Delcam Advanced Manufacturing Solutions

# PowerMILL 2012

## Руководство по макропрограммированию

PowerMILL 2012

# Руководство пользователя

## Макропрограммирование


*Выпуск 4*

# Важные замечания

## PowerMILL



### Уведомление

Данный документ ссылается на ряд зарегистрированных торговых марок, являющихся собственностью их владельцев. Например, Microsoft и Windows являются зарегистрированными торговыми марками или торговыми марками Корпорации Microsoft в США.

### Патенты

Сглаживание траектории выборки находится под защитой патентов:

Патент: GB 2374562 Improvements Relating to Machine Tools

Патент: US 6,832,876 Machine Tools

Некоторые функции модулей ViewMill и Симуляции находятся под защитой патента:

Патент: GB 2 423 592 Surface Finish Prediction

### Лицензии

Разумный курсор запатентован, номера патентов U.S. 5,123,087 и 5,371,845 (Ashlar Inc.)

# Содержание

# Макросы

**Макрос** - это файл, содержащий последовательность команд для автоматизации часто выполняемых операций. Макросы можно создать, записав операции выполняемые в PowerMILL, или набрав команды в текстовом редакторе. Записанные макросы имеют расширение **.mac** и могут быть запущены из ветви **Макросы** в Проводнике PowerMILL.

Вы можете записать один или несколько макросов для выполнения своих задач. Вы можете вызвать макрос из другого макроса.

Существует два типа макросов:

- Макрос инициализации, который называется **pmuser.mac** и выполняется при запуске PowerMILL. По умолчанию, пустая копия этого макроса находится в папке C:\Program Files\Delcam\PowerMILL13.0.06\lib\macro. Перезаписывая или добавляя команды PowerMILL к нему, вы можете настроить ваши собственные параметры и настройки по умолчанию. Также вы можете поместить макрос **pmuser** в папку **pmill4** в базовом каталоге **Home**. Это позволит персонализировать настройки макросов для отдельных учетных записей.
- Макросы пользователя - это макросы, которые вы задаете для автоматизации различных операций.

*Помимо макроса инициализации, вы можете создать макросы для показа, скрытия, или изменения подводов и переходов, установки параметров NC Файлов, задание часто используемых последовательностей обработки и т.д.*

# Задание папки Home для макросов

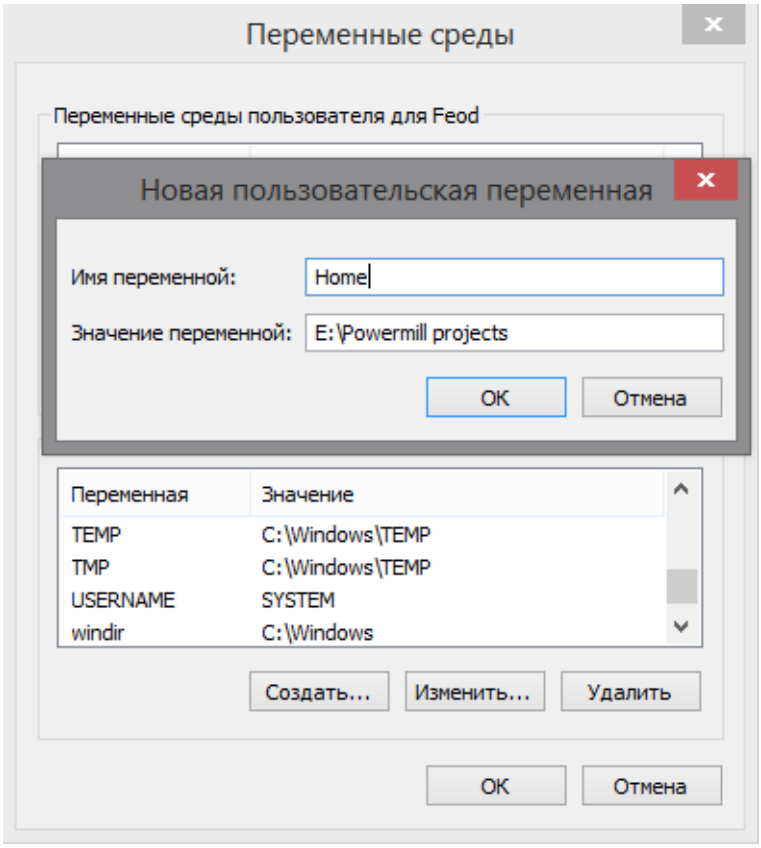
PowerMILL проверяет наличие значения, заданного переменной среды Windows **Home** для путей к папке макросов пользователя. Решите, куда должна направлять переменная среды **Home**, например, **E:\PowerMILL_Projects**, и задайте переменную среды Windows:

- Откройте **Панель управления** Windows и выберите опцию **Система > Дополнительные параметры системы**. Откроется диалог **Свойства системы**.
- Нажмите на вкладку **Дополнительно**.

- Выберите **Переменные среды**.
- Чтобы задать новое имя и значение новой переменной, нажмите **Создать**, чтобы открыть диалог **Новая пользовательская переменная**.

  В поле **Имя переменной** введите **Home**.

  В поле **Значение переменной** введите путь, где должна располагаться папка **Home** для PowerMILL. Например, **E:\PowerMILL_Projects**.




- Нажмите **OK** во всех открытых диалогах, чтобы сохранить изменения и закрыть их.
- Создайте в каталоге **Home** папку с именем **pmill4**. Например, **E:\PowerMILL_Projects\pmill4**.

Когда создаются или вызываются пользовательские макросы, PowerMILL автоматически обращается к этой папке.

# Создание макросов

Вы можете создавать макросы следующими способами:

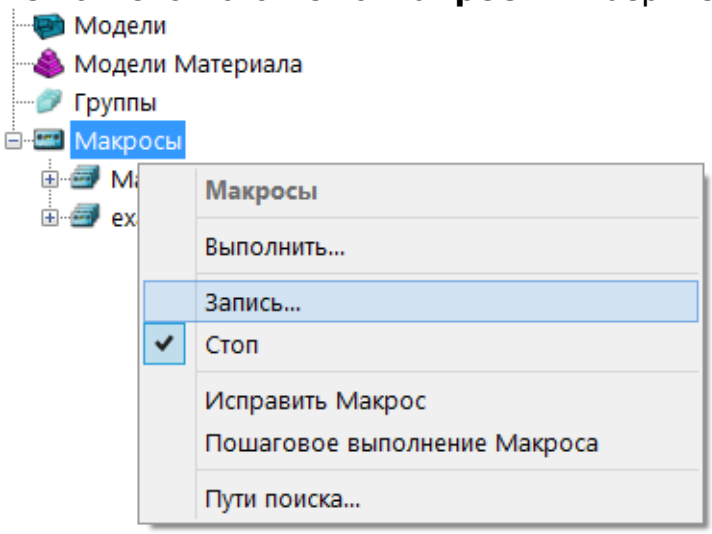
- Записывая последовательность команд, выполняемых в PowerMILL.
- Создавая собственные макросы в текстовом редакторе.

# Запись макросов в PowerMILL

Простейшим способом создания макроса является запись команд PowerMILL в ходе работы с программой. В макрос записываются только те значения, которые вы изменяете в диалогах. Поэтому, чтобы записать в макрос значения которые уже были заданы, нужно ввести заново эти значения, или повторно перевыбрать необходимые настройки. Например, если допуск чистовой обработки сейчас установлен на **0.1** мм, и вы хотите, чтобы макрос сохранил это значение, то нужно ввести **0.1** в поле **Допуск** во время записи макроса.

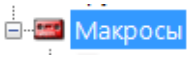
Чтобы записать макрос:

1 Из контекстного меню **Макросы** выберите **Запись**.




Это отобразит диалог **Выбор записываемого макроса**, который является стандартным диалогом Windows **Сохранить**.

2 Выберите подходящую папку, задайте **Имя файла** и нажмите кнопку **Сохранить**.

Значок макроса 

поменяет цвет на красный, показывая, что идет запись.

*Все параметры и настройки в диалогах, которые вам надо включить в макрос, должны быть выбраны/изменены во время записи макроса. Если какой-то параметр уже имеет нужное значение, введите его заново.*

3 Выполните набор команд, которые необходимо добавить в макрос.

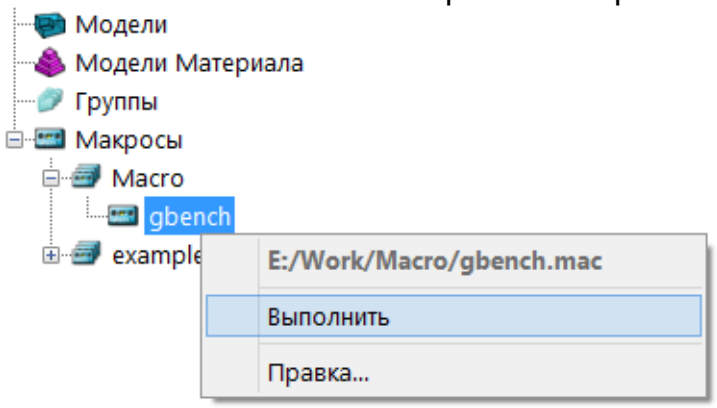
4 В контекстном меню **Макросы** выберите **Стоп**, чтобы остановить запись макроса.

# Запуск макросов

После запуска макроса будут выполнены все команды, содержащиеся в файле макроса.

1 Разверните **Макросы**, и выберите макрос, который хотите запустить.

2 Из контекстного меню макроса выберите **Выполнить**.

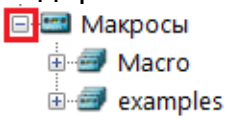



*Вы также можете запустить макрос выполнив двойной щелчок мыши на его имени в проводнике PowerMILL.*

## Выполнение только что записанного макроса

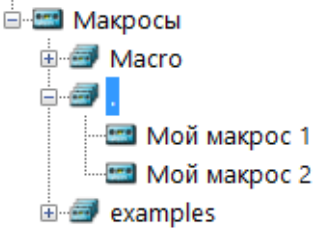
Расположение макроса, который вы только что записали, находится в локальной папке. Таким образом, только что записанный макрос становится доступным в локальном пути поиска макросов . Однако, список макросов динамически не обновляется. Чтобы выполнить принудительное обновление:

1 Нажмите рядом со строкой **Макросы**, чтобы свернуть содержимое.




2 Нажмите рядом со строкой **Макросы**, чтобы развернуть и обновить содержимое.

3 Нажмите рядом с , чтобы посмотреть макросы в этой папке, которая содержит только что созданный макрос.

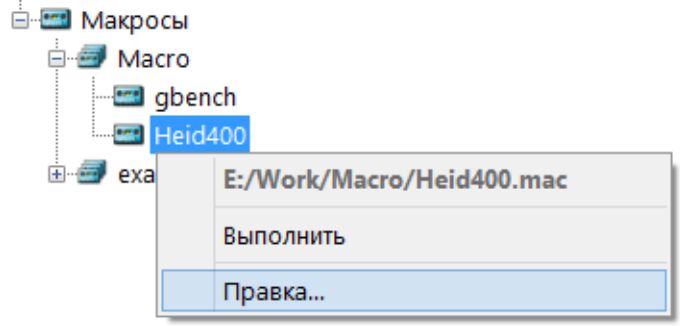


# Редактирование макросов

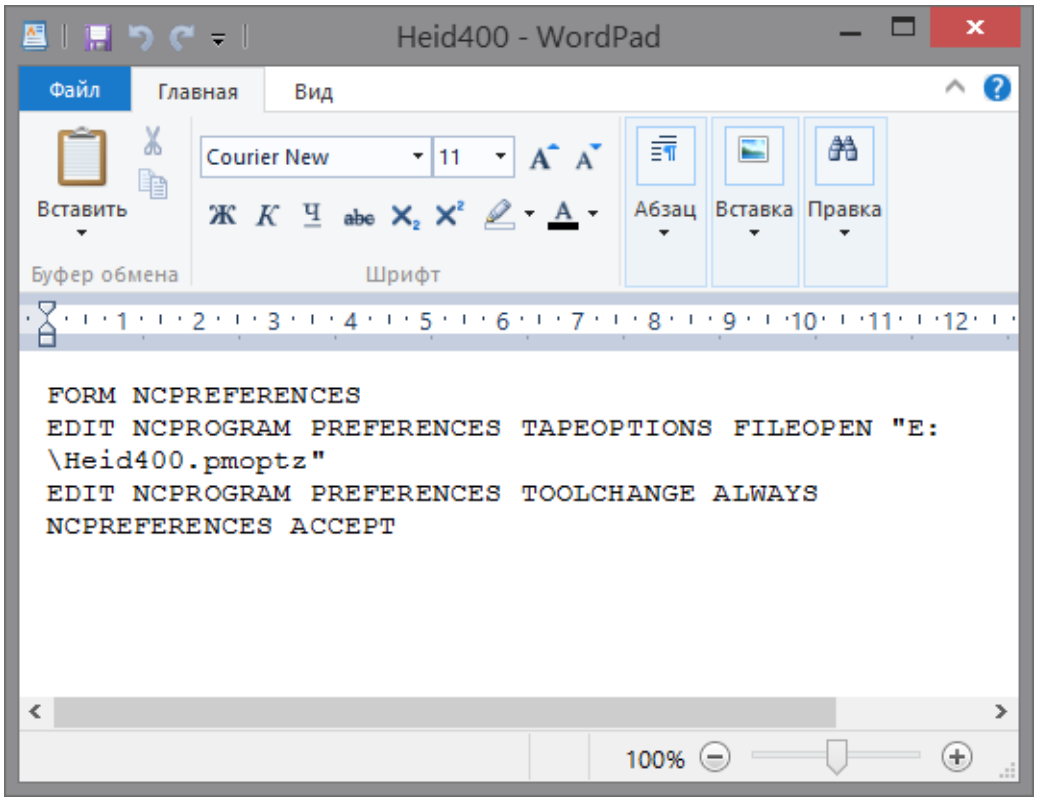
Вы можете редактировать записанные макросы, чтобы устранить какие-либо ошибки.

1 Разверните **Макросы** и выберите макрос, который хотите изменить.

2 В меню отдельного макроса выберите **Правка**.




Откроется стандартный документ WordPad.




*Если появляется сообщение об ошибке, информирующее, что макрос не может быть отредактирован, откройте проводник Windows и в меню Сервис выберите Свойства папки. Щелкните по вкладке Типы файлов и установите текстовый редактор **Wordpad** в качестве программы по умолчанию для файлов с расширением **.mac**.*

3 Отредактируйте команды макроса и сохраните файл.

