Написание Сценария

Написание Сценария Script

При разработке компонентных Определений в PSCAD широко используется язык написания сценариев. Он играет роль коммуникационного интерфейса между разработчиком и программой, предоставляя рекомендации об интеграции компонента в большие проекты. Процесс коммуникации происходит при компиляции, когда каждый сегмент Определения обрабатывается последовательно, а из сценария выделяется значимая для компилятора информация.

Некоторые сегменты допускают добавление кода источника (то есть участки кода, которые не требуют анализа), такие как компилятор Фортран, DSDYN and DSOUT, при этом используемый сценарий Определения обеспечивает актуальность кода, а также независимость от используемого языка и компилятора. В процессе компиляции сценарию Определения предоставляется наибольший приоритет; он обрабатывается в самом начале, а результаты используются в выражениях, заменах и при выполнении функций при дальнейшей компиляции кода.

Эта глава тесто связана с главой Component Design (Проектирование компонента). Перед тем, как приступать к самостоятельному проектированию компонента, рекомендуется усвоить материал обеих глав. В следующей главе описаны доступные инструменты для разработки сценария, а также приведены примеры по возможному их использованию. В разделе Component Design (Проектирование компонента) приведено руководство по созданию пользовательского компонента.

Замены Substitutions

$ Префикс оператора замены значения

% Префикс оператора замены данных

{ } Кавычки

! Оператор ввода комментариев

Замены – это основные операторы сценария, которые могут использоваться в любой части Определения. Они также представляют средство коммуникации между разделами Определения и сегментами, обеспечивают динамику графических компонентов, а также используются при форматировании кода и комментариев.

Доступные операторы замены приведены ниже:

- Префикс замены значения: **$**
    - Контекстная замена текста
    - $#DIM: Замена размерности порта
    - $#Component: Замена идентификатора образа компонента
- Префикс значения данных: **%**
- Операторы с кавычками: **{ }**
- Оператор ввода комментариев: **!**

$ Префикс оператора замены значения

Префикс оператора замены значения (или '$') - это наиболее важный из операторов замены, который используется при проектировании компонентных Определений. Знак $ обычно добавляется к переменной, где переменная может представлять число (или текст), которое было предварительно определено в процессе расчета. В не зависимости от того, что представляет переменная, в случае если ее предваряет префикс $, она будет заменена.

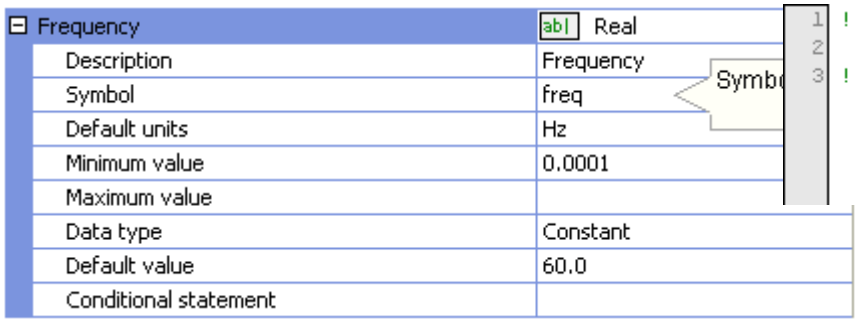
Оператор $ может предварять только переменные определенные одним из следующих способов:

- Имя порта *Symbol*
- Входной параметр, текстовое поле или список с именем *Symbol*
- Любая переменная, определенная в сегменте Computations (Вычисления)
- Имя ключа

Существует множество способов использования оператора $. Одним из способов является использование замены буквенного значения вместо имени переменной: Буквенное значение переменной – для порта, входного параметра или переменной из раздела Computations (Вычисления) – будет заменено при компиляции. Буквенные замены могут также использоваться для создания комментариев.

ПРИМЕР 10-1:

Буквенное значение, введенное в поле входного параметра, может быть заменено напрямую в коде источника при добавления префикса $ к имени поля Symbol.



Свойства диалогового окна входного поля

Замена значений при использовании компилятора Фортран

```
!
      REAL     RT_1             ! EMTDC variable to represent 'freq'
      REAL     OUT


      RT_1 = STOF(ISTOF + 2)    ! EMTDC extracts parameter value
                                ! from stored data array
                                ! (модуль EMTDC
                                ! извлекает параметры из массива хранения
                                ! данных)
```

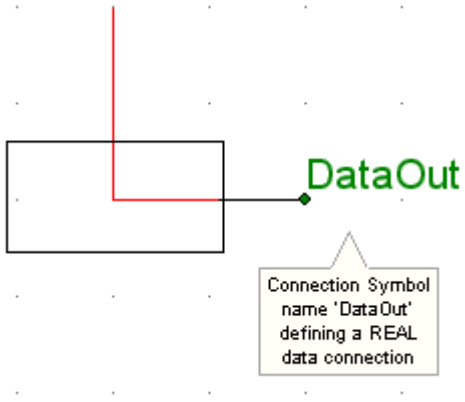
```
                                        OUT = 2.0*PI*RT_1
!
```

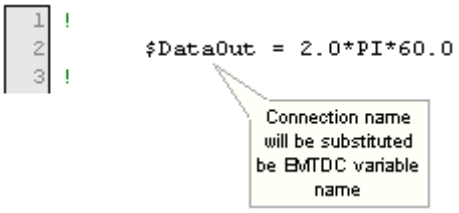
Код Фортрана после компиляции

ПРИМЕЧАНИЕ: Значение входного параметра затененившего параметр *freq* - это результирующее значение после обработки кода. Обращайтесь к разделу Unit System (Системы Единиц) в главе *Operations and Feature Overview (Обзор функций и свойств)* для получения более подробной информации.

ПРИМЕР 10-2:

Значение выходного порта под названием *DataOut* определено в коде компилятора Фортран. При компиляции проекта внутренний компилятор укажет переменную *DataOut* для обозначения порта в связанном файле Фортрана: Оператор $ обеспечивает правильное размещение замененной переменной.





Графика компонента, отображающая соединение

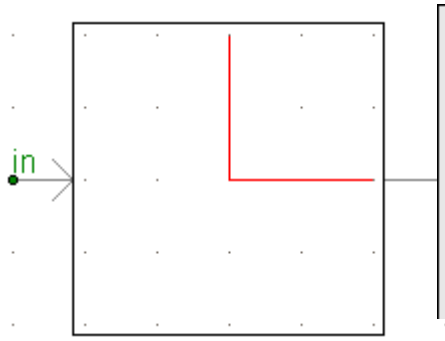
Замена имени порта в коде компилятора Фортран

```
!
      REAL    RT_1        ! EMTDC variable to represent 'DataOut'
!
      RT_1 = 2.0*PI*60.0
!
```

Код компилятора Фортран после компиляции

ПРИМЕР 10-3:

Пользовательский компонент использует функцию, которая требует преобразования значения угла из градусов в радианы. В этом случае вход представляет порт связи, определенный в разделе Графика компонентного определения и обозначен как *in*. Выходная функция под названием *out* представляет выходной порт.



Графический компонент, содержащий порты

```
#FUNCTION REAL PH_CON Phase Angle Converter
!
      $out = PH_CON($in)
!
! When this Script is compiled by PSCAD, $in will be substituted
! by whatever EMTDC decides to name the data node to which 'in' is
! connected (ex. RT_1). $out will substituted by whatever EMTDC
! decides to name the data node to which 'out' is connected
! (ex. RT_2).
!
```

Сегмент сценария компилятора Фортран

```
!
      REAL    PH_CON          ! Phase Angle Converter
      REAL    RT_1, RT_2
!
      RT_2 = STOF(ISTOF + 2)  ! EMTDC extracts port connection value
                              ! from stored data array
                              ! (модуль EMTDC
                              ! извлекает значение порта связи из массива
                              ! хранения данных)
      RT_1 = PH_CON(RT_2)
!
```

Код компилятора Фортран после компиляции

ПРИМЕЧАНИЕ: Для получения более детальной информации по поводу декларирования функций в компиляторе Фортран обращайтесь к разделу с описанием команды #FUNCTION.

