMPP, CODSTP, CODGRF, NEWINT, MINSTP,
 (/NOTAT/)
 RLMAX, RLMIN, INTMAX, MSHEPS, PI, REZB, REZC, REZD,
 (/GRCONF/)
 RELYX, XNMPXL, YNMPXL, XNMSMB, YNMSMB, NCOLOR, NMVPAG, MODES, IK4, IS4.

Точный смысл и FORTRAN типы аргументов можно посмотреть в описании членов COMMON областей из документации по разработке компонент в pradis, расположенной в pradis\res\lsv_pradis\pradis\docs\include\.

Если plugin реализуется на C/C++ FORTRAN типы аргументов вызова инициализации отображаются на типы C следующим образом:

REAL * 8: double*

INTEGER * 4: int*

INTEGER * 2: short*

CHARACTER * N: char*, int* (два параметра! Идут последовательно, второй имеет смысл длины строки и равен N).

Внимание! При реализации функции инициализации следует помнить, что сохранять в глобальных переменных plugin необходимо адреса, а не значения переданных членов COMMON областей. При реализации plugin на C/C++ это означает, что запомнить следует переданные указатели. При реализации plugin на FORTRAN рекомендуется создать аналогичные описанным выше COMMON области, содержащие указатели (POINTER) соответствующих типов. Внутри вызова инициализации следует привязать эти указатели к переданным в аргументах вызова значениям.

4.4 Аргументы вызова модели элемента.

Перечислим аргументы вызова модели элемента:

- I: вектор сил (моментов) для элемента.
- Y: якобиан модели элемента.
- X: вектор перемещений узлов размерности EXT+ENT. Не используется при ADR=2, ADR=3.
- V: вектор скоростей узлов размерности EXT+ENT. Не используется при ADR=3.
- A: вектор ускорений узлов размерности EXT+ENT.
- PAR: массив параметров модели.
- NEW: вектор "нового состояния" модели.
- OLD: вектор "старого состояния" модели.
- WRK: рабочий массив для модели элемента.

Все аргументы имеют тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++. Подробнее о значении аргументов (за исключением X, V, A), и параметрах паспорта модели элемента (EXT, ENT, ADR) см. документацию по разработке компонент в pradis, расположенную в pradis\res\lsv_pradis\pradis\docs\include\.

4.5 Аргументы вызова ПГО.

Перечислим аргументы вызова ПГО:

- NAMEX: имя модели или ПРВП, связанной с ПГО. Массив пробелов, при значениях параметров паспорта VPS=0 и EXT=0 (неподвижный графический образ). Имеет тип CHARACTER * 8 при реализации вызова на FORTRAN, и char*, int* (два аргумента!) при реализации на C/C++.

- I: вектор сил (моментов) для элемента. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- X: вектор перемещений узлов модели, связанной с ПГО, размерности EXT. Не используется при значении параметров паспорта VPS=0 и EXT=0, а также при значении параметра UNV>0 (в этом случае используется вектор INNER). Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- V: вектор скоростей узлов модели, связанной с ПГО, размерности EXT. Не используется при значении параметров паспорта VPS=0 и EXT=0, а также при значении параметра UNV>0 (в этом случае используется вектор INNER). Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- A: вектор ускорений узлов модели, связанной с ПГО, размерности EXT. Не используется при значении параметров паспорта VPS=0 и EXT=0, а также при значении параметра UNV>0 (в этом случае используется вектор INNER). Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- INNER: вектор вещественных чисел имеющих смысл степеней свободы модели элемента, связанной с ПГО. Не используется при значении параметров паспорта VPS=0 и EXT=0, а также при значении параметра UNV=0 (в этом случае используются вектора X,V,A). Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- EXT: количество степеней свободы модели элемента, связанной с ПГО (длина INNER). Имеет тип INTEGER * 4 при реализации вызова на FORTRAN, и int*, при реализации на C/C++.
- PARX: вектор параметров модели, связанной с ПГО. Не используется при значении параметров паспорта VPS=0 и EXT=0. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- WRKX: рабочий вектор модели, связанной с ПГО. Не используется при значении параметров паспорта VPS=0 и EXT=0. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- PAR: вектор параметров ПГО. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- WRK: рабочий вектор ПГО. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- PARLR2: вектор параметров текущего слоя изображения. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.