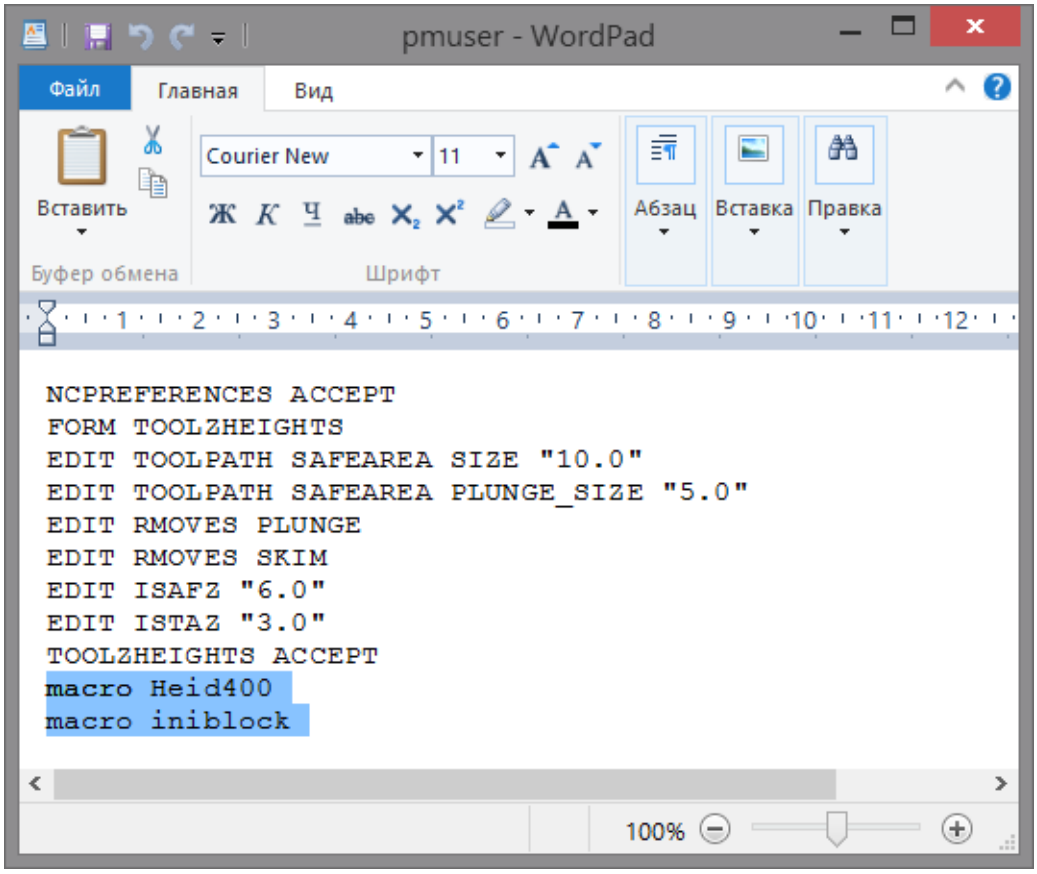
# Запуск макроса из другого макроса

Вы можете создавать небольшие макросы, которые выполняют одну операцию, а затем вызывать их из более крупного макроса. Далее приведен пример, показывающий, как добавить макрос **Heid400** и макрос **iniblock** в макрос **pmuser**.

1 В контекстном меню макроса **pmuser** выберите **Правка**.

2 Перейдите в нижнюю часть файла и добавьте следующие строки:

```
macro Heid400
macro iniblock
```




*Если в начале строки поставить две косые черты (//), то она будет восприниматься как комментарий и не выполняется.*

3 Сохраните и закройте **pmuser.mac**.

4 Выйдите из PowerMILL и запустите его снова, чтобы убедиться, что установки из макроса **pmuser** были активированы.

# Написание собственного макроса

Более мощным способом создания макросов является написание собственного макроса. Принципы написания описаны ниже в Руководстве по макросам.

Макросы позволяют:

- Создавать выражения.
- Использовать выражения, чтобы контролировать поток макросов.
- Использовать ряд реляционных и логических операторов.
- Оценивать выражения.
- Присваивать значения для переменных и параметров, используя операции присваивания.

Пункт меню **Помощь > Переменные > Справка > Functions** содержит список всех стандартных функций, которые вы можете использовать в макросах.

## Команды PowerMILL для макросов

Во время интерактивного использования PowerMILL, каждое нажатие пункта меню или ввод данных в диалогах, посылает команды в программу. Эти команды необходимо добавить в файл макроса, чтобы управлять PowerMILL-ом через макрос.

Следующий пример покажет как:

- Найти команды PowerMILL для добавления их в макрос.
- Разместить их в текстовом редакторе, например в WordPad.
- Отобразить макрос в Проводнике

Чтобы создать макрос:

1. На панели меню выберите **Вид > Панели инструментов > Командное окно**, чтобы открыть командное окно.
2. Выберите **Инструменты > Включить эхо** на панели меню, чтобы отображать в командном окне выполняемые команды.

```
PowerMILL >
Process Command : [FORM LEADLINK\r]

PowerMILL >
Process Command : [EDIT TOOLPATH LEADS SKIMDIST "2"\r]

PowerMILL >
Process Command : [EDIT TOOLPATH LEADS PLUNGEDIST "1"\r]
```

3 Чтобы посмотреть команды, необходимые для вычисления заготовки:

a Нажмите кнопку **Заготовка** на **Главной панели инструментов**.

b Когда откроется диалог **Заготовка**, нажмите **Вычислить**, и затем нажмите **Принять**.

Командное окно отобразит используемые команды:

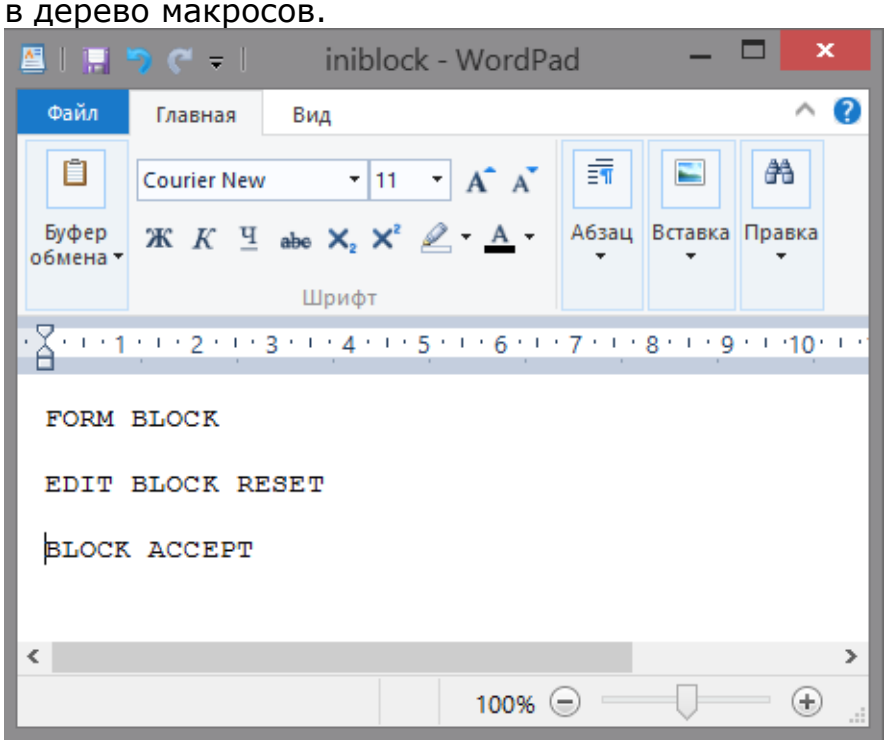
```
PowerMILL >
Process Command : [FORM BLOCK\r]

PowerMILL >
Process Command : [EDIT BLOCK RESET\r]

PowerMILL >
Process Command : [BLOCK ACCEPT\r]

PowerMILL >
```

Команды отображаются в квадратных скобках. Символы **\r** следует игнорировать. Необходимые команды это: **FORM BLOCK**, **EDIT BLOCK RESET** и **BLOCK ACCEPT**.

4 Откройте WordPad, и введите в нём эти команды.

*Команды не чувствительны к регистру. То есть FORM BLOCK тоже самое что и Form Block и тоже самое что и foRm bLock.*

5 Сохраните этот файл как iniblock.mac. Макрос добавится в дерево макросов.




*Дополнительную информацию смотрите в разделе **Запуск макросов**.*

# Добавление комментариев к макросу

Хорошим правилом будет добавление комментариев в файл макроса, чтобы объяснить что он будет делать. Комментарий – это текстовая строка, которая не влияет на работу макроса, но может помочь другим понять макрос, при его исследовании. Комментарии начинаются с //. Например:

```
// Этот макрос импортирует модель, создаёт заготовку
// и шаровую фрезу.
```

Также хорошим правилом будет добавление комментариев, объясняющих, что будет делать каждый раздел макроса. Это может быть очевидным при написании макроса, но впоследствии могут возникнуть трудности с пониманием. Хорошим правилом будет и добавление комментариев, описывающих команды, перед самой командой:

```
// Очистить все черновые границы
MACRO Clean 'boundary\Roughing'
```

Ещё одним способом использования комментариев является временное отключение команд в макросе. При отладке или написании макроса, желательно записывать только один шаг за один раз и перезапускать макрос после каждого изменения. Если макрос содержит длительные расчёты, или пересоздание траекторий, вам возможно захочется временно закомментировать предыдущие части макроса, пока будут проверяться следующие части. Например:

```
// Импортировать шаблон траектории
// IMPORT TEMPLATE ENTITY TOOLPATH "Drilling/Drilling.ptf"
```

# Руководство пользователя по макросам

Следующий пример покажет, как использовать язык макропрограммирования PowerMILL для создания макроса, который выведет слова из песни-считалки "Ten Green Bottles".

*10 green bottles sitting on the wall*
*10 green bottles sitting on the wall*
*And if 1 green bottle should accidentally fall*
*There will be 9 green bottles sitting on the wall*

*9 green bottles sitting on the wall*
*9 green bottles sitting on the wall*
*And if 1 green bottle should accidentally fall*
*There will be 8 green bottles sitting on the wall*

*И так далее до последнего четверостишья:*

*1 green bottle sitting on the wall*
*1 green bottle sitting on the wall*
*And if 1 green bottle should accidentally fall*
*There will be 0 green bottles sitting on the wall.*

Основные шаги:

1. Создание основного макроса.
2. Добавление переменных.
3. Добавление циклов.
4. Выполнение макроса с аргументами.
5. Выбор, производимый в макросе.
6. Более подробно о функциях в макросе.
7. Использование оператора SWITCH.
8. Вывод значений из макроса.
9. Использование цикла FOREACH в макросе.
10. Использование массивов в цикле FOREACH.

## Создание основного макроса

Этот шаг покажет как создать и запустить основной макрос, используя язык программирования PowerMILL.

1 В текстовом редакторе, таком как WordPad, введите:

```
PRINT "10 green bottles sitting on the wall"
PRINT "10 green bottles sitting on the wall"
PRINT "And if 1 green bottle should accidentally fall"
PRINT "There will be 9 green bottles sitting on the wall"
```

2 Сохраните файл как **example.mac**.

3 На панели меню выберите **Вид > Панели инструментов > Командное окно**, чтобы открыть командное окно. Убедитесь, что галочка **Инструменты > Включить эхо** убрана.

4 Из контекстного меню **Макросы** выберите **Выполнить**. Появится диалог **Выберите запускаемый Макрос**.

5 Перейдите в нужную папку, выберите **example.mac**, и нажмите **Открыть**. Макрос запустится, и в командном окне отобразится текст, заключённый в двойные кавычки (") в макросе.

```
PowerMILL > Выполняется макрос:- E:/Work/Macro/example.mac
10 green bottles sitting on the wall
10 green bottles sitting on the wall
And if 1 green bottle should accidentally fall
There will be 9 green bottles sitting on the wall
PowerMILL >
```

## Добавление переменных

Первые две строки в макросе **example.mac** одинаковые. Чтобы минимизировать повторения (и для более лёгкого управления) хорошим способом будет записать строку один раз, а затем повторно вызывать её когда она понадобится. Чтобы сделать это, необходимо создать локальную переменную, содержащую текстовую строку.

Можно создавать различные типы переменных в PowerMILL. Для сохранения строки, содержащей текст, необходимо использовать переменную **STRING**.

1 Откройте файл **example.mac** в текстовом редакторе и измените его на:

```
// Создание переменной, содержащей текст первой строки
STRING bottles = "10 green bottles sitting on the wall"
PRINT $bottles
PRINT $bottles
PRINT "And if 1 green bottle should accidentally fall"
PRINT "There will be 9 green bottles sitting on the wall"
```

*Первая строка – это комментарий, который объясняет значение второй строки.*

2 Сохраните файл как **example.mac**.

3 В PowerMILL нажмите **Выполнить** на этом макросе. Командное окно отобразит то же, что и раньше:

```
PowerMILL > Выполняется макрос:- E:/Work/Macro/example.mac
10 green bottles sitting on the wall
10 green bottles sitting on the wall
And if 1 green bottle should accidentally fall
There will be 9 green bottles sitting on the wall
PowerMILL >
```

Следует учитывать следующие особенности при работе с переменными:

- Вы должны задать все локальные переменные до того, как они будут использоваться, - в нашем случае

  ```
  STRING bottles = "10 green bottles sitting on the wall"
  ```

  задаёт локальную переменную **bottles**.
- Переменная **bottles** является локальной переменной, поэтому она будет использоваться только в макросе, где она задана. Она не является переменной PowerMILL. Если ввести её в командном окне – мы получим ошибку.

  ```
  PowerMILL > PRINT $bottles

  Process Command : [PRINT $bottles\r]

  #ОШИБКА: Неправильное имя
  PowerMILL >
  ```
- После того, как локальная переменная задана, её можно использовать неограниченное число раз в макросе.
- Можно задать неограниченное число локальных переменных в макросе.

## Добавление циклов

В нашем макросе имеются две строки с одинаковым содержимым: `PRINT $bottles`. В данном случае это допустимо, потому что эти строки спользуются только дважды, но если вы захотите повторить их 5 или 20 раз, будет лучше использовать цикл. PowerMILL имеет три оператора циклов:

- WHILE
- DO - WHILE
- FOREACH

Следующий пример использует оператор WHILE для повторения команды 5 раз.

1 Откройте макрос **example.mac** в текстовом редакторе и измените его на:

```
// Создание переменной, содержащей текст первой строки.
STRING bottles = "10 green bottles sitting on the wall"

// Создание переменной для хранения количества раз,
// которые нужно напечатать первую строку.
// В данном случае, 5.
INT Count = 5

// Повторение цикла до тех пор, пока Count больше 0
WHILE Count > 0 {
    // Вывод строки
    PRINT $bottles
    // Уменьшение счётчика на 1
    $Count = Count - 1
}

// Вывод последних двух строк
PRINT "And if 1 green bottle should accidentally fall"
PRINT "There will be 9 green bottles sitting on the wall"
```

*$Count = Count – 1* - это оператор присваивания, согласно которому переменная *$Count* слева от = должна быть с префиксом *$*.

*В пустых строках нет необходимости, но они делают чтение макроса более удобным.*

2 Сохраните файл как **example.mac**.

3 В PowerMILL нажмите **Выполнить** на этом макросе. Командное окно отобразит:

```
PowerMILL > Выполняется макрос:- E:/Work/Macro/example.mac
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
And if 1 green bottle should accidentally fall
There will be 9 green bottles sitting on the wall
PowerMILL >
```

*Изменение INT Count = 5 на INT Count = 10 выведет строку **10 green bottles sitting on the wall** десять раз, вместо пяти.*

## Выполнение макроса с аргументами

Добавленный цикл в макрос **example.mac** работает хорошо, если необходимо всегда выводить строку **10 green bottles sitting on the wall** одинаковое количество раз.
Однако, если необходимо изменить количество повторов во время выполнения, вместо того чтобы редактировать каждый раз макрос, предпочтительнее будет записать макрос таким образом, что ему можно будет задавать количество повторов. Чтобы сделать это, необходимо создать функцию **Main**.

1. Откройте макрос **example.mac** в текстовом редакторе и измените его на:

```
// Создание функции Main, содержащей количество раз,
// необходимое для вывода первой строки.
FUNCTION Main (INT Count) {

   // Создание переменной с текстом первой строки
   STRING bottles = "10 green bottles sitting on the wall"

   // Повторение цикла до тех пор, пока Count больше 0
   WHILE Count > 0 {
      // Вывод строки
      PRINT $bottles
      // Уменьшение счётчика на 1
      $Count = Count - 1
   }

// Вывод последних двух строк
   PRINT "If 1 green bottle should accidentally fall"
   PRINT "There will be 9 green bottles sitting on the wall"
}
```

2. Сохраните файл как **example.mac**.
3. Чтобы запустить этот макрос, нельзя будет выбрать **Выполнить** из контекстного меню **Макросы**, потому что необходимо задать значение для **Count**. Поэтому, наберите в командном окне:

   **MACRO example.mac 5**

   Где **5** - это значение для **Count**. Командное окно отобразит:

```
PowerMILL > MACRO  example.mac  5
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
If 1 green bottle should accidentally fall
There will be 9 green bottles sitting on the wall
PowerMILL >
```

*Если появляется предупреждение, что макрос не найден, проверьте, заданы ли пути поиска макросов.*

## Добавление пользовательских функций

Также как и функцию **Main** , можно создавать пользовательские функции. Их можно использовать для выделения блоков кода. Функции можно использовать:

- для создания библиотеки полезных операций
- для того, чтобы макрос был более понятным.