ПРИМЕР 10-4:

Пользовательское компонентное определение под названием *my_new_comp* запрограммировано для отображения ключевых входных параметров, а также имени Определения в комментариях в сегменте компилятора Фортрана, а именно: *Frequency* (Частота) с именем *freq*, значением *60.0* и единицами измерения *Hz*, а также *Voltage* (Напряжение) с именем *volts*, значением *120.0* и единицами измерения *kV*. Обратите внимание, что оба параметра *freq* и *volts* являются местными переменными и используется краткая форма при обращении к ним.

Frequency	Real
Description	Frequency
Symbol	freq
Default units	Hz
Minimum value	0.01
Maximum value	1e+008
Data type	Literal
Default value	60.0
Conditional statement	

Свойства для входного поля (*freq*)

Voltage	Real
Description	Voltage
Symbol	volts
Default units	kV
Minimum value	-1e+008
Maximum value	1e+008
Data type	Literal
Default value	120.0
Conditional statement	

Свойства для входного поля (*volts*)

При анализе компонента именам в поле Symbol (*freq* и v*olts*) присваиваются значения указанные в соответствующих полях. В нашем примере *freq = 60.0* и *volts = 120.0*. Пользователь имеет возможность использовать эти переменные в любом месте сценария при использовании оператора $.

Похожие комментарии могут появиться в сегментах Fortran, DSDYN или DSOUT:

```
1 ! User-Defined Component: $(Defn:Name)
2 !
3 ! Frequency: $freq Hz,  Voltage: $volts kV
4 !
```

Код в сегменте Фортран

```
1 ! User-Defined Component: my_new_comp
2 !
3 ! Frequency: 60.0 Hz,  Voltage: 120.0 kV
4 !
```

Код компиляционного файла Фортран в PSCAD после выполнения компиляции

Это свойство может быть очень полезным для форматирования комментариев источника кода.

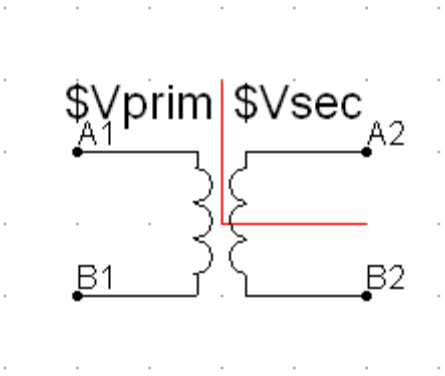
Буквенные значения также могут быть замещены напрямую в текстовых полях в разделе Graphic (Графика) при использовании оператора $. Это полезное свойство для примера может быть использовано для изменения текста графического компонента в зависимости от значения входных параметров.

ПРИМЕР 10-5:

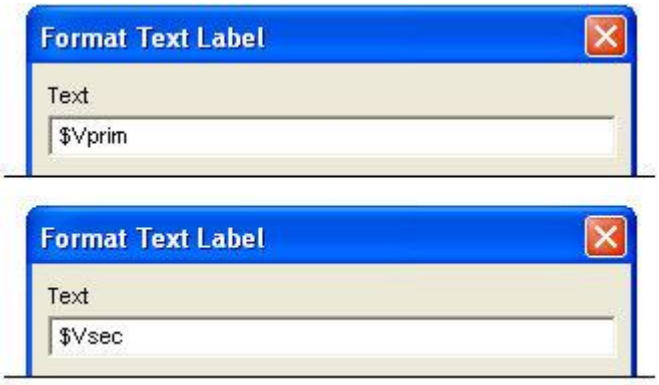
Пользователь разработал компонент однофазного двухобмоточного трансформатора и планирует вывести значения первичного и вторичного напряжения непосредственно на графическом отображении компонента. Первичное напряжение трансформатора указано как 25 *[kV]*, вторичное напряжение составляет *4.0 [kV]*:

Winding #1 voltage (RMS)	230.0 [kV]
Winding #2 voltage (RMS)	230.0 [kV]

В разделе Graphic(Графика) пользователь добавляет две новые записи (расположенные на нужных местах) и вводит значения для *$Vprim* и *$Vsec* как показано ниже:

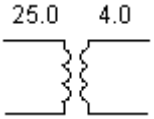


Графическое изображение компонента с текстовыми надписями



Свойства графических надписей

При просмотре компонента на рабочем пространстве Circuit (Схема), входные значения напряжений *Vprim* и *Vsec* появятся непосредственно на графическом компоненте. Для приведенного выше примера,



Контекстная замена текста

Контекстная замена – это формальный способ обработки текстовых замен за пределами Определения. Контекстная замена активизируется при использовании ограничивающих символов следующих за оператором $. Предусмотрен следующий синтаксис:

```
$(<Context>:<Key>)
```

Параметрами для замены являются две части - <Context> и <Key>, разделенные двоеточием и ограниченные круглыми скобками:

- **<Context>**: Этот тест используется для поиска параметра <Key>. Ниже приведен список допустимых имен <Context>.
- **<Key>**: Заменяемый текст (действительная замена), который должен появиться в месте замещения. Ниже приведен список допустимых имен <Key>.

Контекстная замена может быть использована в текстовых полях в секции Graphic (Графика), а также в сегментах Fortran, DSDYN, DSOUT, Branch (Ветвь), Model-Data, Matrix-Fill, Transformers (Трансформаторы) или в T-Lines (Линии электропередачи). Кроме того, она может быть использована в глобальных заменах и различных ключах в соответствующем контексте.

В следующей таблице приведены все наименования, которые могут быть использованы как параметры <Context> и <Key> (обратите внимание, что оба параметра *Context* и *Key* чувствительны к регистру).

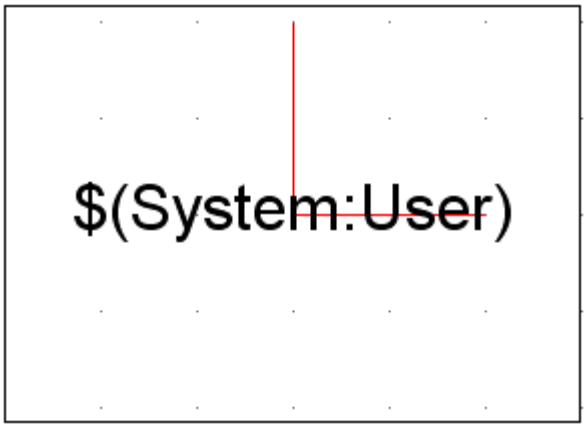
Context Name	**Description**	**Available Keys**
<none>	Пустое поле создается автоматически для компонентного Образа	
Defn	Областью поиска является Определение компонентного Образа	Name, Desc, Path
Project	Областью поиска является Project (Проект), в котором содержатся компонентные Образы	Name, Desc, Version, Author, Path
Session	Областью поиска является текущая сессия PSCAD.	Name, Version
System	Областью поиска является операционная система.	Name, Version, User

При использовании замены в пределах компонентного Образа, имеется возможность не включать идентификатор <Context> и круглые скобки. При этом синтаксис выглядит следующим образом:

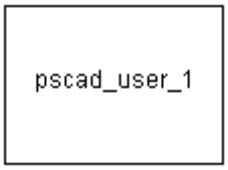
`$(<Key>)` или еще проще `$<Key>`.

ПРИМЕР 10-6:

Пользователь создал Определение и хочет отобразить имя пользователя на графическом изображении компонента. В разделе Graphic (Графика) добавляется текстовая надпись содержащая оператор $ с требуемым значением.