Подробнее о значении аргументов (за исключением X, V, A), и параметрах паспорта ПГО см. документацию по добавлению компонент в pradis, расположенную в pradis\res\lsv_pradis\pradis\docs\include\.

4.6 Аргументы вызова ПРВП.

- XOUT: рассчитываемая выходная переменная или вектор рассчитываемых выходных переменных. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- PAR: массив параметров ПРВП. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- NEW: вектор "нового состояния" ПРВП. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.

- OLD: вектор "старого состояния" ПРВП. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- WRK: рабочий массив для ПРВП. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- A: общий вектор вещественных переменных решателя. Имеет тип REAL * 8 при реализации вызова на FORTRAN, и double*, при реализации на C/C++.
- DOFADDR: массив адресов в векторе A, по которым располагаются необходимые ПРВП значения степеней свободы - перемещения, скорости, или ускорения. Если номер узла для ПРВП в PRADISlang задан как номер_узла – перемещения, если он задан как номер_узла' -скорости, и если он задан как номер_узла''-ускорения. Имеет тип INTEGER * 4 при реализации вызова на FORTRAN, и int*, при реализации на C/C++.
- NDOF: размер вектора DOFADDR. Имеет тип INTEGER * 4 при реализации вызова на FORTRAN, и int*, при реализации на C/C++.

Подробнее о значении аргументов (за исключением A, DOFADDR, NDOF), и параметрах паспорта ПРВП см. документацию по добавлению компонент в pradis, расположенную в pradis\res\lsv_pradis\pradis\docs\include\.

5 Добавление plugin в сборку решателя.

Перечислим шаги, которые необходимо проделать при добавлении plugin компоненты в решатель:

- Создать сборку динамической библиотеки plugin. Под Windows в среде MSVC6 + DIGITAL Fortran рекомендуется пользоваться визардом MS Visual Studio. Если разработка plugin библиотеки ведется в сборке консольного решателя, рекомендуется располагать проект plugin библиотеки в подкаталоге `pradis\src\pradis32\pradis\solver\plugin\`. При этом рекомендуется устанавливать в настройках проекта пути к директории временных файлов, выходным директориям сборки, как это принято в сборке решателя (см. настройки проекта тестовой plugin библиотеки `pradis\src\pradis32\pradis\solver\plugin\balka\balka.dsp`). В настройках runtime FORTRAN (или C) рекомендуется устанавливать использование runtime, как многопоточной динамической библиотеки.
- Добавить в сборку динамической библиотеки plugin исходный файл, с определением и экспортом вызова инициализации библиотеки перед расчетом. Подробнее о соглашениях вызовов plugin библиотек, передаче их аргументов и формате вызова инициализации см. предыдущий пункт. Важно помнить, что определение функции инициализации должно встречаться в кодах plugin библиотеки ровно один раз, в отличие от вызовов компонент (моделей элементов, ПГО, ПРВП), которых может быть произвольное количество. Если plugin библиотека реализуется под Windows на DIGITAL FORTRAN, рекомендуется пользоваться файлом `pradis\src\pradis32\pradis\solver\plugin\init.inc`, в котором определена и экспортирована процедура инициализации (см. включение файла в исходном файле `pradis\src\pradis32\pradis\solver\plugin\balka\balka.for`). В файле `init.inc`, передаваемые решателем члены COMMON областей привязываются к FORTRAN указателям, которые помещены в аналогичные COMMON области. Если plugin библиотека реализуется на C/C++, рекомендуется пользоваться файлом `pradis\src\pradis32\pradis\solver\plugin\init.h`, в котором определена и экспортирована процедура инициализации (см. включение файла в исходном файле `pradis\src\pradis32\pradis\solver\plugin\md\main.cpp`). В файле `init.h`, передаваемые решателем указатели на члены COMMON областей сохраняются в глобальной переменной – агрегате из этих указателей.
- Добавить в сборку динамической библиотеки plugin исходные файл, с определением и экспортом вызовов одной, или нескольких plugin компонент (моделей элементов, ПГО, ПРВП). Подробнее о соглашениях вызовов plugin библиотек, передаче их аргументов и форматах вызовов компонент см. предыдущий пункт. Если plugin библиотека реализуется под Windows на DIGITAL FORTRAN, рекомендуется пользоваться файлом `pradis\src\pradis32\pradis\solver\plugin\common.inc`, в котором определены COMMON области из указателей FORTRAN, соответствующие COMMON областям решателя. Значения членов этих COMMON областей могут требоваться plugin компонентам при расчете. См. включение файла в исходных файлах реализующих тестовые plugin компоненты `balka.for` (модель), `aklab.for` (ПГО), `X.for` (ПРВП). В файлах можно заимствовать способ экспорта вызовов компонент с помощью специального комментария (например, `!DEC$ ATTRIBUTES DLLEXPORT::AKLAB`). Если plugin библиотека реализуется под Windows на C/C++, рекомендуется пользоваться файлом `pradis\src\pradis32\pradis\solver\plugin\common.h`, в котором определена структура-агрегат из указателей на переменные COMMON областей решателя. Значения членов COMMON областей могут требоваться plugin компонентам при расчете. Для доступа к указателям на члены COMMON областей следует пользоваться макросом `PLUGIN_COMMON`, как именем переменной-