*Функцию можно вызывать неограниченное число раз в макросе.*

Следующий пример разделяет вывод первой строки в пользовательской функции таким образом, что функция **Main** становится более понятной.

1 Откройте макрос **example.mac** в текстовом редакторе и измените его на:

```
FUNCTION PrintBottles(INT Count) {

   // Создание переменной с текстом первой строки
   STRING bottles = "10 green bottles sitting on the wall"

   // Повторение цикла до тех пор, пока Count больше 0
   WHILE Count > 0 {
      // Вывод строки
      PRINT $bottles
      // Уменьшение счётчика на 1
      $Count = Count - 1
   }
}

FUNCTION Main (INT Count) {

   // Вывод первой строки число раз, заданное
   // с помощью переменной Count
   CALL PrintBottles(Count)

   // Вывод последних двух строк
   PRINT "If 1 green bottle should accidentally fall"
   PRINT "There will be 9 green bottles sitting on the wall"
}
```

2 Сохраните макрос.

3 Запустите макрос, набрав **MACRO example.mac 5** в командном окне.

```
PowerMILL > MACRO  example.mac  5
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
10 green bottles sitting on the wall
If 1 green bottle should accidentally fall
There will be 9 green bottles sitting on the wall
PowerMILL >
```

Результат вывода будет такой же как и раньше.

*Порядок функций в макросе не имеет значения. Например, совершенно неважно будет ли функция **Main** до или после функции **PrintBottles**.*

*Очень важно, чтобы каждая функция имела уникальное имя и чтобы макрос имел функцию с именем **Main**.*

*Макрос может содержать любое количество функций.*

## Выбор, производимый в макросе

Запуск макроса **example.mac** подразумевает, что вы вводите положительное значение.Однако, если необходимо чтобы строка **10 green bottles sitting on the wall** всегда выводилась по меньшей мере один раз, используйте для этого:

- Цикл **DO - WHILE**, поскольку он выполняет все команды перед проверкой условного выражения.
- Оператор **IF**.

## Цикл DO – WHILE

1 Отредактируйте функцию **PrintBottles** в макросе **example.mac** чтобы получилось следующее:

```
FUNCTION PrintBottles(INT Count) {

  // Создание переменной с текстом первой строки
  STRING bottles = "10 green bottles sitting on the wall"

  // Повторение цикла до тех пор, пока Count больше 0
  DO {
    // Вывод строки
    PRINT $bottles
    // Уменьшение счётчика на 1
    $Count = Count - 1
  } WHILE Count > 0
}
```

Функция Main не меняется:

```
FUNCTION Main (INT Count) {

  // Вывод первой строки число раз, заданное
  // с помощью переменной Count
  CALL PrintBottles(Count)

  // Вывод последних двух строк
  PRINT "And if 1 green bottle should accidentally fall"
  PRINT "There will be 9 green bottles sitting on the wall"
}
```

2 Наберите **MACRO example.mac 0** в командном окне.

```
PowerMILL > MACRO  example.mac  0
10 green bottles sitting on the wall
And if 1 green bottle should accidentally fall
There will be 9 green bottles sitting on the wall
PowerMILL >
```

Строка **10 green bottles sitting on the wall** будет выведена только один раз.

## Оператор IF

Оператор **IF** можно использовать чтобы обеспечить вывод строки **10 green bottles sitting on the wall** по меньшей мере дважды.

1 Отредактируйте функцию **Main** в макросе **example.mac** как указано ниже:

```
FUNCTION Main (INT Count) {

   // Обеспечим, чтобы Count был не менее 2
   IF Count < 2 {
      $Count = 2
   }

   // Вывод первой строки число раз, заданное
   // с помощью переменной Count
   CALL PrintBottles(Count)

   // Вывод последних двух строк
   PRINT "And if 1 green bottle should accidentally fall"
   PRINT "There will be 9 green bottles sitting on the wall"
}
```

Функция **PrintBottles** осталась неизменённой.

```
FUNCTION PrintBottles(INT Count) {

   // Создание переменной с текстом первой строки
   STRING bottles = "10 green bottles sitting on the wall"

   // Повторение цикла до тех пор, пока Count больше 0
   WHILE Count > 0 {
     // Вывод строки
     PRINT $bottles
     // Уменьшение счётчика на 1
     $Count = Count - 1
   }
}
```

2 Наберите **MACRO example.mac 0** в командном окне.

```
PowerMILL > MACRO  example.mac  0
10 green bottles sitting on the wall
10 green bottles sitting on the wall
And if 1 green bottle should accidentally fall
There will be 9 green bottles sitting on the wall
PowerMILL >
```

Строка **10 green bottles sitting on the wall** будет выведена дважды.

## Более подробно о функциях в макросе

На данный момент наш макрос выводил только первое четверостишье из песни-считалки "Ten Green Bottles". Чтобы заставить макрос вывести все четверостишья, необходимо изменить функцию **PrintBottles** таким образом, что она будет считывать значения из двух параметров:

- **Count** – для количества раз вывода строки "X green bottles" .
- **Number -** для количества бутылок.

1 Отредактируйте функцию **PrintBottles** в макросе **example.mac** как указано ниже:

```
FUNCTION PrintBottles(INT Count, INT Number) {
  // Создание переменной с текстом первой строки
  STRING bottles = String(Number) + " green bottles
  sitting on the wall"

  // Повторение цикла до тех пор, пока Count больше 0
   WHILE Count > 0 {
    // Вывод строки
    PRINT $bottles
    // Уменьшение счётчика на 1
    $Count = Count - 1
  }
}
```

Это добавит второй параметр к функции **PrintBottles**, которая будет использовать этот параметр для преобразования **Number** в строковое значение **STRING (Number)**. Которое затем будет соединено (**+**) со строкой **green bottles sitting on the wall** для вывода переменной **bottles**.

2 Отредактируйте функцию **Main** в макросе **example.mac** как указано ниже:

```
FUNCTION Main (INT Count) {
   // Обеспечим, чтобы Count был не менее 2
   IF Count < 2 {
      $Count = 2
   }

   // Начнём с 10 бутылок
   INT Bottles = 10

   WHILE Bottles > 0 {
      // Вывод первой строки число раз, заданное
      // с помощью переменной Count
      CALL PrintBottles(Count, Bottles)
      // Считаем Bottles в обратном порядке
      $Bottles = $Bottles - 1
      // Построение номера строки 'bottles_left'
      STRING bottles_left = "There will be " +
      string(Bottles) + " green bottles sitting on the wall"
      // Вывод последних двух строк
      PRINT "If 1 green bottle should accidentally fall"
      PRINT $bottles_left
   }
}
```

3 Наберите **MACRO example.mac 2** в командном окне.

```
PowerMILL > MACRO example.mac 2
10 green bottles sitting on the wall
10 green bottles sitting on the wall
If 1 green bottle should accidentally fall
There will be 9 green bottles sitting on the wall
9 green bottles sitting on the wall
9 green bottles sitting on the wall
If 1 green bottle should accidentally fall
There will be 8 green bottles sitting on the wall
.....
.....
If 1 green bottle should accidentally fall
There will be 0 green bottles sitting on the wall
PowerMILL >
```

*В функции **Main**, когда вызывается функция **PrintBottles** , вы передаёте ей два параметра **Count** и **Bottles** , в то время как в функции **PrintBottles** параметр **Bottles** определяется как **Number**. Параметры, передаваемые в функцию, не должны иметь такого же имени как и функция, которая их вызывает.*

*Порядок вызова параметров имеет важное значение.*

*Любые изменения, внесенные в значение параметра внутри функции, не изменяют значение параметра в вызывающей функции, если только параметр не определяется значением OUTPUT.*

## Использование оператора SWITCH

До сих пор для вывода количества бутылок мы использовали цифры, но будет лучше использовать слова. Таким образом, вместо **10 green bottles** будет выводиться **Ten green bottles.**

Одним из способов как это сделать, является использование большой цепи оператора **IF - ELSEIF**, чтобы выбрать текстовое представление числа. Другим способом является использование оператора **SWITCH**.

```
SWITCH Number {
  CASE 10
    $Text = "Ten"
    BREAK
  CASE 9
    $Text = "Nine"
    BREAK
  CASE 8
    $Text = "Eight"
    BREAK
  CASE 7
    $Text = "Seven"
    BREAK
  CASE 6
    $Text = "Six"
    BREAK
  CASE 5
    $Text = "Five"
    BREAK
  CASE 4
    $Text = "Four"
    BREAK
  CASE 3
    $Text = "Three"
    BREAK
  CASE 2
    $Text = "Two"
    BREAK
  CASE 1
    $Text = "One"
    BREAK
  DEFAULT
    $Text = "No"
    BREAK
}
```

Оператор **SWITCH** сопоставляет значение его аргумента (в данном случае это **Number**) с соответствующим значением **CASE** и выполняет все последующие строки пока не встретися оператор **BREAK**. Если совпадения значений не найдено, будет выбран блок **DEFAULT** (в данном случае **No**).

***DEFAULT** - это опциональный шаг.*

## Вывод значений из макроса

Следующий пример покажет как создать переменную **OUTPUT** из оператора **SWITCH**.

1. Создайте новую функцию с именем **NumberStr** содержащую оператор **SWITCH** как на предыдущей странице. Первой строкой будет:

   ```
   FUNCTION NumberStr(INT Number, OUTPUT STRING Text) {
   ```

   и последней строкой:

   ```
   }
   ```

2. Отредактируйте функцию **PrintBottles** в макросе **example.mac** чтобы получилось следующее:

   ```
   FUNCTION PrintBottles(INT Count INT Number) {

      // Конвертация переменной Number в текстовую строку
      STRING TextNumber = ''
      CALL NumberStr(Number,TextNumber)

      // Создание переменной с текстом первой строки
      STRING bottles = TextNumber + " green bottles sitting
      on the wall"

      // Повторение цикла до тех пор, пока Count больше 0
      WHILE Count > 0 {
         // Вывод строки
         PRINT $bottles
         // Уменьшение счётчика на 1
         $Count = Count - 1
      }
   }
   ```

Это добавит переменную **OUTPUT** в функцию **PrintBottles**.

3 Отредактируйте функцию **Main** в макросе **example.mac** чтобы получилось следующее:

```
FUNCTION Main (INT Count) {

   // Обеспечим, чтобы Count был не менее 2
    IF Count < 2 {
      $Count = 2
   }

   // Начнём с 10 бутылок
   INT Bottles = 10

   WHILE Bottles > 0 {
      // Вывод первой строки число раз, заданное
      // с помощью переменной Count
      CALL PrintBottles(Count, Bottles)
      // Считаем Bottles в обратном порядке
      $Bottles = $Bottles - 1

     // Конвертация переменной Bottles в текстовую строку
      STRING BottlesNumber = ''
      CALL NumberStr(Bottles, BottlesNumber)

      // Построение номера строки 'bottles_left'
      STRING bottles_left = "There will be " +
      lcase(BottlesNumber) + " green bottles sitting on
      the wall"
      // Вывод последних двух строк
      PRINT "If one green bottle should accidentally fall"
      PRINT $bottles_left
   }
}
```

Переменная **BottlesNumber** заявлена в цикле **WHILE** функции **MAIN** .

*Каждый блок кода или функция могут задавать свои собственные наборы локальных переменных. Область действия переменной начинается с момента её заявления и заканчивается в конце содержащего её блока или функции.*

4 Полный текст макроса с функцией **NumberStr** будет таким:

```
FUNCTION PrintBottles(INT Count, INT Number) {

  // Конвертация переменной Number в текстовую строку
  STRING TextNumber = ''
  CALL NumberStr(Number,TextNumber)

  // Создание переменной, содержащей текст первой строки
  STRING bottles = TextNumber + " green bottles sitting on
  the wall"

  // Повторение цикла до тех пор, пока Count больше 0
   WHILE Count > 0 {
    // Вывод строки
    PRINT $bottles
    // Уменьшение счётчика на 1
    $Count = Count - 1
  }
}

FUNCTION Main (INT Count) {

  // Обеспечим, чтобы Count был не менее 2
  IF Count < 2 {
  $Count = 2
  }

  // Начинаем с 10 бутылок
  INT Bottles = 10

  WHILE Bottles > 0 {
    // Вывод первой строки число раз, заданное
    // с помощью переменной Count
    CALL PrintBottles(Count, Bottles)
    // Считаем Bottles в обратном порядке
    $Bottles = $Bottles - 1

    // Конвертация переменной Bottles в текстовую строку
    STRING BottlesNumber = ''
    CALL NumberStr(Bottles, BottlesNumber)

    // Построение номера строки 'bottles_left'
    STRING bottles_left = "There will be " +
    lcase(BottlesNumber) + " green bottles sitting
    on the wall"
    // Вывод последних двух строк
    PRINT "If one green bottle should accidentally fall"
    PRINT $bottles_left
  }
}
```

```
FUNCTION NumberStr(INT Number, OUTPUT STRING Text) {
SWITCH Number {
    CASE 10
      $Text = "Ten"
      BREAK
    CASE 9
      $Text = "Nine"
      BREAK
    CASE 8
      $Text = "Eight"
      BREAK
    CASE 7
      $Text = "Seven"
      BREAK
    CASE 6
      $Text = "Six"
      BREAK
    CASE 5
      $Text = "Five"
      BREAK
    CASE 4
      $Text = "Four"
      BREAK
    CASE 3
      $Text = "Three"
      BREAK
    CASE 2
      $Text = "Two"
      BREAK
    CASE 1
      $Text = "One"
      BREAK
    DEFAULT
      $Text = "No"
      BREAK
  }
}
```

Для запуска макроса наберите **MACRO example.mac 2**.

```
PowerMILL >
PowerMILL > MACRO example.mac 2
Ten green bottles sitting on the wall
Ten green bottles sitting on the wall
If one green bottle should accidentally fall
There will be nine green bottles sitting on the wall
Nine green bottles sitting on the wall
Nine green bottles sitting on the wall
If one green bottle should accidentally fall
There will be eight green bottles sitting on the wall
.....
.....
If one green bottle should accidentally fall
There will be no green bottles sitting on the wall
PowerMILL >
```

## Использование цикла FOREACH

Следующий пример покажет как использовать цикл **FOREACH** для управления числом бутылок вместо цикла **WHILE**.

1 Отредактируйте функцию **Main** в макросе **example.mac** чтобы получилось следующее:

```
FUNCTION Main (INT Count) {
     // Обеспечим, чтобы Count был не менее 2
     IF Count < 2 {
     $Count = 2
     }

     FOREACH Bottles IN {10,9,8,7,6,5,4,3,2,1} {
        // Вывод первой строки число раз, заданное
        // с помощью переменной Count
        CALL PrintBottles(Count, Bottles)
        // Считаем Bottles в обратном порядке
        $Bottles = $Bottles - 1

       // Конвертация переменной Bottles в текстовую строку
        STRING BottlesNumber = ''
        CALL NumberStr(Bottles, BottlesNumber)

        // Построение номера строки 'bottles_left'
        STRING bottles_left = "There will be " +
        lcase(BottlesNumber) + " green bottles sitting on
        the wall"
        // Вывод последних двух строк
        PRINT "If one green bottle should accidentally fall"
        PRINT $bottles_left
     }
  }
```

Остальная часть макроса **example.mac** не меняется.