Графическое отображение компонента, содержащее текстовую надпись



Образ компонента на рабочем пространстве

$#DIM: Замена размерности порта

Размерность любого порта может быть заменена на целое число непосредственно в сценарии компонента при использовании оператора *$#DIM*. Оператор *$#DIM* может использоваться в сегментах Fortran, DSDYN, DSOUT и Checks (проверки). Формат записи для оператора имеет следующий вид:

```
$#DIM(<Node_Name>)
```

Где:

- **<Node_Name>**: (**<**Имя_Узла**>**): Наименование порта соединения.

Пример использования рассмотренного оператора приведен ниже:

```
!

#IF $#DIM(NT)==3 || $#DIM(NF)==3  ! Used to define #IF logic based on the
dimension of port NT. (! Используется для определение условия #IF в
зависимости от размерности порта NT.)

!

ERROR Dimesion Exceed - Maximum 3 phase : (Превышена размерность –
максимально возможно 3 фазы) $#DIM(NT)<4 && $#DIM(NF)<4 ! Used in the Checks
segment.( ! Используется в разделе проверок.)

!

IF (($si .GT. 0).AND.($si .LE. $#DIM(in))) THEN ! Used directly as a
substitution in Fortran code. (Используется непосредственно как замена в коде
компилятора Фортран)

!
```

$#Component: Замера идентификатора компонентного Образа

Идентификатор любого Образа может быть заменен при использовании оператора *$#Component*. Оператор *$#Component* может использоваться в сегментах Fortran, DSDYN and DSOUT. Формат записи рассматриваемого оператора принимает следующий вид:

```
$#Component
```

Основной целью использования этого оператора является предоставление идентификатора Образа вызываемого компонента модулю EMTDC для определения источника возникновения сообщения. Таким образом, PSCAD может установить обратную связь с панелью сообщений в процессе расчета, если сообщение сгенерировано определенным компонентом. Пример использования этого оператора приведен ниже:

```
!

CALL COMPONENT_ID(ICALL_NO,$#Component)  ! Used to assign Call number and
Instance number of a component. These values are (Используются для
присваивания параметров компонента - Call number (Номер вызова) и  Instance
number (Номер Образа))

                                         ! passed to EMTDC via the
COMPONENT_ID intrinsic subroutine. (попадают в модуль EMTDC через внутреннюю
подпрограмму COMPONENT_ID)

!
```

% Префикс оператора замены данных

Префикс замены данных % используется для прямой замены текста. Замена данных очень похожа на префикс замены значения $, за исключением того, что возможно заменить значение параметра в текстовом поле и может использоваться только для замены входных параметров.

Оператор % может использоваться в текстовых надписях раздела Graphic (Графика), а также в закоментированном коде в сегментах Fortran, DSDYN, DSOUT, Branch (Ветвь), Model-Data, Matrix-Fill, Transformers (Трансформаторы) или T-Lines (Линии электропередач).

ПРИМЕР 10-7:

Рассмотрим Пример 10-4 при использовании оператора % вместо оператора $. Приведенные ниже комментарии могут появиться в разделах компилятора Фортран, DSDYN или DSOUT:

```
1 ! User-Defined Component: $(Defn:Name)
2 !
3 ! Frequency: %freq,  Voltage: %volts
4 !
```

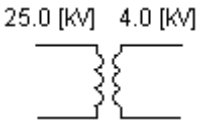
Сегмент кода Фортрана

```
1 ! User-Defined Component: my_new_comp
2 !
3 ! Frequency: 60.0 [Hz],  Voltage: 120.0 [kV]
4 !
```

Код файла компилятора Фортран PSCAD после компиляции

ПРИМЕР 10-8:

В рассмотренном выше примере 10-5 заменим оператор $ на оператор % в текстовых надписях. Теперь значения *Vprim* и *Vsec* будут отображаться следующим образом:



{ } Скобки

Скобки используются при выполнении математических, логических и форматирующих действий и обрабатываются компилятором в первую очередь. Скобки могут использоваться в любом месте компонентного Определения. Используются, как правило, в двух случаях: Замена выражений и блок обработки. Ниже приведен требуемый синтаксис:

```
[$]{<Expression>}
```

Где:

- **$**: Optional (Опционально). В некоторых случаях значок префикса замены значения $ можно не применять, например, для большинства блоков обработки.
- **<Expression>**: (**<Выражение>**:) Может быть математическим выражением или текстом.

Важно отметить, что математические и логические действия внутри кавычек обрабатываются перед вставкой кода в файлы данных или компилятора Фортран. Следовательно, это операторы не могут использовать переменные значения – только буквенные и константы (такие как постоянные входные параметры, местные константы или константы определенные в сегменте Computations (Вычисления)).

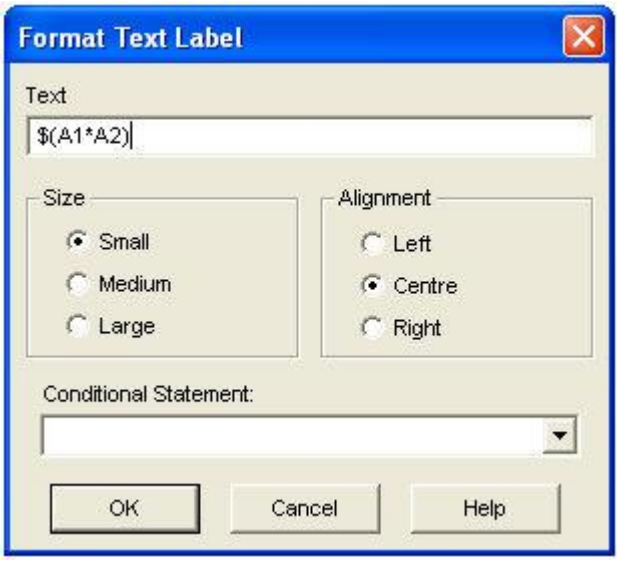
Некоторые примеры приведены ниже, с дополнительными примерами можно ознакомиться при изучении компонентных Определений в библиотеке стандартных компонентов.

ПРИМЕР 10-9:

Следующий пример демонстрирует, как выражения в кавычках могут быть использованы в текстовых надписях: математические операции используют замещенные значения. Предположим, что вводятся два входных параметра с именами *A1* и *A2* со значениями *2.0* и *3.0.* Пользователь хочет отобразить значение, полученное в результате умножения указанных значений:

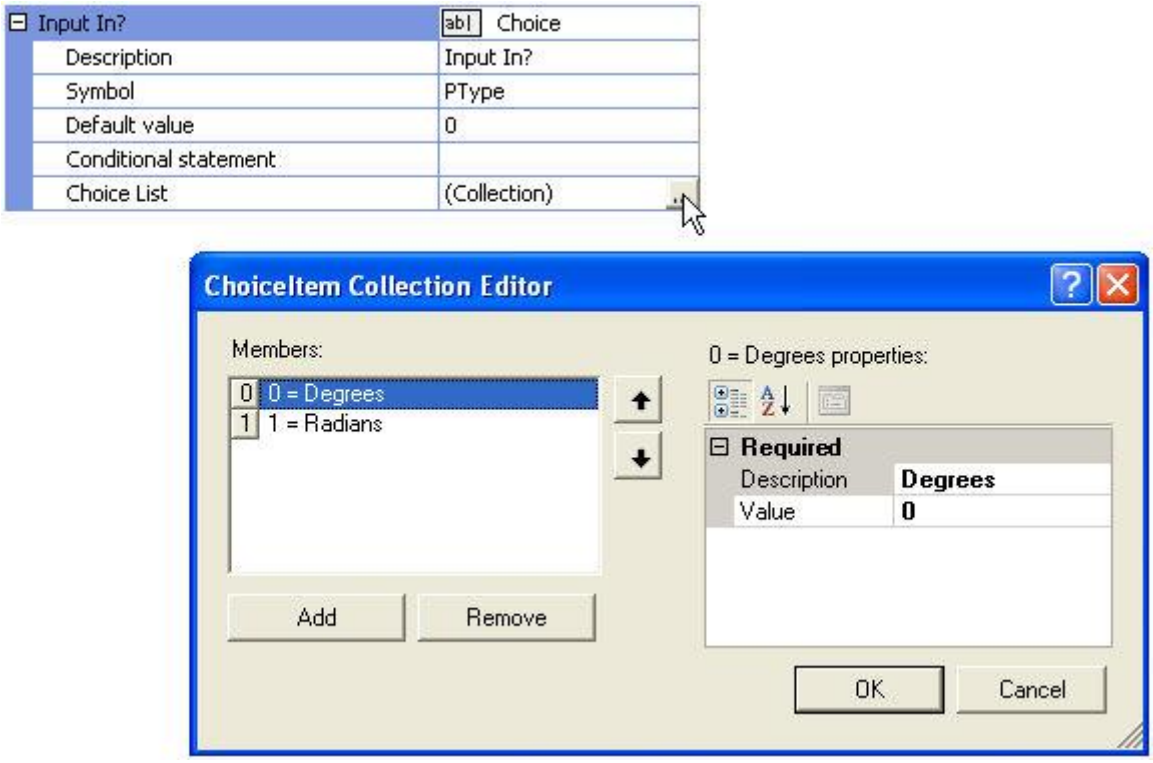
Text Label = A1*A2 = 6.0

Следующее значение вводится в текстовое поле в диалоговом окне текстовой надписи:



ПРИМЕР 10-10:

Пользователь изменил функцию *PH_CON* в примере 10-3, чтобы преобразование фазного угла осуществлялось в обе стороны (то есть углы в радианы и радианы в углы). В разделе Parameters (Параметры) список с выбором под названием *Ptype* предусматривает два варианта действия как показано ниже:



Редактор списка с выбором (*Ptype*)

Функция *PH_CON* изменена таким образом, чтобы учитывать второй аргумент, отвечающий за тип необходимого преобразования в зависимости от состояния *Ptype*. Дополнительный аргумент является строкой символов, в которой параметр *D2R* отвечает за преобразование градусов в радианы, а параметр *R2D* – за преобразования радианы в градусы. В синтаксисе используются скобки, а также функция а #CASE и оператор ~ Line Continuation:

```
1 #FUNCTION REAL PH_CON Frequency Converter
2 !
3       $out = PH_CON($in,~
4 !
5 #CASE FType {~'D2R'} {~'R2D'}
6 !
```

Сценарий сегмента компилятора Фортран

```
!

      RT_2 = F_CON(RT_1,'D2R')

!
```

Код компилятора Фортран после выполнения компиляции

RT_1 и *RT_2* – это замены переменных произведенные модулем EMTDC: Их имена могут меняться.

! Оператор ввода комментариев

Символ восклицательного знака используется компонентным Определением для указания на строку с комментарием. Все строки кода, которые начинаются с этого оператора, будут рассматриваться как комментарии для включения в проектные файлы компилятора Фортран или файлы данных.

> **ПРИМЕЧАНИЕ**: Закомментированный код, который записывается в файл данных, должен начинаться с оператора комментария на первом знаковом месте слева.

ПРИМЕР 10-11:

Комментарии используются при описании пользовательского кода, а также для организации параграфов и промежутков между разделами для улучшения читаемости кода. Оператор ввода комментариев может использоваться везде кроме раздела Branch (Ветвь).

Приведенный ниже пример демонстрирует использование комментариев в разделе компилятора Фортран или DSDYN пользовательского Определения:

```
!
! MY FIRST PSCAD SCRIPT
! ----------------------
!
! Storage:
!
#STORAGE REAL:1
!
! Main body of script:
!
```

> **ПРИМЕЧАНИЕ**: Обращайтесь к разделу с описанием функции #STORAGE для получения более подробной информации.

Директивы сценария

#IF, #ELSEIF, #ELSE, #ENDIF

#CASE

#STORAGE

#LOCAL

#BEGIN/#ENDBEGIN

#FUNCTION

#SUBROUTINE

#OUTPUT

#TRANSFORMERS

#WINDINGS

#VERBATIM

Директивы сценария используются для организации кода, который будет обрабатываться при создании проектного исполняемого кода. Каждая директива сценария имеет особое назначение в соответствии с приведенным ниже описанием. Директивы сценария всегда начинаются с префикса #, при этом допустимы символы пробела до и после символа #.

Директивы представляют важный инструмент написания сценариев, поскольку они дают указание внутреннему компилятору какой кусок кода компилировать и каким образом. Рекомендуется по возможности использовать директивы сценариев, поскольку они гарантируют, что пользовательский сценарий совместим со стандартами кодирования при последующих изменениях и модификациях.

#IF, #ELSEIF, #ELSE, #ENDIF

Директивы #IF, #ELSEIF, #ELSE, #ENDIF используются как логические структуры подобно встроенным функциям IF, ELSE и так далее. Однако внутри сценария они используются для обработки или исключения разделов сценария.

Директивы #IF, #ELSEIF, #ELSE, #ENDIF могут использоваться во всех сегментах и должны выглядеть следующим образом:

```
#IF <Logic> <Логическое выражение>
      ...Application_Code...       (...Применяемый_код...)
#ELSEIF <Logic> <Логическое выражение>
      #IF <Logic> <Логическое выражение>
            ...Application_Code... (...Применяемый_код...)
      #ELSE
            ...Application_Code... (...Применяемый_код...)
      #ENDIF
#ELSE <Logic> <Логическое выражение>
      ...Application_Code... (...Применяемый_код...)
#ENDIF
```

При использовании условия IF-THEN может использоваться краткое выражение:

```
#IF <Logic> {<Expression>}<Логическое выражение> {<Выражение>}
```

<Logic>(<Логическое выражение>) - это логическое выражение с логическими операторами. <Expression> (<Выражение>) – может переменным Определением.

ПРИМЕР 10-12:

Пользователь планирует изменять значение выходного сигнала в зависимости от выбранной функции синуса или косинуса. Компонентное Определение предоставляет список выбора с входными параметрами в разделе Parameters (Параметры) под названием *Type*. При значении этого параметра равного *1* выходная функция является синусоидальной (при выборе 0 функция является косинусоидальной).

Приведенный ниже код должен появиться в разделах компилятора Фортран, DSDYN или DSOUT:

```
!
! Signal Generator (Генератор сигналов)
!
#IF Type == 1
      $OUT = SIN(TWO_PI*$F)
#ELSE
      $OUT = COS(TWO_PI*$F)
#ENDIF
!
```

Где *F* – это заранее определенная переменная (возможно входной параметр или переменная из раздела Computations (вычисления)), а *OUT* – это выходной порт связи в разделе Graphic (Графика).

> **ПРИМЕЧАНИЕ:** *Type == 1* в приведенном выше коде – это Logical Expression (Логическое выражение). Обращайтесь к разделу Expression Evaluation (Оценка выражений) для получения более подробной информации.

При использовании оператора Enfolding Operators (Скобки) вышеприведенные примеры будут представлены в следующем виде:

```
!
! Signal Generator (Генератор сигналов)
```

```
!

#IF Type == 1 {       $OUT = SIN(TWO_PI*$F)}

#IF Type != 1 {       $OUT = COS(TWO_PI*$F)}

!
```

EXAMPLE 10-13:

Пользователь разработал электрический компонент, который может быть резистором, индуктивностью или конденсатором. Компонентное Определение предоставляет список с выбором в разделе Parameters (Параметры) под названием *Type* . Этот параметр может принимать значения *1*, *2* или *3* для выбора резистивного сопротивления, индуктивности или емкости. Также представлены три текстовых окна (под названием *R*, *L* или *C*) для задания значения указанных элементов.