структуры, членами которой являются указатели. См. пример включения файла `common.h`, и применения макроса `PLUGIN_COMMON` в исходном файле `pradis\src\pradis32\pradis\solver\plugin\md\main.cpp`, реализующем тестовую `plugin` компоненту (модель элемента MD). В проекте `pradis\src\pradis32\pradis\solver\plugin\md` также можно заимствовать способ экспортирования и декорирования вызова компоненты в динамической библиотеке с помощью `.DEF` файла.

- Зарегистрировать компоненту в бинарном каталоге PRADIS `armctlg`. Для этого следует воспользоваться утилитой `arm` с параметром `!`. Её следует вызвать в директории, где расположен файл с расширением `.FOR`, начинающийся со специального комментария компоненты. Если `plugin` компонента реализуется на C++, для регистрации с помощью `arm` следует создать файл с расширением `.FOR`, содержащий только этот специальный комментарий. После регистрации с помощью `arm`, следует определить `armctlg` идентификатор компоненты, для последующего внесения в репозиторий `plugin` (см. следующий подпункт). Для этого предлагается открыть модифицированный файл `armctlg` редактором бинарных файлов MS Visual Studio. В файле следует найти первое вхождение имени добавленной компоненты и запомнить номер строки вхождения (слева в 16-ричной форме). Это число следует перевести в десятичную форму, поделить на 80, и прибавить 1, для чего, например, можно воспользоваться стандартным калькулятором Windows. Так будет получен `armctlg` идентификатор компоненты.
- Зарегистрировать в репозитории `plugin` библиотеку, вызов ее инициализации и вызовы реализованных в ней `plugin` компонент. См. пункт, описывающий репозиторий `plugin`, а также пример регистрации тестовых `plugin` библиотек. О получении идентификатора компоненты для регистрации см. в предыдущем подпункте.

Если в `plugin` компонентах требуется вызов библиотечных функций решателя (например, `S00X`), их нельзя вызывать из статических библиотек, напрямую связываемых с `plugin` библиотекой. Это потому, что тогда библиотечные функции могут пытаться пользоваться COMMON областями решателя напрямую, а они в модуле динамической `plugin` библиотеки отсутствуют. Для того, чтобы пользоваться библиотечными функциями в реализациях `plugin` компонент, необходимо вынести их “обертки” в модуль `itgdll` и связать `plugin` библиотеку при сборке с `itgdll.lib`. Тогда при вызове функции `plugin` библиотека будет обращаться к модулю `itgdll`, а там уже функции будут корректно вызываться. См. пример вызова из `pradis\src\pradis32\pradis\solver\plugin\balka\aklab.for` функций `DRAWAB` и `W_GLASS`, которые являются обертками реальных библиотечных функций и определены в `pradis\src\pradis32\pradis\solver\itgdll`. В настоящий момент в `itgdll` включен ограниченный набор библиотечных функций, которые потребовались при реализации тестовых `plugin` библиотек `balka` и `md`. Предполагается, что в дальнейшем все необходимые функции будут обернуты аналогами, определенными в `pradis\src\pradis32\pradis\solver\itgdll\libdllexp.f90`.