Полный текст макроса будет таким:

```
FUNCTION PrintBottles(INT Count, INT Number) {

  // Конвертация переменной Number в текстовую строку
  STRING TextNumber = ''
  CALL NumberStr(Number,TextNumber)

  // Создание переменной, содержащей текст первой строки
  STRING bottles = TextNumber + " green bottles sitting on
  the wall"

  // Повторение цикла до тех пор, пока Count больше 0
    WHILE Count > 0 {
    // Вывод строки
    PRINT $bottles
    // Уменьшение счётчика на 1
    $Count = Count - 1
  }
}

FUNCTION Main (INT Count) {

  // Обеспечим, чтобы Count был не менее 2
  IF Count < 2 {
  $Count = 2
  }

  FOREACH Bottles IN {10,9,8,7,6,5,4,3,2,1} {
    // Вывод первой строки число раз, заданное
    // с помощью переменной Count
    CALL PrintBottles(Count, Bottles)
    // Считаем Bottles в обратном порядке
    $Bottles = $Bottles - 1

    // Конвертация переменной Bottles в текстовую строку
    STRING BottlesNumber = ''
    CALL NumberStr(Bottles, BottlesNumber)

    // Построение номера строки 'bottles_left'
    STRING bottles_left = "There will be " +
    lcase(BottlesNumber) + " green bottles sitting
    on the wall"

    // Вывод последних двух строк
    PRINT "If one green bottle should accidentally fall"
    PRINT $bottles_left
  }
}
```

```
  FUNCTION NumberStr(INT Number, OUTPUT STRING Text) {
  SWITCH Number {
    CASE 10
      $Text = "Ten"
      BREAK
    CASE 9
      $Text = "Nine"
      BREAK
    CASE 8
      $Text = "Eight"
      BREAK
    CASE 7
      $Text = "Seven"
      BREAK
    CASE 6
      $Text = "Six"
      BREAK
    CASE 5
      $Text = "Five"
      BREAK
    CASE 4
      $Text = "Four"
      BREAK
    CASE 3
      $Text = "Three"
      BREAK
    CASE 2
      $Text = "Two"
      BREAK
    CASE 1
      $Text = "One"
      BREAK
    DEFAULT
      $Text = "No"
      BREAK
  }
}
```

*Нет необходимости декларировать тип или начальное значение переменной **Bottles**, так как цикл **FOREACH** оперирует этими значениями.*

Чтобы запустить макрос наберите **MACRO example.mac 2** в командном окне.

```
PowerMILL >
PowerMILL > MACRO example.mac 2
Ten green bottles sitting on the wall
Ten green bottles sitting on the wall
If one green bottle should accidentally fall
There will be nine green bottles sitting on the wall
Nine green bottles sitting on the wall
Nine green bottles sitting on the wall
If one green bottle should accidentally fall
There will be eight green bottles sitting on the wall
.....
.....
If one green bottle should accidentally fall
There will be no green bottles sitting on the wall
PowerMILL >
```

Это даст точно такой же вывод как и в предыдущем примере. Этот пример показывает альтернативный способ создания одинакового вывода.

## Использование массивов в цикле FOREACH

Следующий пример покажет как использовать массив в цикле **FOREACH**, вместо использования списка, для управления числом бутылок.

1 Отредактируйте функцию **Main** в макросе **example.mac** чтобы получилось следующее:

```
FUNCTION Main (INT Count) {

  // Обеспечим, чтобы Count был не менее 2
    IF Count < 2 {
    $Count = 2
  }

  // Задаём массив чисел бутылок
  INT ARRAY BottleArray[10] = {10,9,8,7,6,5,4,3,2,1}

  FOREACH Bottles IN BottleArray {
     // Вывод первой строки число раз, заданное
     // с помощью переменной Count
     CALL PrintBottles(Count, Bottles)
     // Считаем Bottles в обратном порядке
     $Bottles = $Bottles - 1
```

```
        // Конвертация переменной Bottles в текстовую строку
        STRING BottlesNumber = ''
        CALL NumberStr(Bottles, BottlesNumber)
        // Построение номера строки 'bottles_left'
        STRING bottles_left = "There will be " +
        lcase(BottlesNumber) + " green bottles
        sitting on the wall"
        // Вывод последних двух строк
        PRINT "If one green bottle should accidentally fall"
        PRINT $bottles_left
    }
}
```

Остальная часть макроса **example.mac** не меняется.

2 Наберите **MACRO example.mac 2** в командном окне.

```
PowerMILL >
PowerMILL > MACRO example.mac 2
Ten green bottles sitting on the wall
Ten green bottles sitting on the wall
If one green bottle should accidentally fall
There will be nine green bottles sitting on the wall
Nine green bottles sitting on the wall
Nine green bottles sitting on the wall
If one green bottle should accidentally fall
There will be eight green bottles sitting on the wall
.....
.....
If one green bottle should accidentally fall
There will be no green bottles sitting on the wall
PowerMILL >
```

Это даст точно такой же вывод как и в предыдущем примере. Этот пример показывает альтернативный способ создания одинакового вывода.

# Переменные в макросах

Вы можете создавать переменные в макросе так же, как и в проекте PowerMILL. При создании переменной в макросе, она имеет те же свойства что и параметр PowerMILL, и может содержать или значение или выражение.

*Существуют некоторые ограничения при использовании переменных в макросе.*

- Имена переменных должны начинаться с алфавитного символа (a-z, A-Z) и могут содержать любое количество буквенно-цифровых последовательностей (a-z, A-Z, 1-9, _). Например, можно назвать переменную как **Count1**, но нельзя как **1Count**.
- Имена переменных нечувствительны к регистру. Например, все имена **Count**, **count**, и **CoUnT** относятся к одной переменной.
- Все переменные должны иметь один из следующих типов:

  **INT** — Целые числа. Например, 1, 21, 5008.

  **REAL** — Действительные числа. Напр., 201, -70.5, 66,0 .

  **STRING** — Последовательность символов. Например, hello.

  **BOOL** — Значение истины, **0** (неверно) или **1** (верно).

- Необходимо задать тип переменной, например:

```
INT Count = 5
REAL Diameter = 2.5
STRING Tapefile = "MyFile.tap"
```

- Можно обращаться к любому параметру PowerMILL при задании переменных, в выражениях или назначениях.
- Любые переменные созданные в макросе, доступны только из этого макроса. Когда выполнение макроса закончится, переменные будут больше не доступны и не могут быть использованы в выражениях или других макросах.
- Если необходимо создать переменную, которая может быть использована в любое время в проекте PowerMILL, тогда следует создать **Пользовательский параметр**.

## Присваивание параметров

Когда вы присваиваете значение переменной, выражение вычисляется и результат присваивается, а фактическое выражение не сохраняется.
Это аналогично использованию модификатора **EVAL** в команде параметра PowerMILL **EDIT PAR**.
Следующие два оператора равнозначны:

```
EDIT PAR "Stepover" EVAL "Tool.Diameter * 0.6"
$Stepover = Tool.Diameter * 0.6
```

*Имена параметров и переменных могут дополнительно иметь префикс $. В большинстве случаев, вы можете опустить префикс $ , но он ДОЛЖЕН быть использован, когда вы присваиваете значение любой переменной или параметру внутри макроса.*

## Ввод значений в макросы

Диалог ввода позволяет вводить конкретные значения в макрос. Основная схема:

```
$<переменная> = INPUT <string-prompt>
```

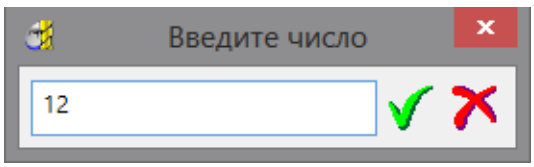
Это отобразит диалог ввода с конкретным заголовком, который позволяет ввести значение.

*Если вы добавите диалог ввода, вам следует рассмотреть добавление функции проверки ошибок, чтобы удостовериться, что введенные значения являются корректными.*

Например:

```
string prompt = "Введите число"
  $i = input $prompt
  $err = ERROR i
  }
```

Выведет такой диалог:




Вы также можете использовать **INPUT** при задании переменной.

Например:

```
REAL X = INPUT "Введите число"
```

## Задание вопроса Да/Нет

Диалог запроса **Да/Нет** очень простой диалог.
Выбор ответа **Да** присваивает значение **1** переменной. Выбор ответа **Нет** присваивает значение **0** переменной.

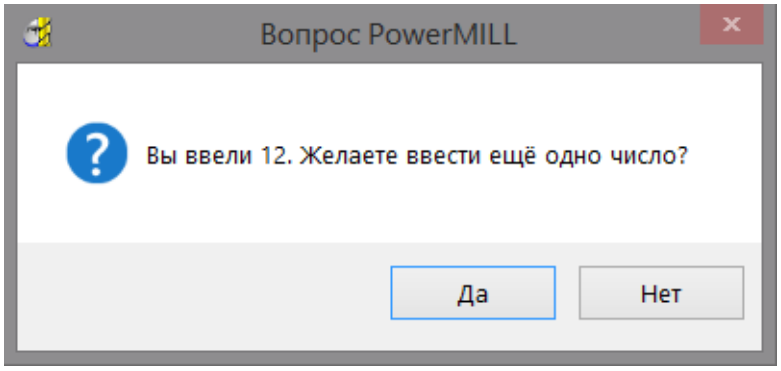
Основная схема:

```
$<переменная> = QUERY <string-prompt>
```

Например:

```
string yesnoprompt = "Вы ввели 12. Желаете ввести ещё одно число?"
bool carryon = 0
$carryon = query $yesnoprompt
```

выводит такой диалог:




## Создание диалога с сообщением

Существует три типа диалогов с сообщениями:

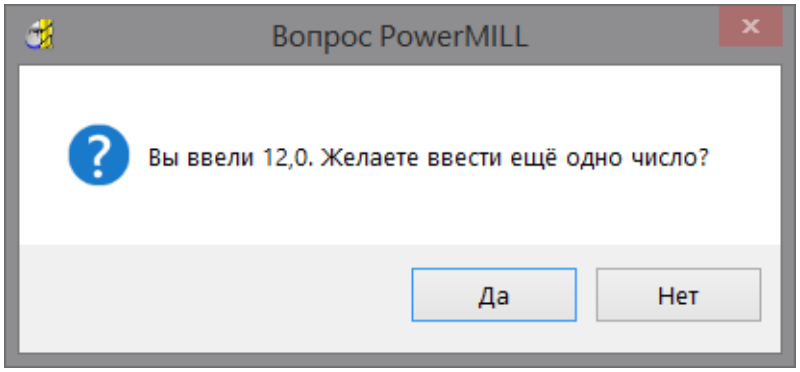
- Информационные диалоги
- Диалоги предупреждений
- Диалоги ошибок

Основная схема:

```
MESSAGE INFO|WARN|ERROR <выражение>
```

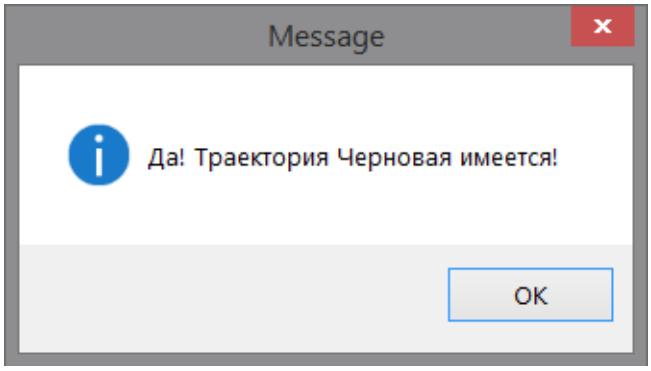
Например, диалог ввода, для ввода числа в макрос:

```
real i = 3
string prompt = "Введите число"
do {
  bool err = 0
  do {
    $i = input $prompt
    $err = ERROR i
    if err {
      $prompt = "Пожалуйста 'Введите число'"
    }
  } while err
  string yesnoprompt = "Вы ввели " + string(i) + ". Желаете
  ввести ещё одно число?"
  bool carryon = 0
  $carryon = query $yesnoprompt
} while $carryon
message info "Спасибо!"
```




Следующий пример помогает найти траекторию по имени:

```
string name = ""
$name = input "Введите имя траектории"
if pathname('toolpath',name) == "" {
  message error "Извините. Не могу найти траекторию " +
  name
} else {
  message info "Да! Траектория " + name + " имеется!"
}
```



## Массивы

Вдобавок к простым переменным  INT, REAL, или STRING можно также создавать их массивы. Когда вы объявляете массив необходимо инициализировать все его члены, используя список инициализации. Синтаксис для массива следующий:

```
BASIC-TYPE ARRAY name[n] = {…}
```

Например, чтобы объявить массив трёх строк:

```
STRING ARRAY MyArray[3] = {'First','Second','Third'}
```

Все элементы в списке инициализации должны быть такого же основного типа как и массив.

Доступ к элементам массива происходит с помощью индексации. Первый элемент массива имеет индекс 0. Например:

```
INT Index = 0
WHILE Index < size(MyArray) {
  PRINT = MyArray[Index]
  $Index = Index + 1
}
```

Выведет:

**First**

**Second**

**Third**

Массивы можно использовать в циклах **FOREACH** .

## Сравнение переменных

Сравнение переменных позволяет проверить информацию и задать образ действий при использовании операторов **IF** и **WHILE**.

Результат сравнения либо истина, либо ложь. Когда истина – результат равен **1**, когда ложь – результат равен **0**.

Простое сравнение может состоять из двух переменных с оператором отношения между ними:

| Оператор отношения | | Описание |
|---|---|---|
| **Символ** | **Текст** | |
| == | EQ | равно |
| != | NE | не равно |
| < | LT | меньше чем |
| <= | GE | меньше или равно |
| > | GT | больше чем |
| >= | GE | больше или равно |

*Вы можете использовать в сравнении или символ или текст.*

Например,

```
BOOL C = (A == B)
```

тоже самое что и:

```
BOOL C = (A EQ B)
```

**C** присваивается **1** (истина) если **A** равно **B**. Если **A** не равно **B**, тогда **C** равно **0** (ложь).

*Операторы = и == различны.*

*Одиночный оператор равно, =, присваивает значение с правой стороны левой стороне.*

*Двойной оператор равно, ==, сравнивает два значения на равнозначность.*

Если вы сравниваете значения в двух строках, вы должны использовать правильный регистр.

Например, если вы желаете проверить является ли инструмент концевой фрезой, тогда вы должны использовать:

```
Tool.Type == 'end_mill'
```

но не:

```
Tool.Type == 'End_Mill
```

Если вы не уверены в регистре строки, тогда вы можете использовать одну из встроенных функций **lcase()** или **ucase()** чтобы проверить варианты строки в нижнем или верхнем регистре

```
lcase(Tool.Type) == 'end_mill'
```

```
ucase(Tool.Type) == 'END_MILL'
```

Например, сравнение переменных:

```
BOOL bigger = (Tool.Diameter+Thickness
>=ReferenceToolpath.Tool.Diameter+ReferenceToolpath.Thick
ness)
```

выводит результат **1** (истина) когда **Tool.Diameter + Thickness** больше или равен **ReferenceToolpath.Tool.Diameter + ReferenceToolpath.Thickness** и результат **0** (ложь) в другом случае.

## *Логические операторы*

Логические операторы позволяют сделать больше, чем одно сравнение одновременно. Есть четыре логических оператора:

- AND
- OR
- XOR
- NOT

*Результатом сравнения является или истина или ложь. Когда истина, результат равен **1** ; когда ложь, результат равен **0**.*

### Использование логического оператора AND

Результатом будет истина (**1**) если все операнды истинны, в другом случае результатом будет ложь (**0**).

| Операнд 1 | Операнд 2 | Операнд 1 AND Операнд 2 |
|---|---|---|
| истина (1) | истина (1) | истина (1) |
| истина (1) | ложь (0) | ложь (0) |
| ложь (0) | истина (1) | ложь (0) |
| ложь (0) | ложь (0) | ложь (0) |

### Использование логического оператора OR

Результатом будет истина (**1**) если по крайней мере один операнд истинен. Если все операнды ложны (**0**) результатом будет ложь.

| Операнд 1 | Операнд 2 | Операнд 1 OR Операнд 2 |
|---|---|---|
| истина (1) | истина (1) | истина (1) |
| истина (1) | ложь (0) | истина (1) |
| ложь (0) | истина (1) | истина (1) |
| ложь (0) | ложь (0) | ложь (0) |

### Использование логического оператора XOR

Результатом будет истина (**1**) если только один операнд истинен. Если все операнды ложны, результатом будет ложь (**0**). Если больше чем один операнд будет истина, результатом будет ложь (**0**).

| Операнд 1 | Операнд 2 | Операнд 1 XOR Операнд 2 |
|---|---|---|
| истина (1) | истина (1) | ложь (0) |
| истина (1) | ложь (0) | истина (1) |
| ложь (0) | истина (1) | истина (1) |
| ложь (0) | ложь (0) | ложь (0) |

### Использование логического оператора NOT

Результатом будет значение, обратное введённому.

| Операнд 1 | NOT Операнд 1 |
|---|---|
| истина (1) | ложь (0) |
| ложь (0) | истина (1) |

## Дополнительные параметры переменных

### *Временные переменные*

Можно создавать и управлять переменными в командном окне. Они называются временными переменными и их можно использовать чтобы проверить результаты вычисления параметров без необходимости писать макрос.