Следующий код должен появиться в разделе Branch (Ветвь):

```
#IF Type == 1

      $N1   $N2   $R   0.0   0.0

#ELSEIF Type==2

      $N1   $N2   0.0   $L   0.0

#ELSE

      $N1   $N2   0.0   0.0   $C

#ENDIF
```

N1 и *N2* представляют электрические порты определенные в разделе Graphic (Графика). При использовании оператора Enfolding Operators (Скобки) приведенный выше пример принимает следующий вид:

```
!

#IF Type == 1 {$N1   $N2   $R    0.0   0.0}

#IF Type == 2 {$N1   $N2   0.0   $L    0.0}

#IF Type == 3 {$N1   $N2   0.0   0.0   $C}

!
```

#CASE

Директива #CASE представляет краткую форму обозначения, которая может быть использована в директивах #IF, #ELSEIF, #ELSE, #ENDIF. Обычно он используется совместно с оператором ~ Line Continuation Operator для создания более компактного кода, особенно при использовании большего количества директив #IF, #ELSEIF, #ELSE, #ENDIF.

Директива #CASE может использоваться во всех сегментах и принимает следующий вид:

```
#CASE <Expression> {<Clause_0>} {<Clause_1>} ...
```

<Expression> (<Выражение>) должно возвращать целое число от *0* до *n*, может быть математическим выражением или предопределенным значением. <Clause_n> представляет событие, которое произойдет в зависимости от значения в поле <Expression>. Например, произойдет событие <Clause_0>, если <Expression> равно 0. Событие <Clause_1> произойдет, если <Expression> равно 1, и так далее.

EXAMPLE 10-14:

Рассмотрим выход генератора сигналов, рассмотренный в Примере 10-12:

```
!
! Signal Generator (Генератор сигналов)
!
#IF Type == 1
      $OUT = SIN(TWO_PI*$F)
#ELSE
      $OUT = COS(TWO_PI*$F)
#ENDIF
!
```

Ниже приведен пример равнозначного сценария при использовании директивы #CASE:

```
!
! Signal Generator(Генератор сигналов)
!
       $OUT = ~
```

```
#CASE Type {~COS~} {~SIN~}

~(TWO_PI*$F)

!
```

Обратите внимание, что *Type* должен принимать значение 0 или 1.

ПРИМЕР 10-15:

Ниже приведено описание директивы #CASE при использовании для 2-х обмоточного 3-х фазного трансформаторного компонента в разделе Transformers (Трансформаторы) из библиотеки стандартных компонентов.

В приведенном примере *YD1* и *Lead* представляют компонентные входные параметры. Параметр *YD1* может принимать значения 0 или 1, а параметр *Lead* значения *1* или *2*. Умножение этих двух целочисленных значений дает результат *0*, *1* или *2*, оно определяет используемое в расчете событие.

```
!

#CASE YD1*Lead {$A1 $G1~} {$A1 $B1~} {$A1 $C1~}

!
```

#STORAGE

Эта директива применяется в Определении только при использовании Storage Arrays (Массив памяти) модуля EMTDC. Она используется для определения показателей использования памяти – тип и количество требуемой памяти модуля EMTDC. Директива #STORAGE – это интерфейс массивов памяти, который обеспечивает возможность передачи данных из одного расчетного шага в следующий, а также перенос данных между разделами BEGIN и соответствующими разделами DSDYN или DSOUT. Использование массива данных модуля EMTDC должно быть определено через директиву #STORAGE для гарантии того, что память имеет правильную размерность в PSCAD во время компиляции.

Директива #STORAGE используется только в разделах компилятора Фортран, DSDYN или DSOUT и должна выглядеть следующим образом:

```
#STORAGE <TYPE>:<Number>
```

Запись #STORAGE может появиться в любом месте в разделе, но должна быть размещена в верхней части. <<Number> – это количество элементов для хранения, может быть замещенной константой.

Все приведенные ниже примеры с хранением данных корректны при использовании в рассматриваемой директиве оператора $ Substitution Prefix Operator (Префикс оператора замены значения):

```
#STORAGE REAL:$(X) INTEGER:$(Y)              // X and Y are literal parameters
                                              (X и Y – буквенные параметры)
#STORAGE REAL:$#DIM(N1) INTEGER:$#DIM(N2)   // N1 & N2 are electrical ports
                                              (N1 и N2 – электрические порты)
#STORAGE INTEGER:$#DIM(N1) LOGICAL:7
#STORAGE REAL:${X+Y}
```

	Тип	(массив данных модуля EMTDC)	Описание
Передача данных между расчетными шагами	STOR	STOR(NEXC)	Этот массив использовался в PSCAD V2 и является устаревшим. По возможности не используйте этот массив.
	REAL	STORF(NSTORF)	Только для хранения чисел с плавающей точкой (REAL).
	INTEGER	STORI(NSTORI)	Только для хранения типа INTEGER.
	LOGICAL	STORL(NSTORL)	Только для хранения типа LOGICAL.
	COMPLEX	STORC(NSTORC)	Только для хранения типа COMPLEX.
Передача данных из разделов BEGIN в DSDYN/DSOUT	RTCF	RTCF(NRTCF)	Только для хранения чисел с плавающей точкой (REAL).
	RTCI	RTCI(NRTCI)	Только для хранения типа INTEGER.
	RTCL	RTCL(NRTCL)	Только для хранения типа LOGICAL.
	RTCC	RTCC(NRTCC)	Только для хранения типа COMPLEX.

ПРИМЕР 10-16:

Рассмотрим раздел сценария из раздела компилятора Фортран компонента передаточной функции XYZ библиотеки стандартных компонентов:

```
#IF NL==0

#STORAGE INTEGER:2 REAL:110

#ELSEIF NL==1

#STORAGE INTEGER:2 REAL:1100

#ELSEIF NL==2

#STORAGE INTEGER:2 REAL:11000

#ENDIF

#FUNCTION REAL XYZFUNC XYZ-Transfer function

! File name: $FILE

      $Z = XYZFUNC($X,$Y,$MODE,$Xoff,$Yoff,$Zoff,$Kx,$Ky,$Kz)
```

В этом сценарии используются директивы #IF, #ELSEIF, #ELSE, #ENDIF, а также #STORAGE для определения требований к массиву памяти модуля EMTDC для этого компонента. В зависимости от значения переменной *NL*, которая в этом компоненте является входным параметром *Number of Lines in the File* (***Количество линий в файле***), число элементов типа REAL принимает значения от 110 до 11000. Число элементов типа INTEGER всегда равно 2.

Обратите внимание, что выше приведена информация как рассматривается память с переменными параметрами до введения замещения в директивах хранения. Наиболее простым и эффективным способом является нижеприведенный код, где переменная *NREAL* определяется входным параметром *NL*, (*Number of Lines in the File - **Количество линий в файле**),* и замещается размерностью хранения типа REAL:

```
#STORAGE INTEGER:2 REAL:$(NREAL)

#FUNCTION REAL XYZFUNC XYZ-Transfer function

! File name: $FILE

      $Z = XYZFUNC($X,$Y,$MODE,$Xoff,$Yoff,$Zoff,$Kx,$Ky,$Kz)
```

Эта запись представляет тип и число элементов массива памяти модуля EMTDC, которые используются функцией *XYZFUNC.* Обратите внимание, что код функции *XYZFUNC* использует массивы *STORI* и *STORF* для хранения и перемещения данных из одного расчетного шага в другой в процессе моделирования.

#LOCAL

Эта директива используется для объявления локальных переменных непосредственно в сценарии Определения. Достаточно часто возникает необходимость объявлять локальные переменные, например, для определения фиктивной переменной для неиспользуемого аргумента подпрограммы или, например как промежуточное значение при вычислениях.

Директива #LOCAL указывает компилятору PSCAD разместить объявление о локальной переменной непосредственно в главный модуль файла компилятора Фортран при построении проекта. Переменные типа #LOCAL не требуют применения оператора $ Value Substitution Prefix Operator (Оператора замены значения).

Директивы #LOCAL используются только в разделах Fortran, DSDYN или DSOUT; они должны принимать следующий вид:

```
#LOCAL <TYPE> <Name> <Array_Size_1> <Array_Size_2>
```

<TYPE> может принимать значения REAL, INTEGER, LOGICAL, или COMPLEX. <Name> - назначенное имя для локальной переменной. <Array_Size_1> - дополнительное целое число или замененная константа, которая определяет размерность массива. При объявлении матрицы параметр <Array_Size_1> определяет число рядов, а <Array_Size_2> – число столбцов. Если переменная является скалярной величиной, то оставьте значения <Array_Size_1> и <Array_Size_2> пустыми.

Приведенные ниже примеры с параметрами памяти являются корректными при использовании оператора $ Substitution Prefix Operator (Оператор замены префикса) в рассматриваемой директиве:

```
#LOCAL REAL X $#DIM(N1) 2          // N1 is an electrical port
#LOCAL REAL Y $#DIM(N1) $#DIM(N2)  // N1 & N2 are electrical ports
#LOCAL INTEGER Z $(XXX)            // XXX is a literal parameter
#LOCAL INTEGER ZZZ ${X-Y}          // X and Y are literal parameters
#LOCAL COMPLEX D 2                 // D is a locally declared, complex array
```

EXAMPLE 10-17:

Пользовательский компонент использует две локальные переменные как аргументы подпрограммы. При использовании условий применения, основанных на значении переменной *A*, локальная целочисленная переменная *MY_X* и локальный массив данных типа REAL под названием *Error* определяются до вызова подпрограммы.

Объявление #LOCAL выглядит следующим образом:

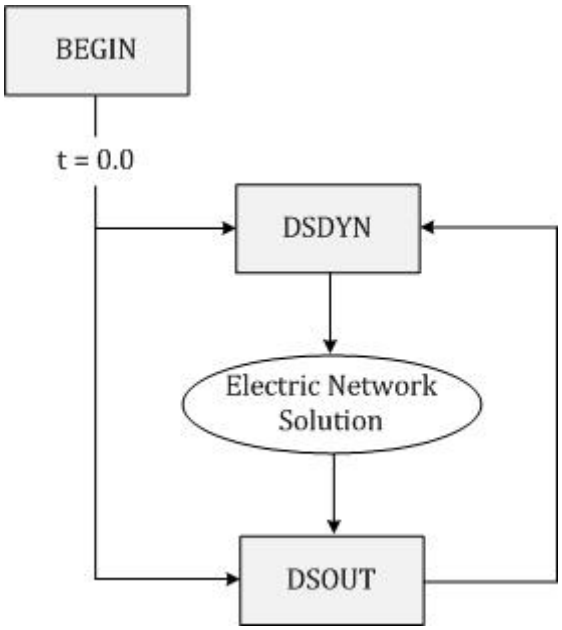
```
#LOCAL INTEGER MY_X
#LOCAL REAL Error 2
!
#IF A > 1
      MY_X = 1
      Error(1) = 0.2
```

```
#ELSE

      MY_X = 0

      Error(2) = 0.8

#ENDIF

!

      CALL SUB1(MY_X, Error)

!
```

NOTE: Выражение *A > 1* в приведенном выше коде является Logical Expression (Логическим Выражением). Обращайтесь к разделу Expression Evaluation (Оценка Выражений) для получения более подробной информации.

#BEGIN/#ENDBEGIN

Этот директивный блок предоставляет доступ к внешнему процессу BEGIN модуля EMTDC, который является модульной подпрограммой, вызываемой перед разделами DSDYN и DSOUT в нулевой момент времени.



Этот уровень процесса обеспечивает свойство *Runtime Configuration* (*Конфигурация расчета*) для поддержки существования компонентов в модулях с множеством Образов. Для получения более подробной информации обращайтесь к разделу Multiple Instance Modules (Модули с множеством Образов) в Главе 2 руководства модуля EMTDC.

Директивный блок #BEGIN/#ENDBEGIN используется только в разделах компилятора Фортран, DSDYN или DSOUT и используется в следующем виде:

```
#BEGIN

   ...

#ENDBEGIN
```

Содержание директивного блока является произвольным и зависит от проектируемого компонента – но в любом случае требует выполнения минимального набора правил. Использование блока #BEGIN гарантирует, что код обрабатывается только во время компиляции, при этом логическая проверка *TIMEZERO* на каждом расчетной шаге не производиться; таким образом, гарантируется эффективная расчетная скорость. Блок #BEGIN содержит следующие общие части:

- **Time Zero Initialization (Инициализация в нулевой момент времени)**: Весь инициализационный код для пользовательского компонента размещенный внутри директивного блока #BEGIN будет вставлен как код компилятора Фортран в разделы DSDYN или DSOUT подпрограммы BEGIN (в соответствии с главным модулем) при компиляции проекта.

- **Component Instance and Call Numbers (Образ компонента и номера вызова)**: Если существует вероятность появления предупредительных сообщений или сообщений об ошибках, выдаваемых в пределах блока #BEGIN, то должны быть предоставлены номера Образа и вызова для модуля EMTDC. В случае возникновения сообщения, источник сообщения может быть определен внутри PSCAD при использовании этих номеров. Номера Образа и вызова предоставляются при использовании подпрограммы *COMPONENT_ID*. Обращайтесь к разделу Custom Model Design (Проектирование Пользовательской Модели) Главы 5 для получения более подробной информации.

ПРИМЕР 10-18:

Одним из наиболее простых примеров из библиотеки стандартных компонентов, который использует код в блоке BEGIN, является экспоненциальная функция. Этот компонент моделирует экспоненциальную функцию, которая представляет основание и экспоненциальный коэффициент:

$$y = A \cdot e^{B \cdot x} \text{ or } y = A \cdot 10^{B \cdot x}$$

Коэффициенты *A* и *B* объявлены как константы в том смысле, что значения могут определяться как переменные, которые принимают различные значения в зависимости от конкретного Образа главного модуля (то есть от рабочего пространства, на котором размещен компонент). Таким образом, эти входные величины должны храниться в последовательности в пределах массивов памяти BEGIN, чтобы была возможность извлечь значения в нужной последовательности, что определяется самим Образом.

Ниже приведена упрощенная версия сегмента компилятора Фортран для компонентного Определения экспоненциальной функции (подразумевается, что *e* является основанием и a - скалярным входным сигналом):

```
#BEGIN

      RTCF(NRTCF)    = $A

      RTCF(NRTCF+1) = $B

      NRTCF = NRTCF + 2

#ENDBEGIN

#STORAGE RTCF:2
```

```
!

      $OUT = RTCF(NRTCF) * EXP(RTCF(NRTCF+1) * $IN)

      NRTCF = NRTCF + 2

!
```

Директивный блок #BEGIN/#ENDBEGIN размещенный в верхней части этого сценария гарантирует, что код размещен компилятором в разделе BEGIN системной динамики: коэффициенты *A* и *B* сохранены в нужной последовательности в соответствии с текущим значением показателя *NRTCF*.

Остальная часть сценария за пределами директивного блока #BEGIN/#ENDBEGIN размещена в разделах DSDYN или DSOUT системной динамики. Отметим, что значения коэффициентов *A* и *B* (то есть *RTCF(NRTCF)* и *RTCF(NRTCF+1)* соответственно) извлекаются из участка памяти, в котором они были сохранены в нулевой момент времени. Показатель *NRTCF* при этом увеличивается.

Обращайтесь к разделу EMTDC Storage Arrays (Массивы памяти модуля EMTDC) Главы 5 для получения более подробной информации о функционировании.

#FUNCTION

Эта директива используется для объявления функции, а также типа аргументов, которые она возвращает. Применение директивы #FUNCTION является обязательным, если функция используется в пределах компонентного Определения: это будет гарантировать, что объявление функции размещено в пределах подпрограммы источника, в которой размещен компонентный код.

Директива #FUNCTION используется только в разделах компилятора Фортран, DSDYN или DSOUT и выглядит следующим образом:

```
#FUNCTION <TYPE> <Name> <Description>
```

<TYPE> может быть REAL, INTEGER или LOGICAL. <Name> - присвоенное имя функции. <Description> - будет включена как строка комментария около начала соответствующей модульной подпрограммы источника.