Например, чтобы протестировать часть кода, наберите в командном окне по очереди:

```
STRING Test = Tool.Name
DEACTIVATE TOOL
ACTIVATE TOOL $Test
```

```
PowerMILL >
PowerMILL > STRING Test = Tool.Name
PowerMILL > DEACTIVATE TOOL
PowerMILL > ACTIVATE TOOL $Test
PowerMILL >
```

Чтобы очистить временные переменные, наберите в командном окне:

```
RESET LOCALVARS
```

Если не использовать команду RESET LOCALVARS, локальная переменная Test останется заданной, пока вы не выйдете из PowerMILL.

## Использование переменных и параметров в командах макроса

Вы можете подставить значение переменной или параметра в команду, когда команда ожидает ввода числа или строки. Чтобы сделать это, добавьте префикс **$** к переменной или параметру.

Например, чтобы создать инструмент с диаметром, который составляет половину от активного инструмента:

```
// Вычислить новый диаметр и имя инструмента

REAL HalfDiam = Tool.Diameter/2
STRING NewName = string(Tool.Type) + " D-" + string(HalfDiam)

// Создать новый инструмент и сделать его активным
COPY TOOL ;
ACTIVATE TOOL #

// Переименовать инструмент и изменить его диаметр
RENAME TOOL ; $NewName
EDIT TOOL $NewName DIAMETER $HalfDiam
```

Это создаст инструмент с пол-диаметра активного инструмента.

## Область использования переменных

Переменная существует со времени её объявления и до конца блока кода внутри которого она была объявлена. Блоками кода могут быть макросы и управляющие структуры (**WHILE**, **DO - WHILE**, **SWITCH**, **IF-ELSEIF- ELSE**, и **FOREACH**).

Переменная с собственным именем может быть задана только один раз внутри любого блока кода.

Например, код:

```
// Задание локальной переменной 'Count'
INT Count = 5
// Задание второй локальной переменной 'Count'
INT Count = 2
```

выдаст ошибку, так как переменная `Count` будет задана дважды.

Однако, во внутреннем блоке кода можно задать другую переменную, с таким же именем как и переменная (уже заданная) во внешнем блоке:

```
INT Count = 5
IF Count > 0 {
// Задание новой локальной переменной 'Count'
INT Count = 3
// Выводим 3
PRINT $Count
// Локальная переменная Count больше не определена
}
// Выводим 5
PRINT $Count
```

Переменная, заданная в пределах внутреннего блока кода скрывает любую переменную, объявленную во внешнем блоке. Это также важно, если вы используете имя переменной, которая соответствует одному из параметров PowerMill.

Например, если радиальный шаг (stepover) равен 5 и ваш макрос содержит:

```
// 'Скрыть' глобальный рад. шаг созданием собственной
переменной

REAL Stepover = 2

// Вывести радиальный шаг

PRINT $Stepover
```

будет выведено значение **2** а не **5**, и значение радиального шага траектории не изменится. Чтобы получить доступ к текущему параметру радиального шага траектории, необходимо использовать **toolpath.Stepover**.

```
// 'Скрыть' глобальный рад. шаг созданием собственной
переменной
REAL Stepover = 2
// Вывести 2
PRINT $Stepover
// Вывести значение радиального шага  траектории, который
равняется 5
PRINT $toolpath.Stepover
```

*Так как макропеременные прекращают существование в конце макроса или блока кода, вы не должны использовать переменную, заданную в макросе, внутри содержащего её выражения. Вы можете использовать присваивание, так как значение вычисляется немедленно. Не используйте макропеременную в выражении EDIT PAR без EVAL, так как это приведёт к ошибке выражения, когда PowerMILL попытается вычислить его.*

```
REAL Factor = 0.6
// Следующие две команды правильные, потому что выражение
// вычисляется немедленно.
$Stepover = Tool.Diameter * Factor
EDIT PAR "Stepover" EVAL "Tool.Diameter * Factor"

// Следующая команда неправильная, потому что выражение
// фиксируется.
EDIT PAR "Stepover" "Tool.Diameter * Factor"
```

Переменная Factor прекращает существование в конце макроса, поэтому Stepover будет расцениваться как ошибка.

# Использование выражений в макросах

Арифметическое выражение -это список переменных и значений с операторами, которые задают эти значения. Обычно их используют при присваивании и объявлении переменных.

```
// Объявление переменной
REAL factor = 0.6
REAL value = Tolerance * factor

// Присваивание
$Stepover = Tool.Diameter * factor
$factor = 0.75
```

*При использовании присваивания вы ДОЛЖНЫ использовать префикс переменной **$**. Таким образом PowerMILL сможет устранить неоднозначность при присваивании слов макроязыка.*

*В присваиваниях, выражение всегда вычислено и его значение присваивается переменной , находящейся слева от операнда **=** .*

Вдобавок к использованию выражений в расчётах, вы можете использовать логические выражения для принятия решений в макросах. Операторами принятия решений в PowerMILL являются **IF-ELSE-IF**, **SWITCH**, **WHILE** и **DO-WHILE**. Они выполняют команды внутри своего блока кода, если результатом выражения является истина (**1**).

Например:

```
IF active(Tool.TipRadiused) {
  // Выполняется, если активен скруглённый инструмент.
}
```

Или

```
IF active(Tool.TipRadiused) AND Tool.Diameter < 5 {
  // Выполняется, если активен скруглённый инструмент и
  // его диаметр меньше 5.
}
```

Вы можете использовать любое выражение, чтобы решить, будет или нет выполняться блок кода.

## Операторы для целых и действительных чисел

Стандартные арифметические операторы доступны для целых и действительных чисел

| Оператор | Описание | Примеры |
|---|---|---|
| + | Сложение | 3+5 равно 8 |
| - | Вычитание | 5-3 равно 2 |
| * | Умножение | 5*3 равно 15 |
| / | Деление | 6/2 равно 3 |
| % | Остаток после деления двух целых чисел | 11%3 равно 2 |
| ^ | Степень | 2^3 то же что и 2*2*2 и равно 8 |

Полный список операторов смотрите на HTML странице, отображаемой при выборе пункта меню в PowerMILL : **Помощь > Переменные > Справка > Functions**.

## Операторы для строк

Для строк доступен оператор сцепления (**+**) .

Например `"abc"+"xyz"` вычислится как abcxyz.

Это можно использовать для построения строк, состоящих из различных частей.

Например:

```
MESSAGE "Радиальный шаг: " + string(Stepover)
```

## Приоритет операторов

Порядок, в котором различные части выражения вычисляются, влияет на результат. Приоритет операторов однозначно определяет порядок, в котором вычисляются подвыражения.

- Умножение и деление выполняются перед сложением и вычитанием.

  Например, 3 * 4 +2 тоже самое что и 2 + 3 * 4 и даёт результат 14.

- Степени и корни выполняются до умножения и сложения.

  Например, 3 + 5 ^ 2 тоже самое что и 3 + $5^2$ и даёт результат 28.

  -3 ^ 2 тоже самое что и $-3^2$ и даёт результат -9.
- Используйте круглые скобки, чтобы избежать путаницы.

  Например, 2 + 3 * 4 тоже самое что и 2 + (3 * 4) и даёт результат 14.
- Круглые скобки изменяют порядок старшинства, так как выражения внутри скобок вычисляются в первую очередь.

  Например, (2 + 3) * 4 даёт результат 20.

  Или, (3 + 5) ^2 тоже самое что и $(3 + 5)^2$ и даёт результат 64.
- Вы должны взять аргументы функции в круглые скобки.

  Например, y = sqrt(2), y = tan(x), y = sin(x + z).
- Операторы отношения выполняются после сложения и вычитания.

  Например, a+b >= c+d тоже самое что и (a+b) >= (c+d).
- Логические операторы выполняются после реляционных операторов, тем не менее скобки часто добавляют для ясности.

  Например:

  5 == 2+3 OR 10 <= 3*3

  тоже самое что и:

  (5 == (2+3)) OR (10 <= (3*3))

  но обычно пишется как

  (5 == 2+3) OR (10 <= 3*3).

## Приоритет:

| Порядок | Действие | Описание |
|---|---|---|
| 1 | () | вызов функции, операции сгруппированы в круглых скобках |
| 2 | [] | операции сгруппированы в квадратных скобках |
| 3 | + - ! | унарный префикс (унарные операции имеют только один операнд, такой как, !x, -y) |
| 4 | cm mm um ft in th | преобразование единиц измерения |
| 5 | ^ | степени и корни |
| 6 | * / % | умножение, деление, процент |
| 7 | + - | сложение и вычитание |
| 8 | < <= > >= (LT, LE, GT, GE) | реляционные сравнения: меньше, меньше или равно, больше, больше или равно |
| 9 | == != (EQ, NE) | реляционные сравнения: равно, не равно |
| 10 | AND | логический оператор AND |
| 11 | NOT | логический оператор NOT |
| 12 | XOR | логический оператор XOR |
| 13 | OR | логический оператор OR |
| 14 | , | Разделение элементов в списке |

Примеры приоритета:

| Выражение | Эквивалент |
|---|---|
| a * - 2 | a * (- 2) |
| !x == 0 | (!x) == 0 |
| $a = -b + c * d – e | $a = ((-b) + (c * d)) – e |
| $a = b + c % d – e | $a = (b + (c % d)) – e |
| $x = y == z | $x = (y == z) |
| $x = -t + q * r / c | $x = ((-t) + ((q * r) / c)) |
| $x = a % b * c + d | $x = (((a % b) * c) + d) |
| $a = b <= c \| d != e | $a = ((b <= c) \| (d != e)) |
| $a = !b \| c & d | $a = ((!b) \| (c & d)) |
| $a = b mm * c in + d | $a = (((b mm) * (c in)) + d) |

# Макро функции

При запуске макроса можно использовать аргументы, такие как имя траектории, инструмент или допуск. Вы должны структурировать макрос, чтобы он мог принимать аргументы, созданные функцией под названием **Main**, а затем задать аргументы и их типы.

Например, макрос для полигонизации активной границы с заданным допуском:

```
FUNCTION Main(REAL tol) {
  EDIT BOUNDARY ; SMASH $tol
}
```

Макрос для задания диаметра указанного инструмента:

```
FUNCTION Main(
  STRING name
  REAL diam
)
{
  EDIT TOOL $name DIAMETER $diam
}
```

Чтобы запустить эти макросы с аргументами, добавьте аргументы в правильном порядке в конце команды **MACRO**:

```
MACRO MyBoundary.mac 0.5
MACRO MyTool.mac "ToolName" 6
```

Если вы используете функции в макросе, тогда все команды должны быть внутри тела функции. Это означает, что у вас должна быть функция **Main**, которая автоматически будет вызываться при запуске макроса.

```
FUNCTION CleanBoundary(string name) {
  REAL offset = 1 mm
  REAL diam = entity('boundary';name).Tool.Diameter
  // Удалить сегменты меньше чем диаметр инстр.
  EDIT BOUNDARY $name SELECT AREA LT $diam
  DELETE BOUNDARY $name SELECTED
  //Смещение наружу и внутрь для сглаживания гран.
  EDIT BOUNDARY $name OFFSET $offset
  EDIT BOUNDARY $name OFFSET ${-offset}
}
FUNCTION Main(string bound) {
  FOREACH bou IN folder(bound) {
  CALL CleanBoundary(bou.Name)
  }
}
```

Внутри функции вы можете создавать и использовать переменные, которые являются локальными для функции, также как и внутри цикла **WHILE**. Тем не менее, функция не может получить доступ к любой переменной, которая определена в другом месте макроса, если только эта переменная не была передана функции в качестве аргумента.

*В функции **CleanBoundary**, ${-offset} смещает границу с отрицательным смещением. Если вы хотите подставить значение выражения в команду PowerMILL вместо значения параметра, используйте синтаксис ${выражение}. Выражение может содержать любое допустимое выражение параметра PowerMILL, в том числе: вызовы встроенных функций; математические, логические операторы и операторы сравнения.*

*Так как этот макрос требует аргумент (имя границы), необходимо запустить его из командного окна. Чтобы запустить макрос **Clean_Boundary.mac** для границы **Cavity**, нужно ввести в командном окне macro Clean_Boundary "Cavity".*

## Функция Main

Если макрос имеет какие-либо функции:

- Он должен иметь одну, и только одну функцию, с именем **Main**.
- Функция **Main** должна быть первой вызываемой функцией.

Имя функции не чувствительно к регистру: **MAIN**, **main**, и **MaIn** это всё одна и та же функция.

Запуск макроса в котором функция **Main** вызывается либо с неправильным числом аргументов или с не соответствующими типами аргументов, вызывает ошибку.

Например, макрос:

```
MACRO MyTool.mac 6 "ToolName"
```

выдаст ошибку, так как макрос ожидает строку и затем число, а задаётся число и затем строка.

Если вы хотите повторить последовательность команд в различных точках в пределах макроса, вы можете использовать функции.

Например, если вы хотите удалить любые маленькие островки, которые меньше, чем диаметр инструмента и сгладить любые незначительные изломы после расчета границы, одним из решений будет повторение одной и той же команды после каждого вычисления границы:

```
EDIT BOUNDARY ; SELECT AREA LT Boundary.Tool.Diameter
DELETE BOUNDARY ; SELECTED
EDIT BOUNDARY ; OFFSET "1 mm"
EDIT BOUNDARY ; OFFSET "-1 mm"
```

Это то что надо, если у вас есть макрос, который создает одну границу, но если он создаст множество границ, вы остановите макрос из-за чрезмерных повторений. Однако, с помощью функции, можно определить последовательность единожды:

```
FUNCTION CleanBoundary(string name) {
  REAL offset = 1 mm
  REAL diam = entity('boundary';name).Tool.Diameter
  // Удалить сегменты меньше чем диаметр инстр.
  EDIT BOUNDARY $name SELECT AREA LT $diam
  DELETE BOUNDARY $name SELECTED
  //Смещение наружу и внутрь для сглаживания гран.
  EDIT BOUNDARY $name OFFSET $offset
  EDIT BOUNDARY $name OFFSET ${-offset}
}
```

и затем вызывать её каждый раз, когда это необходимо:

```
FOREACH bou IN folder('boundary') {
  CALL CleanBoundary(bou.Name)
}
CREATE BOUNDARY Shallow30 SHALLOW
EDIT BOUNDARY Shallow30 CALCULATE
CALL CleanBoundary('Shallow30')
```

## Возвращение значений из функций

Есть два типа аргументов функций:

- Входные переменные ($ Input аргументы). Если параметр является входным, то любые изменения в параметре внутри функции теряются, когда происходит возврат из функции. Это значение по умолчанию.
- Выходные переменные ($ Outut аргументы) сохраняют свое значение после возврата из функции.

При вызове функции PowerMILL создает временные копии всех аргументов функции; эти копии будут удалены при возврате из функции. Тем не менее, если макрос содержит **OUTPUT** в качестве аргумента, то вместо создания временной копии переменной, он создает алиас (псевдоним идентификаторов) для существующей переменной. Любые изменения, внесенные в алиас, изменят фактическую переменную.

Например, функция **Test** содержит два аргумента: **aInput** и **aOutput**. Внутри этой функции:

- Аргумент **aInput** это новая временная переменная, которая существует только внутри функции; любые изменения её значения будут действовать только временно, и будут потеряны при выходе из функции.
- Переменная **aOutput** это алиас для переменной, которая была передана в команду **CALL**, любые изменения её значения фактически изменят значение переменной, которое передаётся в команду **CALL**.

```
FUNCTION Test(REAL aInput, OUTPUT REAL aOutput) {
  PRINT $aInput
  $aInput = 5
  PRINT $aOutput
  $aOutput = 0
  PRINT $aOutput
}

FUNCTION Main() {
  REAL Par1 = 2
  REAL Par2 = 1
  CALL Test(Par1, Par2)
  // Выводит 2 - значение не меняется
  PRINT $Par1
  // Выводит 0 - значение изменено
  PRINT $Par2
}
```

Когда выполняется команада **CALL** в функции **MAIN**:

1. PowerMILL создаёт новую действительную переменную **REAL** с именем **aInput**. Ей присваивается значение **Par1**, и передаётся в **Test**.
2. PowerMILL передаёт **Par2** напрямую в **Test,** где он называется **aOutput**.

## Распределение функций между макросами

Вы можете распределить функции между макросами используя оператор **INCLUDE**. Если поместить все общие функции в один файл, то его затем можно будет включить в другие макросы с помощью оператора **INCLUDE**. Например, если поместить функцию **CleanBoundary** в файл с названием **common.inc**, тогда следует переписать макрос таким образом:

```
INCLUDE common.inc
FUNCTION Main(input string bound) {
  FOREACH bou IN folder(bound) {
    CALL CleanBoundary(bou.Name)
  }
}
```

Чтобы вызвать этот макрос из PowerMILL:

```
// Очистить все границы
MACRO Clean 'boundary'
// Очистить все черновые границы
MACRO Clean 'boundary\Roughing'
```

# Оператор IF

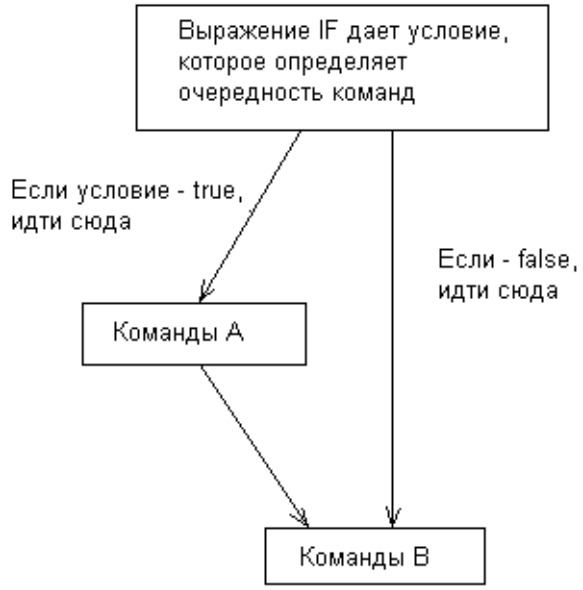
Оператор **IF** выполняет серию команд, когда встречаются определённые условия.

Основная управляющая структура:

```
IF <выражение> {
  Команды A
}
Команды B
```

Если **выражение** является истиной, тогда выполняются **Команды A**, и следующие за ними **Команды B.**

Если **выражение** ложно, тогда выполняются только **Команды B**.




Например, если вы хотите вычислить траекторию, но не хотите тратить время на повторное вычисление траектории, которая была уже вычислена:

```
// Если активная траектория не вычислена, вычислить сейчас
IF NOT Computed {
  EDIT TOOLPATH $TpName CALCULATE
}
```

Необходимо заключить **Команды A** в фигурные скобки, **{}**, которые должны быть правильно расположены. Например, следующая команда неправильная:

```
IF (radius == 3) PRINT "Неправильный радиус"
```

Чтобы сделать эту команду правильной, добавьте фигурные скобки:

```
IF (radius == 3) {
  PRINT "Неправильный радиус"
}
```

*Первая фигурная скобка должна быть последним символом в строке, и должна находится на той же строке, что и **IF**.*

*Закрывающая фигурная скобка должна быть на отдельной строке.*

# Оператор IF - ELSE

Оператор **IF - ELSE** выполняет серию команд когда встречаются определённые условия и другую серию команд в противном случае.

Основная управляющая структура:

```
IF <выражение> {
  Команды A
} ELSE {
  Команды B
}
Команды C
```

Если **выражение** истинно, тогда выполняются **Команды A**, и следующие за ними **Команды C**.

Если **выражение** ложно, тогда выполняются **Команды B** и следующие за ними **Команды C**.


```
// Если фреза скруглённая, задать ось фрезы как
// Атаки/Наклона.
// В противном случае использовать вертикальную ось фрезы.
IF active(Tool.TipRadius) OR Tool.Type == "ball_nosed" {
  EDIT TOOLAXIS TYPE LEADLEAN
  EDIT TOOLAXIS LEAD "5"
  EDIT TOOLAXIS LEAN "5"
} ELSE {
  EDIT TOOLAXIS TYPE VERTICAL
}
```

*Чтобы заработал данный пример с оператором **IF - ELSE**, необходимо этот блок кода вставить в цикл, например в цикл **FOREACH**.*

# Оператор IF - ELSEIF - ELSE

Оператор **IF - ELSEIF - ELSE** выполняет серию команд когда встречаются определённые условия, и другую серию команд, когда первое условие не встречается, а встречается второе условие, и третью серию команд, когда ни одно из условий не встречается.

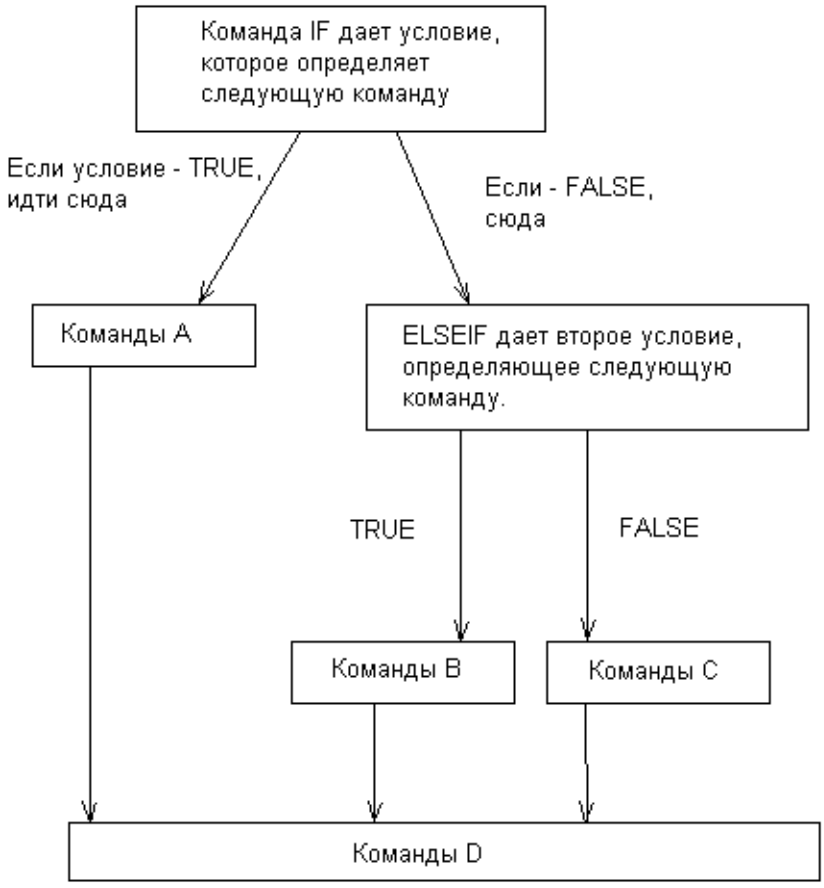
Основная управляющая структура:

```
IF <выражение_1> {
  Команды A
} ELSEIF <выражение_2> {
  Команды B
} ELSE {
  Команды C
}
Команды D
```

Если **выражение_1** истинно, тогда выполняются **Команды A** и следующие за ними **Команды D**.

Если **выражение_1** ложно, а **выражение_2** истинно, тогда выполняются **Команды B** и следующие за ними **Команды D**.

Если **выражение_1** ложно и **выражение_2** ложно, тогда выполняются **Команды C** и следующие за ними **Команды D**.



*__ELSE__ является необязательным оператором. В блоке кода может быть любое количество операторов __ELSEIF__, но не больше одного __ELSE__.*

```
IF Tool.Type == "end_mill" OR Tool.Type == "ball_nosed" {
  $radius = Tool.Diameter/2
} ELSEIF active(Tool.TipRadius) {
  $radius = Tool.TipRadius
} ELSE {
  $radius = 0
  PRINT "Неправильный тип инструмента"
}
```

Это задаст переменную **radius** как:

- Половину диаметра инструмента, если инструмент является концевой или шаровой фрезой.
- Радиус кромки, если инструмент является скруглённой фрезой.
- Выведет **Неправильный тип инструмента**, если инструмент будет другой.

## Оператор SWITCH

Если сравнить переменную с числом возможных значений и каждое значение определяет вывод различного результата, желательно использовать оператор **SWITCH**.

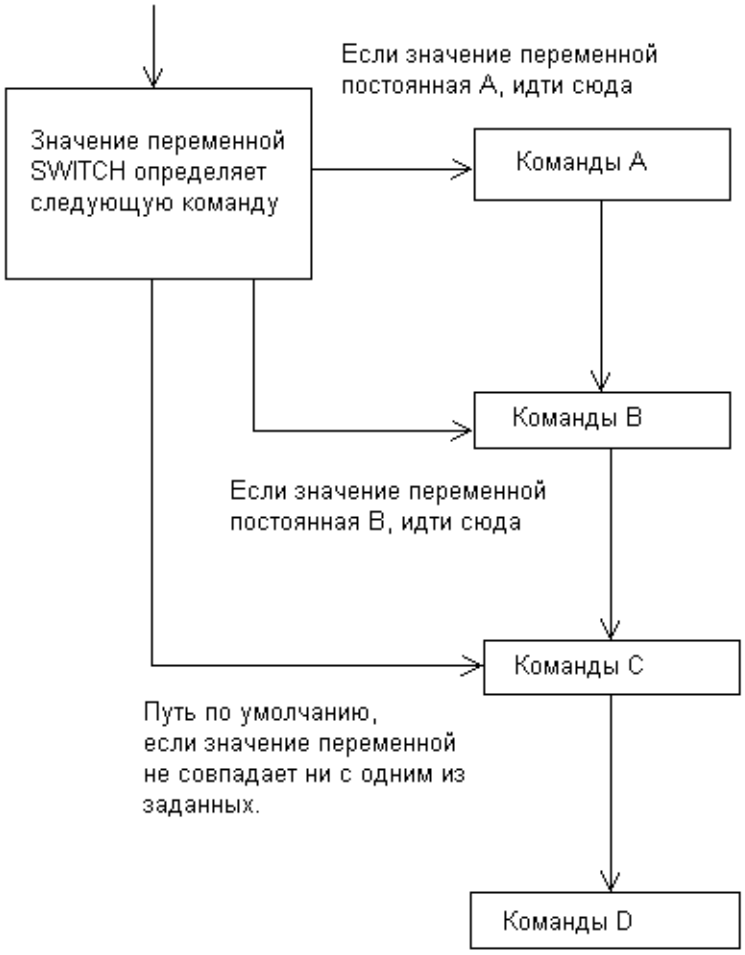
Оператор **SWITCH** позволяет сравнивать переменную со списком возможных значений. Это сравнение определяет, какие команды будут выполняться.

Основная управляющая структура:

```
SWITCH variable {
  CASE (постоянная_A)
      Команды A
  CASE (постоянная_B)
      Команды B
DEFAULT
      Команды C
}
Команды D
```

Если **постоянная_A** истинна, тогда выполняются **Команды A**, **B**, **C**, и **D**. Если **постоянная_B** истинна, тогда выполняются **Команды B**, **C**, и **D**.

Если **постоянная_A** и **постоянная_B** ложны, тогда выполняются **Команды C**, и **D**.




*Когда совпадение будет найдено, все команды в оставшихся операторах **CASE** будут выполнены. Это можно предотвратить с помощью оператора **BREAK**.*

*Вы можете иметь любое число операторов **CASE**, но не более одного оператора **DEFAULT**.*

Следующий пример вносит изменения в распределение точек в зависимости от типа оси инструмента. Есть три варианта:

1. 3+2- осевые траектории будут иметь тип распределения точек **По допуску, сохраняя дуги** и расстояние подводов и отводов равное 200.
2. 3- осевые траектории будут иметь тип распределения точек **По допуску, сохраняя дуги**.
3. 5- осевые траектории будут иметь тип распределения точек **Переразместить**.

*Так как блок кода **CASE 'direction'** не имеет оператора **BREAK**, макрос также выполняет код в блоке **'vertical'**.*

```
SWITCH ToolAxis.Type {
  CASE 'direction'
    EDIT TOOLPATH LEADS RETRACTDIST "200.0"
    EDIT TOOLPATH LEADS APPROACHDIST "200.0"
    // Приступаем к выполнению
   CASE 'vertical'
    // Размещение точек "По допуску, сохраняя дуги"
    EDIT FILTER TYPE STRIP
    BREAK
  DEFAULT
    // Размещение точек "Переразместить"
    EDIT FILTER TYPE REDISTRIBUTE
    BREAK
}
```

*Чтобы заработал данный пример с оператором **SWITCH**, необходимо этот блок кода вставить в цикл, например в цикл **FOREACH**:*

```
FOREACH ent IN folder("toolpath") {
ACTIVATE TOOLPATH $ent.Name

.....

EDIT TOOLPATH ; REAPPLY
}
```

# Оператор BREAK в операторе SWITCH

Оператор **BREAK** останавливает работу оператора **SWITCH**.

Основная управляющая структура:

```
SWITCH variable {
  CASE (постоянная_A)
      Команды A
      BREAK
  CASE (постоянная_B)
      Команды B
      BREAK
  DEFAULT
      Команды C
}
```

Если **постоянная_A** истинна, тогда выполняются **Команды A** и следующие за ними **Команды D**.

*Запомните, если нет оператора **BREAK**, тогда выполняются команды **A**, **B**, **C**, и **D**.*

Если **постоянная_B** истинна, тогда выполняются **Команды B** и следующие за ними **Команды D**.

Если **постоянная_A** и **постоянная_B** ложны, тогда выполняются **Команды C** и следующие за ними **Команды D**.

# Повторение команд в макросах

Если вы хотите повторить набор команд несколько раз, например, создать окружность в начале каждой строки в модели, вы можете использовать циклы.

Например, если у вас есть две 2D Модели, **Top** и **Bottom**, содержащие отверстия, которые вы хотите просверлить сверху и снизу модели соответсвенно, используйте макрос:

```
STRING Fset = 'Top'

INT Count = 0
WHILE Count < 2 {
  ACTIVATE FEATURESET $Fset
  ACTIVATE WORKPLANE FROMENTITY FEATURESET $Fset
  IMPORT TEMPLATE ENTITY TOOLPATH "Drilling\Drilling.ptf"
  EDIT TOOLPATH ; CALCULATE
  $Fset = 'Bottom'
  $Count = Count + 1
}
```

Существует три структуры циклов:

- Циклы **FOREACH** многократно выполняют блок команд для каждого элемента в списке.
- Циклы **WHILE** многократно выполняют блок команд до тех пор, пока условие не является истинным.
- Циклы **DO - WHILE** сначала выполняют блок команд и только затем проверяют его условие.

## Цикл FOREACH

Цикл **FOREACH** многократно выполняет блоки команд для каждого элемента в списке или массиве.

Основная управляющая структура:

```
FOREACH item IN sequence{
  Команды A
}
Команды B
```

где:

**item** – это автоматически создаваемая переменная, которую PowerMILL инициализирует для каждого повторения цикла;

`sequence` - это или список или массив.

**Команды A** выполняются для первого элемента в списке.

**Команды A** выполняются для следующего элемента в списке. Этот шаг повторяется до тех пор, пока в списке больше не будет элементов.

В конце списка выполняются **Команды B**.


Например,

```
FOREACH item IN folder("path") {
  Команды A
}
Команды B
```

Где <path> это папка в Проводнике, такая как **Toolpath** (Траектории), **Tool** (Инструменты), **Toolpath\Finishing** и тд.

Внутри цикла **FOREACH** вы можете:

- Остановить цикл, используя оператор **BREAK**.
- Перейти непосредственно к следующему повторению, используя оператор **CONTINUE**.

Вы не можете создавать свой собственный список переменных, так как есть встроенные функции в PowerMILL, которые возвращают списки (смотрите документацию по параметрам для компонентов и папок).