ПРИМЕР 10-19:

Компонент Hard Limiter (Фиксированный ограничитель) из библиотеки стандартных компонентов использует функцию вещественного типа под названием *LIMIT* для определения параметра для внешнего порта O в зависимости от входных параметров *LL*, *UL*, или *I*. *LL* и *UL* представляют входные параметры *Lower Limit (Нижний предел)* и *Upper Limit (Верхний Предел)* соответственно, где переменная *I* зависит от состояния входного порта в разделе Graphic (Графика).

Следующий код появиться в разделе Фортран компонентного Определения Hard Limiter (Фиксированный ограничитель).

```
#FUNCTION REAL LIMIT Hard Limiter

!

      $O = LIMIT($LL, $UL, $I)

!
```

#SUBROUTINE

Эта директива используется для описания подпрограммы, которая вызывается из компонента. #SUBROUTINE используется для лучшего представления компонента, но не является обязательной (хотя и рекомендуется к использованию).

Директива #SUBROUTINE используется только в сегментах компилятора Фортрана, DSDYN или DSOUT и выглядит следующим образом:

```
#SUBROUTINE <Name> <Description>
```

<Name> - это присвоенное имя подпрограммы. <Description> будет отображаться как строка комментария около начала соответствующей модульной подпрограммы источника.

ПРИМЕР 10-20:

Пользователь включает вызов подпрограммы под названием SUB1 в сценарий компонентного Определения. Рекомендуется использовать комментарии.

В разделе компилятора Фортрана, DSDYN или DSOUT появиться следующий код:

```
#SUBROUTINE SUB1 User Subroutine (Пользовательская подпрограмма)

!

      CALL SUB1($X, $Y, $Z)

!
```

X, *Y* и *Z* – это предопределенные переменные, которые могут быть портом связи, переменными раздела Computations (Вычисления) или входными параметрами.

#OUTPUT

Эта директива извлекает определенные значения из расчетных данных таким образом, что они могут отслеживаться, выводиться на печать или использоваться внешними компонентами. #OUTPUT выполняет две задачи: определяет новую переменную, а затем присваивает ей значение в соответствии с выражением.

Директива #OUTPUT используется только в разделах компилятора Фортран, DSDYN или DSOUT и выглядит следующим образом:

```
#OUTPUT <TYPE> <Name> <Array_Size> {<Expression>}
```

<TYPE> может быть REAL, INTEGER, или LOGICAL. <Name> - присвоенное имя переменной. <Array_Size> - дополнительное число, определяющее размеры массива. Если переменная является одномерной, то это поле оставляют пустым. <Expression> может быть математическим выражением, участком памяти или переменной.

ПРИМЕР 10-21:

Переменная, заявленная в директиве #OUTPUT, может быть вычислена множеством способов. Ниже приведен список примеров, демонстрирующих возможные методы. Обратите внимание, что библиотека стандартных компонентов также содержит множество примеров использования #OUTPUT в компонентном Определении.

```
! Defines a REAL variable 'freq' and substitutes
! the value of a pre-defined variable 'Fout'.
!(Определяет вещественную переменную 'freq' и замещает значение заранее
!определенной переменной 'Fout')
#OUTPUT REAL freq {$Fout}
!
! Defines an INTEGER variable 'Xon' and assigns
! it the value of a storage location.
!(Определяет целочисленную переменную 'Xon' и присваивает ей значение
! массива хранения)
#OUTPUT INTEGER Xon {STORI(NSTORI+1)}
!
! Defines an REAL variable 'POut' and assigns
! it the value of a given mathematical
```

```
! expression.

!(Определяет вещественную переменную 'POut' и присваивает ей значение из

! математического выражения)

#OUTPUT REAL POut {$V*$I}

!
```

#TRANSFORMERS (Трансформаторы)

Эта директива должна использоваться для учета взаимосвязанных обмоток компонентного Определения. Обычно она используется совместно с директивой #WINDINGS (Обмотки) (описанной ниже). Примерами элементов из библиотеки стандартных компонентов, которые используют рассматриваемую директиву, являются трансформаторы и компоненты с π-секциями.

Директива #TRANSFORMERS выполняет две основные задачи:

1. Предоставляет возможность последовательно пронумеровать все компоненты в пределах проекта, в которых участвуют взаимосвязанные обмотки для определения размерностей матриц и массивов модуля EMTDC.

2. Предоставляет ссылочную информацию для отслеживания токов взаимосвязанных обмоток. Токи обмоток трансформатора измеряются при использовании внутренней матрицы *CDCTR(M,N)* модуля EMTDC. *CDCTR(M,N)* представляет ток в обмотке M трансформатора N.

Директива #TRANSFORMERS используется только в разделе Transformers (Трансформаторы) и принимает следующий вид:

```
#TRANSFORMERS <Number> (<Число>)
```

<Number> (<Число>) указывает на количество трансформаторов в пределах компонента.

ПРИМЕР 10-22:

Трехфазный двухобмоточный классический трансформаторный компонент, расположенный в разделе *Transformers* (Трансформаторы) библиотеки стандартных компонентов состоит из трех однофазных трансформаторов.

Директива в разделе Transformers(Трансформаторы) будет выглядеть следующим образом:

```
#TRANSFORMERS 3

#WINDINGS 2
```

#WINDINGS (Обмотки)

Эта директива используется для задания числа связанных трансформаторных обмоток и обычно используется совместно с директивой #TRANSFORMERS (Трансформаторы). PSCAD проведет поиск по определению наибольшего числа из доступных директив #WINDINGS и присвоит полученное число (как максимальное число обмоток) в Map file (Файл данных).

Директива #WINDINGS используется только в разделе (Трансформаторы) и выглядит следующим образом:

```
#WINDINGS <Number>(<Число>)
```

<Number> (<Число>) указывает на максимальное количество связанных обмоток в пределах компонента.

Обращайтесь к приведенному ниже примеру 10-22.

#VERBATIM (дословная передача)

Эта директива используется для передачи строки сценария из компонента непосредственно в файл Фортрана без обработки и изменений.

Директива #VERBATIM выглядит следующим образом:

```
#VERBATIM {<Text>}({<Текст>})
```

<Text> (<Текст>) может быть любой строкой текста, например комментарием, директивой компилятора или строкой кода.
<Text> (<Текст>) появиться в сгенерированном PSCAD файле компилятора Фортран в неизменном виде (то есть дословно).

```
! Caution should be exercised as ANY line of code will be written to
! the Fortran file, be it Fortran compatible or not!
!(Должна соблюдаться предосторожность, поскольку любая строка кода будет
! записана в файл компилятора Фортран, не зависимо от того, является ли
! она допустимой в Фортране или нет)

! PSCAD Script: (Сценарий PSCAD)
!
#VERBATIM {! This is a comment line.} {! Это строка комментария.}
#VERBATIM {  X = 1.0 ! This is a line of Fortran code.} (X = 1.0 ! Эта
строка кода в Фортране)
#VERBATIM {@#$%^&*!& This is a line of rubbish.} {@#$%^&*!& Это
```

```
бессмысленная строка.}

!

! Fortran File: (Файл компилятора Фортран)

!

! This is a comment line.(Это строка с комментарием)

  X = 1.0 ! This is a line of Fortran code.  (Это строка с кодом Фортрана)

@#$%^&*!& This is a line of rubbish. (Это бессмысленная строка)
```

~ Line Continuation Operator (Оператор разделения строки)

В некоторых случаях возникает необходимость разделить строку сценария на несколько строк. Ситуация возникает, если строка сценария очень длинная или если часть строки изменяется при определенных условиях.

Операторы разделения строк обычно используются в сценарии для организации упорядоченной структуры. Таким образом, PSCAD автоматически разобьет строки, длина которых превышает 72 знака, как правило, пользователь не должен беспокоиться о длине строк. Однако, для организации структурированного кода, оператор разделения строк должен использоваться в процессе проектирования.