Вы можете использовать одну из встроенных функций, чтобы получить список элементов, или вы можете использовать массивы, чтобы создать последовательность строк или чисел для перебора. Например, использовать встроенную функцию папки, чтобы получить список элементов.

Примером использования цикла FOREACH является пакетное вычисление Профилей Патрона:

```
FOREACH ent IN folder('Tool') {
  ACTIVATE TOOL $ent.Name
  EDIT TOOL ; UPDATE_TOOLPATHS_PROFILE
}
```

*Переменная цикла **ent** создаётся циклом и удаляется, когда цикл завершён.*

Другим примером является перенумерование всех инструментов в проекте:

```
INT nmb = 20
FOREACH t IN folder('Tool') {
  $t.number.value = nmb
  $t.number.userdefined = 1
  $nmb = nmb + 2
}
```

Чтобы получить максимальную отдачу от макро-функций, вы должны ознакомиться со встроенными функциями, описанными в **Помощь > Переменные > Справка**.

## Цикл WHILE

Цикл **WHILE** многократно выполняет блоки команд до тех пор, пока условие не является истинным.

Основная управляющая структура:

```
WHILE условие {
  Команды A
}
Команды B
```

Если **условие** истинно, тогда выполняются **Команды A**.

Пока **условие** остаётся истинным, то выполняются **Команды A**.

Когда **условие** ложно, выполняются **Команды B**.


Внутри цикла **WHILE** вы можете:

- Отменить цикл, используя оператор **BREAK**.
- Перейти непосредственно к следующему повторению, используя оператор **CONTINUE**.

## Цикл DO - WHILE

Цикл **DO - WHILE** выполняет блоки команд и затем выполняет проверку условия на истинность, в то время как цикл **WHILE** сначала выполняет проверку условия и только потом решает, выполнять команды или нет.

Основная управляющая структура:

```
DO {
  Команды A
} WHILE условие
Команды B
```

Выполняются **Команды A**.

Пока **условие** остаётся истинным, выполняются **Команды A**

Когда **условие** ложно, выполняются **Команды B**.


С циклом **DO - WHILE** вы можете:

- Отменить цикл, используя оператор **BREAK**.
- Перейти непосредственно к следующему повторению, используя оператор **CONTINUE**.

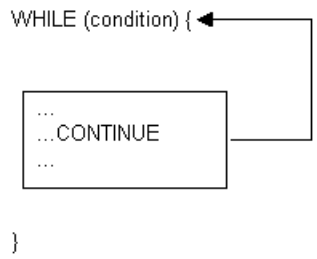
## Оператор CONTINUE

Оператор **CONTINUE** вызывает переход к проверке условия любой структуры цикла - **WHILE**, **DO - WHILE**, и **FOREACH** в которых он встречается, и начинает следующее повторение, если таковые имеются.

Следующий пример рассчитывает и смещает все незаблокированные границы  наружу и внутрь.

```
FOREACH bou IN folder('Boundary') {
  IF locked(bou) {
    // Эта граница блокирована, переходим к следующей
    CONTINUE
  }
  REAL offset = 1 mm
  EDIT BOUNDARY $bou.Name CALCULATE
  EDIT BOUNDARY $bou.Name OFFSET $offset
  EDIT BOUNDARY $bou.Name OFFSET ${-offset}
  }
```

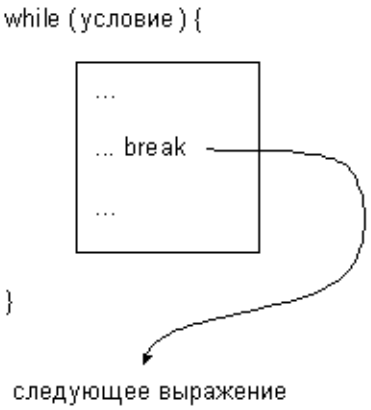
Оператор **CONTINUE** позволяет выбрать следующую границу.




## Оператор BREAK в цикле WHILE

Оператор **BREAK** завершает цикл **WHILE**.

*Вложенные конструкции могут требовать несколько операторов **BREAK**.*




## Оператор RETURN

Оператор **RETURN** немедленно прекращает выполнение текущего макроса. Это полезно, если обнаружена ошибка, и вы не хотите продолжать выполнять оставшиеся команды в макросе.

Основная управляющая структура:

```
EDIT TOOLPATH $tp.Name CALCULATE
IF NOT Computed {
  // Остановить,если траектория не вычислена
  RETURN
}
```

Вы также можете использовать оператор **RETURN**, чтобы немедленно выйти из функции.

```
FUNCTION Calculate(STRING TpName) {

  IF NOT active(entity('toolpath',TpName).Tool.TipRadius) {
    // Ошибка, если траектория не использует
    // скруглённый инструмент
    PRINT "Траектория не содержит скруглённый инстр."
    RETURN
  }

  EDIT TOOLPATH ; CALCULATE
}

FUNCTION Main() {

  FOREACH tp IN folder('Toolpath') {
    ACTIVATE TOOLPATH $tp.Name
    CALL Calculate(tp.Name)
  }
}
```

## Вывод значений выражений

Чтобы вывести значения скалярных выражений или параметров, используйте следующий синтаксис:

```
PRINT = выражение
```

Например, для вывода значения простого арифметического выражения, введите:

```
PRINT = 2*5
```

Когда вы запустите макрос, командное окно выведет результат 10.

```
PowerMILL >
PowerMILL >
Выберите файл >10,0
PowerMILL >
```

Вы также можете применять арифметические выражения для задания значений параметров. Например:

```
EDIT TOOL ; DIAMETER 10
PRINT = Tool.Diameter * 0.6
```

Когда вы запустите макрос, командное окно выведет результат 6.

# Встроенные функции

Этот раздел содержит описание всех встроенных функций, которые вы можете использовать при написании макросов.

- Общие математические функции.
- Тригонометрические функции.
- Функции работы со строками.
- Функции создания списка.
- Функции пути (имена Папок, Каталогов и Базы).
- Условные функции.
- Функции преобразования типов.
- Функции параметров.
- Статистические функции.

## Общие математические функции

Основная структура общих математических функций:

| Описание возвращаемого значения | Функция |
|---|---|
| Экспоненциальная функция | `real exp( real a )` |
| Натуральный логарифм | `real ln( real a )` |
| Десятичный логарифм | `real log( real a )` |
| Квадратный корень | `real sqrt( numeric a )` |
| Абсолютный (положительное значение) | `real abs( numeric a )` |
| Возвращает либо -1, 0 или 1 в зависимости от знака величины | `real sign( numeric a )` |
| Возвращает 1 или 0 в зависимости от того, будет ли разница между `a` и `b` меньше или равна допуску | `real compare( numeric a numeric b numeric tol )` |

## Тригонометрические функции

Основная структура тригонометрических функций:

| Описание возвращаемого значенияя | Функция |
|---|---|
| Тригонометрический синус | `real sin( angle ∅ )` |
| Тригонометрический косинус | `real cos( angle ∅ )` |
| Тригонометрический тангенс | `real tan( angle ∅ )` |

| | |
|---|---|
| Тригонометрический арксинус | `real asin( real a )` |
| Тригонометрический арккосинус | `real acos( real a )` |
| Тригонометрический арктангенс | `real atan( real a )` |
| Тригонометрический арктангенс `a`/`b`, квадрант определяется знаком двух аргументов | `real atan2( real a, real b )` |

## Функции работы со строками

Параметры и переменные PowerMILL могут содержать строки символов. Есть целый ряд встроенных функций, которые можно использовать для тестирования и работы со строками.

Основная структура строковых фукнций:

| Описание возвращаемого значения | Функция |
|---|---|
| Возвращает количество символов в строке. | `int length( string str )` |
| Возвращает позицию строки **target** от начала строки **str**, или **-1** если строка **target** не найдена.<br>Если использовать дополнительный аргумент **start**, то сканирование начинается с этой позиции в строке. | `int position( string str, string target[, numeric start] )` |
| Заменяет все вхождения строки **target** строкой **replacement**. Исходная строка не изменяется. | `string replace( string str, string target, string replacement)` |

| | |
|---|---|
| Возвращает часть строки. Вы можете определить, где подстрока начинается и её длину. Исходная строка не изменяется. | `string substring( string str, int start, int length)` |
| Возвращает в верхнем регистре строку. Исходная строка не изменяется. | `string ucase( string str)` |
| Возвращает в нижнем регистре строку. Исходная строка не изменяется. | `string lcase( string str)` |

Первый символ строки всегда с индексом **0**. Вы можете присоединять (добавлять) строки вместе, используя оператор **+**. Например:

```
STRING One = "One"
STRING Two = "Two"
STRING Three = "Three"
PRINT = One + ", " + Two + ", " + Three
```

Когда вы запустите макрос, командное окно отобразит результат **One**, **Two**, **Three**.

```
PowerMILL >
PowerMILL > One, Two, Three
PowerMILL >
```

Ещё один способ получения такого-же результата:

```
STRING One = "One"
STRING Two = "Two"
STRING Three = "Three"
STRING CountToThree = One + ", " + Two + ", " + Three
PRINT = CountToThree
```

Когда вы запустите макрос, командное окно отобразит результат **One**, **Two**, **Three.**

## *Преобразование числового значения в строковое*

Вы можете использовать строковые фукнции для преобразования числового значения в строковое значение.

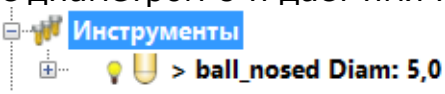
Основная структура:

```
STRING string( numeric str )
```

Это полезно, когда вы хотите добавить числа в строку. Например, предположим, что вы хотите назвать инструмент таким образом, чтобы он содержал в названии тип инструмента и его диаметр:

```
CREATE TOOL ; BALLNOSED

EDIT TOOL ; DIAMETER 5
STRING TName = string(Tool.Type) + " Diam: " +
string(Tool.Diameter)
RENAME TOOL ; $TName
PRINT = Tool.Name
```

Когда вы запустите макрос, PowerMILL создаст шаровую фрезу с диаметром 5 и даст имя инструменту как **ball_nosed Diam: 5.0**.




а командное окно отобразит результат **ball_nosed Diam: 5.0**.

```
PowerMILL >
PowerMILL > ball_nosed Diam: 5,0
PowerMILL >
```

## *Функция числа элементов в строке*

Функция числа элементов возвращает количество символов в строке. Основная структура:

```
int length( string str )
```

Например:

```
STRING One = "One"
PRINT = length(One)
```

Командное окно отобразит результат **3**.

```
PowerMILL >
PowerMILL > 3
PowerMILL >
```

Другой пример:

```
STRING One = "One"

STRING Two = "Two"
STRING CountToTwo = One + ", " + Two
PRINT = length(CountToTwo)
```

Командное окно отобразит результат **8**.

```
PowerMILL >
PowerMILL > 8
PowerMILL >
```

Ещё один способ получения такого-же результата:

```
STRING One = "One"
STRING Two = "Two"
PRINT = length(One + ", " + Two )
```

Командное окно отобразит результат **8**.

### *Функция позиционирования в строке*

Возвращает позицию строки **target** от начала строки **str**, или **-1**, если строка **target** не найдена.

Если использовать дополнительный аргумент **start**, то сканирование начинается с этой позиции в строке.

Основная структура:

```
int position( string str, string target [, numeric start] )
```

Например:

```
PRINT = position("Scooby doo", "oo")
```

Командное окно отобразит результат **2**. PowerMILL находит первое совпадение **oo** и вычисляет, какая у него позиция (**S** имеет позицию 0, **c** позицию 1 и **o** позицию 2).

```
PRINT = position("Scooby doo", "oo", 4)
```

Командное окно отобразит результат **8**. PowerMILL находит первое совпадение **oo** после позиции 4 и вычисляет, какая у него позиция (**b** имеет позицию 4, **y** позицию 5, " " позицию 7 и **o** позицию 8).

```
PRINT = position("Scooby doo", "aa")
```

Командное окно отобразит результат **-1**, так как PowerMILL не может найти совпадений **aa**.

Вы можете использовать эту функцию, чтобы проверить, существуют ли подстроки внутри другой строки. Предположим, что у вас есть деталь, которая содержит полость (cavity), и что вы вычислили траектории с грубым допуском и что в имени каждой траектории, которая обрабатывает этот элемент, есть слово **CAVITY**. Таким образом у вас есть траектории с именами, например, **CAVITY AreaClear**, **CAVITY flats**. Если вы хотите пересчитать эти траектории с более точным допуском, вы можете это сделать с помощью команд макроса:

```
// цикл для всех траекторий
FOREACH tp IN folder('Toolpath') {
  // если в имени траектории есть 'CAVITY'
  IF position(tp.Name, "CAVITY") >= 0 {
    // Отмена вычислений траекторий
    INVALIDATE TOOLPATH $tp.Name
    $tp.Tolerance = tp.Tolerance/10
  }
}
BATCH PROCESS
```

### *Замена одной строки на другую строку*

Заменяет все вхождения строки **target** строкой **replacement**. Исходная строка не изменяется.

Основная структура:

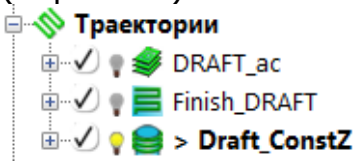
```
string replace( string str, string target, string
replacement)
```

Например:

```
STRING NewName = replace("Scooby doo", "by", "ter")

PRINT = NewName
```

Командное окно отобразит результат Scooter doo.

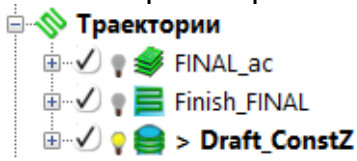
Ещё один пример - пока вы пытаетесь вычислить траектории с различными значениями, вы можете добавить слово **DRAFT** (черновик) к имени каждой траектории.




Когда вы будете удовлетворены конкретными траекториями, вы захотите изменить **DRAFT** на **FINAL**. Чтобы спасти себя от ручного редактирования имён траекторий, можно использовать макрос для переименования траекторий.

```
FOREACH tp IN folder('Toolpath') {
  ACTIVATE TOOLPATH $tp.Name
   STRING NewName = replace(Name, 'DRAFT', 'FINAL')
   RENAME TOOLPATH ; $NewName
}
```

Этот макрос переименует траектории на:




*Любое совпадение слова **DRAFT** в имени траектории будет изменено на **FINAL**. Однако, макрос чувствителен к регистру, так что совпадение **Draft** не изменится.*

Кроме того, можно написать макрос для переименования траектории без её активации:

```
FOREACH tp IN folder('Toolpath') {
  STRING NewName = replace(tp.Name, 'DRAFT', 'FINAL')
  RENAME TOOLPATH $tp.Name $NewName
}
```

## Подстроки

Функция подстроки возвращает часть строки. Вы можете определить, где начинается подстрока, а её длину. Исходная строка не изменяется.

Основная структура is:

```
string substring( string str, int start, int length)
```

Например:

```
PRINT = substring("Scooby doo", 2, 4)
```

Командное окно отобразит результат **ooby**.

### *Функция верхнего регистра в строке*

Функция верхнего регистра конвертирует строку в верхний регистр. Исходная строка не изменяется.

Основная структура:

```
string ucase( string str)
```

Например:

```
PRINT = ucase("Scooby doo")
```

Командное окно отобразит результат **SCOOBY DOO**.

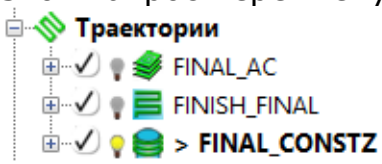
```
PowerMILL >
PowerMILL > SCOOBY DOO
PowerMILL >
```

В предыдущем примере замены одной строки на другую, совпадение **DRAFT** было заменено на **FINAL**, а совпадение **Draft** нет.