Оператор разделения строки используется следующим образом: Строка с символом оператора в конце строки ~ будет объединена со следующей строкой, если она начинается с символа ~. Это преобразование происходит после определения

> **ПРИМЕЧАНИЕ**: Оператор разделения строки может быть использован в любом месте за исключением строк с директивными указаниями. Единственным исключением является директива #CASE, которая поддерживает использование оператора разделения строки.

ПРИМЕР 10-23:

В примере 10-12 директивы #IF, #ELSEIF, #ELSE, #ENDIF использовались для определения выходной формы сигнала в модели генератора сигналов. Сценарий из этого примера приведен ниже:

```
!

! Signal Generator (Генератор сигнала)

!

#IF Type == 1

      $OUT = SIN(TWO_PI*$F)

#ELSE

      $OUT = COS(TWO_PI*$F)

#ENDIF
```

```
!
```

В качестве простого принтера использования оператора разделения строки ~, приведенный выше сценарий может быть переписан следующим образом:

```
!
! Signal Generator
!
        $OUT = ~
#IF Type==1
~SIN~
#ELSE
~COS~
#ENDIF
~(TWO_PI*$F)
!
```

В случае если *Type = 2*, то при обработке директив IF, #ELSEIF, #ELSE, #ENDIF и оператора разделения строки, сценарий будет выглядеть следующим образом:

```
!
! Signal Generator(Генератор сигнала)
!
        $OUT = COS(TWO_PI*$F)
!
```

Expression Evaluation (Оценочные выражения)

Математические функции

Арифметические Операторы

Логические Операторы

Тройной Оператор

Все важные функции компилятора Фортран доступны для кодирования в разделах компилятора Фортран, DSDYN или DSOUT. Однако для таких разделов как Computations (Вычисления) также необходимы некоторые математические и логические функции: в PSCAD встроены некоторые внутренние математические функции и логические операторы для использования в сегментах, в которых не доступны функции компилятора Фортран.

Математические функции

Математические функции могут быть использованы для входных параметров, входных сигналов, а также для результатов расчета.

В таблице приведены списки всех доступных математических функций. Эти функции используются только в разделе Computations (Вычисления):

Функция	**Описание**
`CEIL(x)`	Округляет дробную часть до следующего большего целого числа
`FLOOR(x)`	Округляет дробную часть до следующего меньшего целого числа
`ROUND(x)`	Добавляет 0.5 к вещественному значению и затем представляет целочисленную функцию (то есть ROUND(X) = INT(X+0.5)) Не использовать для отрицательных чисел.
`TRUNC(x)`	Округляет x до нуля, при этом возвращает ближайшую целую часть, не превышающую амплитуду x.
`FRAC(x)`	Отбрасывает целую часть вещественного значения (левую часть десятичной дроби)
`REAL(x)`	Вещественная часть комплексного числа
`IMAG(x)`	Мнимая часть комплексного числа
`ABS(x)`	Абсолютное значение
`ARG(x)`	Возвращает фазный угол (или угловую компоненту) комплексного числа x, выраженный в радианах.
`NORM(x)`	Нормированное значение комплексного числа $(x^2 + y^2)$
`SIN(x)`	Функция синусоиды
`COS(x)`	Функция косинуса
`TAN(x)`	Функция тангенса
`ASIN(x)`	Функция арксинуса
`ACOS(x)`	Функция арккосинуса
`ATAN(x)`	Функция арктангенса
`SINH(x)`	Гиперболическая функция синуса
`COSH(x)`	Гиперболическая функция косинуса
`TANH(x)`	Гиперболическая функция тангенса
`SQRT(x)`	Корень квадратный

LOG(x)	Натуральный логарифм
LOG10(x)	Логарифм с основанием 10
EXP(x)	Экспонента
RAND(x)	Случайная величина между 0 и x
P2RX(m,θ)	Преобразование из полярных в прямоугольные (θ в градусах)
P2RY(m,θ)	Преобразование из полярных в прямоугольные (θ в градусах)
R2PM(x,y)	Преобразование из прямоугольных в полярные
R2PA(x,y)	Преобразование из прямоугольных в полярные
INT(x)	Отбрасывает дробную часть вещественного значения (правая часть десятичной дроби)
NINT(x)	Возвращает ближайшее целое к аргументу

Арифметические операторы

Ниже приведен список доступных в PSCAD арифметических операторов. Эти операторы используются для выполнения математических вычислений в разделах Computations (Вычисления), компилятор Фортран, DSDYN и DSOUT.

Функция	Описание
+	Добавить
–	Вычесть
*	Умножить
/	Поделить
%	Остаток от деления
**	Возведение в степень
\	Соединение в параллель (xy) / (x +y)

Логические Операторы

Ниже приведены логические операторы доступные в PSCAD. Логические операторы в основном используются совместно с директивами #IF, #ELSEIF, #ELSE, #ENDIF, а также с Тройным Оператором. Они также используются в разделе Checks (Проверки).

> **ПРИМЕЧАНИЕ:** Логические операторы возвращают значение *1,* если условие выполняется, значение *0,* если условие не выполняется.

Функция	Описание
==	Равенство

!=	Не равенство
!	Не
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
||	Или
&&	И

ПРИМЕР 10-24:

Логические операторы используются в различных разделах. Ниже приведены примеры, демонстрирующие возможности использования рассматриваемых операторов:

```
!
! ...with #IF, #ELSEIF, #ELSE, #ENDIF Directives (С директивами #IF,
!#ELSEIF, #ELSE, #ENDIF)
!
#IF F >= 60.0
      Fout = 60.0
#ENDIF
!
! ...with a Ternary Operator in the Computations
! segment
! (С тройным оператором в разделе Вычисления)
!
REAL L = X == 0.0 ? Y*2.0 : Y/3
!
! ...in the Checks segment (В разделе Проверки)
!
ERROR Value too small : R > 0.001
```

```
!
```

Тройной Оператор

Тройной Оператор – это краткая форма представления выражений типа IF-ELSE-ENDIF при разработке сценария. Он позволяет пользователю в одной строке определить переменную при выполнении определенных условий.

Тройной оператор используется в разделе Computations (Вычисления) и выглядит следующим образом:

```
<Logic> ? <Value_if_True> : <Value_if_False> (<Логическое выражение> ?
<Значение если условие выполняется> : <Значение если условия не выполняется>)
```

<Logic> (<Логическое выражения>) – Логическое выражения с логическими операторами. <Value_if_True> (<Значение если условие выполняется>) и <Value_if_False> (Значение если условие не выполняется) может быть константой или математическим выражением.

> **ПРИМЕЧАНИЕ**: Тройные операторы должны использоваться с осторожностью. Убедитесь, что переменные, используемые в выражениях с тройным оператором, будут доступны при различных условиях. Другими словами, несвязанная логика может сделать недоступной одну или несколько переменных, используемых в выражении с тройным оператором, что может привести к некорректному результату для выражения с тройным оператором. Входные переменные активизируются / деактивизируются через Условия Применения, Слои и Фильтры.

ПРИМЕР 10-25:

Пользователь планирует определить вещественную переменную X в разделе Computations (Вычисления) компонентного определения. Значение переменной X должно быть *1.0,* если входной параметр *N* равен *2* or *3*, а в других случаях определяться математическим выражением.

Нижеприведенный код должен появиться в разделе Computations(Вычисления):

```
REAL X = (N==2 || N==3) ? 1.0 : SQRT(2)*V
```

Где *V* – это заранее определенная константа.

ПРИМЕР 10-26:

Пользователь планирует определить вещественную переменную *Torq* в разделе Computations (Вычисления) компонентного Определения. Значение *Torq* определяется математическим выражением, в котором один из элементов зависит от определенных условий. Это может быть реализовано путем применения тройного оператора:

```
REAL Torq = (X > 1 ? 0.0 : Tm) + Te*100
```

Где *X*, *Tm* и *Te* - заранее определенные константы.