Использование оператора *ucase* заменит совпадения **Draft**, **draft**, **dRAft** на **DRAFT**. Остаток макроса заменит **DRAFT** на **FINAL**.

```
FOREACH tp IN folder('Toolpath') {
  // Получение имени в верхнем регистре
  STRING UName = ucase(tp.Name)
  // Проверка, содержит ли имя 'DRAFT'
  if position(UName, 'DRAFT') >= 0 {
  // Замена DRAFT на FINAL
  STRING NewName = replace(UName, 'DRAFT', 'FINAL')
  RENAME TOOLPATH $tp.Name $NewName
  }
}
```

Этот макрос переименует траектории на:




Траектория **Draft_ConstZ** не была переименована в прошлый раз, а переименовалась только сейчас. Все имена траекторий теперь в верхнем регистре.

### *Функция нижнего регистра в строке*

Функция нижнего регистра конвертирует строку в нижний регистр. Исходная строка не изменяется.

Основная структура:

```
string lcase( string str)
```

Например:

```
PRINT = lcase("SCOOBY DOO")
```

Командное окно отобразит результат **scooby doo**.

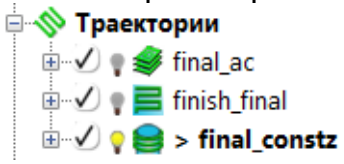
При замене одной строки на другую совпадение **DRAFT** будет заменено на **FINAL**, а совпадение **Draft** не будет.

Функция верхнего регистра конвертирует совпадения **Draft**, **draft**, **dRAft** на **DRAFT**.

Оператор lcase изменяет имена траекторий с верхнего регистра на нижний регистр.

```
FOREACH tp IN folder('Toolpath') {
  // Получение имени в верхнем регистре
  STRING UName = ucase(tp.Name)
  // Проверка, содержит ли имя 'DRAFT'
  if position(UName, 'DRAFT') >= 0 {
  // Замена DRAFT на FINAL
  STRING NewName = replace(UName, 'DRAFT', 'FINAL')
  RENAME TOOLPATH $tp.Name $NewName
  // Замена верхнего регистра на нижний
  STRING LName = lcase(tp.Name)
  RENAME TOOLPATH $tp.Name $LName
  }
}
```

Этот макрос переименует траектории на:




Все имена траекторий теперь в нижнем регистре.

## Функции создания списка

Функции создания списка возвращают содержание списка. Основная структура этой функции:

| Описание возвращаемого значения | Функция |
|---|---|
| Возвращает компоненты другого объекта. | `list components( entity entity )` |
| Возвращает список всех элементов в папке. | `list folder( string folder )` |

## Список компонентов

Встроенная функция компонентов возвращает компоненты другого объекта.

*На данный момент поддерживаются параметры только двух элементов - **NC Program (NC Файлы)** и **Group (Группы)**.*

*Функция компонентов возвращает список всех элементов независимо от их типа. Вы должны проверить тип переменной каждого элемента в списке.*

Основная структура:

```
list components( entity entity )
```

Например, если вы хотите автоматизировать процесс расчёта профилей патронов для инструментов в группе, которая содержит траектории, границы и инструменты:

```
FOREACH ent IN components(entity('Group', '1')) {
    IF lcase(ent.RootType) == 'tool' {
    EDIT TOOL $ent.Name UPDATE_TOOLPATHS_PROFILE
  }
}
```

Например, чтобы обеспечить в NC файле для всех траекторий Выборки охлаждение Полив, а для всех остальных траекторий охлаждение Туман:

```
FOREACH item IN components(entity('ncprogram','')) {
  // Проверка траекторий в NC-файле
  IF lcase(item.RootType) == 'nctoolpath' {
    // Если активен параметр AreaClearance (Выборка), тогда
    использовать Полив
    IF active(entity('toolpath',item.Name).AreaClearance) {
      $item.Coolant.Value = "flood"
    } else {
      $item.Coolant.Value = "mist"
    }
  }
}
```

## Список элементов папки

Встроенная функция папки возвращает список всех элементов внутри папки включая подпапки.

Основная структура:

```
list folder( string folder )
```

Имена корневых папок:

- MachineTool (Станки)
- NCProgram (NC Файлы)
- Toolpath (Траектории)
- Tool (Инструменты)
- Boundary (Границы)
- Pattern (Шаблоны)
- Featureset (2D Модели)
- Workplane (СК Деталей)
- Level (Слои и Наборы)
- Model (Модели)
- StockModel (Модели Материала)
- Group (Группы)

 *Имя папок чувствительно к регистру, так что необходимо использовать **Tool** , а не **tool**.*

Можно использовать цикл **FOREACH**, чтобы обработать все элементы внутри папки.

Например, для пакетного вычисления профилей патронов всех инструментов:

```
FOREACH tool IN folder ('Tool') {
  EDIT TOOL $tool.Name UPDATE_TOOLPATHS_PROFILE
}
```

Например, для пакетного вычисления всех границ в проекте:

```
FOREACH bou IN folder('Boundary') {
  EDIT BOUNDARY $bou.Name CALCULATE
}
```

## Функции пути

Функция пути возвращает часть имени пути элемента.

Основная структура функций пути:

| Описание возвращаемого значения | Функция |
|---|---|
| Функция имени папки возвращает полное имя папки элемента или пустую строку, если элемент не существует. | `string pathname( entity ref )` |
| Функция имени папки, которая также может быть использована для вывода имени папки элемента. | `string pathname( string type, string name)` |

| | |
|---|---|
| Функция имени каталога возвращает префикс пути каталога. | `string dirname( string path)` |
| Функция базового имени возвращает суффикс, не являющийся директорией пути. | `string basename( string path)` |

## *Имя папки*

Функция имени папки возвращает полное имя папки элемента или пустую строку, если элемент не существует.

Основная структура:

```
string pathname( entity ref )
```

Или:

```
string pathname( string type, string name)
```

Возвращает полное имя папки элемента.

Например, если у вас есть инструмент **BN 16.0 diam** в папке **Ballnosed tools**, тогда:

```
STRING path = pathname('tool', 'BN 16.0 diam')
PRINT = path
```

выведет строку **Tool\Ballnosed tools\BN 16.0 diam**.

*Если элемент не существует, то будет выведена пустая строка.*

Эту функцию можно использовать совместно с функцией имени каталога для вычисления всех траекторий в той же папке, что и активная траектория.

```
STRING path = pathname('toolpath',Name)
// Проверка, есть ли активная траектория
IF path != '' {
  FOREACH tp IN folder(dirname(path)) {
    ACTIVATE TOOLPATH $tp.Name
    EDIT TOOLPATH ; CALCULATE
  }
} ELSE {
  MESSAGE info "Нет активной траектории"
}
  RETURN
```

### *Имя каталога*

Функция имени каталога возвращает префикс пути каталога.

Основная структура:

```
string dirname( string path)
```

Например, можно использовать её, чтобы получить аргумент для встроенной функции папки.

```
STRING flder = dirname(pathname('toolpath',Name))
```

### *Базовое имя*

Функция базового имени возвращает суффикс, не являющийся директорией пути.

Основная структура:

```
string basename( string path)
```

Обычно `basename(pathname('tool',tp.Name))` тоже самое что и `tp.Name`, но эту функцию можно использовать совместно с функцией имени каталога, чтобы разделить на части имена папок.

Предположим, что траектории расположены в папках:

Toolpath\Элемент1\Черновая

Toolpath\Элемент1\Доработка

Toolpath\Элемент1\Чистовая

Toolpath\Элемент2\Черновая

Toolpath\Элемент2\Доработка

Toolpath\Элемент2\Чистовая

Можно переименовать все траектории таким образом, чтобы они содержали имя элемента и индикатор (черновая, доработка или чистовая).

```
FOREACH tp in folder('Toolpath') {
  // Получение имени папки
  STRING path = pathname(tp)
  // Получение нижнего имени папки
  STRING type = basename(dirname(path))
  // Получение след. нижнего имени папки
  STRING feat = basename(dirname(dirname(path)))
  // Получение имени траектории
  STRING base = basename(path)
  // Создание нового имени траектории
  STRING NewNamePrefix = feat + "-" + type
  // Проверка,чтобы траектория не была уже переименована
  IF position(base,NewNamePrefix) < 0 {
    RENAME TOOLPATH $base ${NewNamePrefix+" " + base}
  }
}
```

## Условные функции

Основная структура условных функций:

| Описание возвращаемого значения | Функция |
|---|---|
| Возвращает значение выражение1, если условное выражение истинно, в другом случае выводит значение выражение2. | `variant select(условное выражение; выражение1; выражение2)` |

*Оба выражения должны быть одного типа.*

Следующий пример позволяет получить либо радиус инструмента, либо радиус его скругления, если оно есть.

Для этого можно использовать оператор IF:

```
REAL Radius = Tool.Diameter/2
  IF active(Tool.TipRadius) {
    $Radius = Tool.TipRadius
  }
PRINT = Radius
```

Или встроенную функцию ***select()***:

```
REAL Radius = select(active(Tool.TipRadius),
Tool.TipRadius, Tool.Diameter/2)
PRINT = Radius
```

*Если вы присваиваете параметру выражение, то вы должны всегда использовать встроенную функцию **select()**. Внутри макроса можно использовать любой метод.*

## Функции преобразования типа переменной

Функции преобразования типа позволяют временно конвертировать переменную из одного типа в другой внутри выражения.

Основная структура функций преобразования типа:

| Описание возвращаемого значения | Функция |
|---|---|
| Конвертирует в целое значение. | `int int( scalar a)` |
| Конвертирует в действительное значение. | `real real( scalar a)` |
| Конвертирует в булево значение. | `bool bool( scalar a)` |
| Конвертирует в строковое значение. | `string string( scalar a)` |

Для преобразования числа в строку обычно используется встроенная функция **string()**:

```
STRING $Bottles = string(Number) + " green bottles ..."
```

Для преобразования целого числа в действительное, или наоборот:

```
INT a = 2
INT b = 3
REAL z = 0
$z = int(a/b)
PRINT $z
```

Будет выведено **0.0**.

Если вы хотите вывести соотношение, тогда необходимо преобразовать или **a** или **b** в **действительное число** внутри выражения присваивания:

```
INT a = 2
INT b = 3
REAL z = 0
$z = real(a)/b
PRINT $z
```

Будет выведено **0.666667**.

## Функции параметров

Все параметры PowerMILL имеют активное состояние, которое определяет, является ли параметр соответствующим для конкретного типа объекта или операции.

Основная структура функций параметров:

| Описание возвращаемого значения | Функция |
|---|---|
| Вычисляет активное выражение **par**. | `bool active( par )` |
| Возвращает, может ли параметр быть изменен или нет. | `bool locked( par )` |
| Возвращает количество подпараметров, содержащихся в **par**. | `int size( par )` |

Например, параметр **Boundary.Tool** неактивен для границы по Заготовке или по Эскизу. Вы можете проверить, является ли параметр активным или нет, с помощью встроенной функции **active()**. Это может быть полезно при расчетах и принятии решений.

Основная структура:

```
IF active(….) {
  …
}
```

Вы можете проверить, является ли определенный параметр заблокированным или нет, с помощью встроенной функции **locked()**. Обычно вы не можете редактировать заблокированный параметр, потому что его элемент используется как ссылка другим элементом. Если вы попытаетесь редактировать заблокированный параметр с помощью команды **EDIT PAR**, PowerMILL вызовет диалог с запросом о разрешении внести изменения. Вы можете отключить это сообщение при помощи команды **EDIT PAR NOQUERY**. Функция блокировки **locked()** позволяет предоставить собственные пользовательские сообщения и запросы, которые соответствуют вашим задачам.

Например:

```
IF locked(Tool.Diameter) {
  BOOL copy = 0
  $copy = QUERY "Данный инструмент используется,
  хотите сделать его копию?"
  IF NOT copy {
    // Невозможно продолжить, выход.
    RETURN
  }
  COPY TOOL ;
}
$Tool.Diameter = 5
```

Встроенная функция **size()** возвращает количество элементов в параметре. Например, её можно использовать, чтобы определить количество траекторий в папке:

```
PRINT = size(folder('Toolpath\Cavity'))
```

## Статистические функции

Статистические функции позволяют выводить минимальные и максимальные значения любого количества числовых аргументов.

Основная структура статистических функций:

| Описание возвращаемого значения | Функция |
|---|---|
| Возвращает наибольшее значение в списке чисел. | `real max( list numeric a)` |
| Возвращает наименьшее значение из списка чисел. | `real min( list numeric a)` |

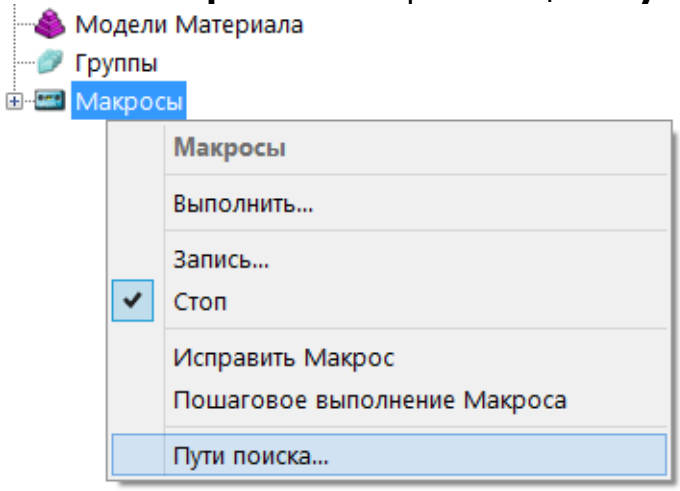
Следующий пример находит максимальную и минимальную высоту заготовки в траекториях в активном NC файле.

```
REAL maxz = -100000
REAL minz = abs(maxz)
FOREACH item IN components(entity('ncprogram','')) {
    IF item.RootType == 'nctoolpath' {
    $maxz = max(maxz,entity('toolpath',item.Name))
    $minz = min(minz,entity('toolpath',item.Name))
  }
}
MESSAGE info "Min = " + string(minz) + ", Max = " +
string(maxz)
```

# Организация макросов

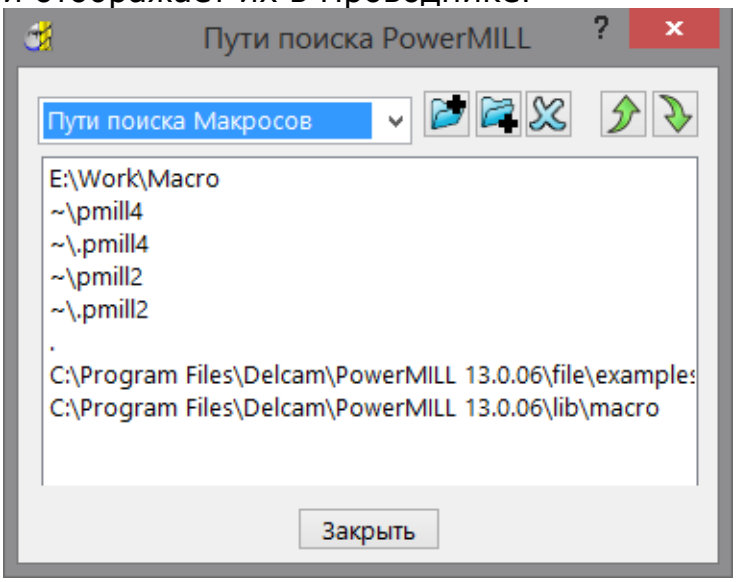
Записанные макросы отображаются в Проводнике в ветви **Макросы**. В этом примере показано, как управлять путями поиска макросов.

1 В меню **Макросов** выберите опцию **Пути поиска**.




*Или в меню **Инструменты** выберите **Пути поиска > Пути поиска макросов**.*

Откроется диалог **Пути поиска PowerMILL**, показывающий все пути поиска макросов по умолчанию. PowerMILL автоматически находит макросы, находящиеся в этих папках, и отображает их в Проводнике.




*Точка (.) показывает путь к локальной папке (текущая папка, в которой сохранен проект). Тильда (~) указывает на базовый каталог **Home**.*

2 Чтобы создать путь поиска макросов, нажмите на , и используйте диалог **Выберите путь** для выбора нужного пути. Путь добавляется в начало списка.

3 Чтобы изменить порядок поиска путей, выберите путь, который хотите изменить, и перемещайте его с помощью кнопок и .

4 Нажмите **Закрыть**.

5 Чтобы загрузить новые пути в PowerMILL, разверните в Проводнике ветвь **Макросы**.

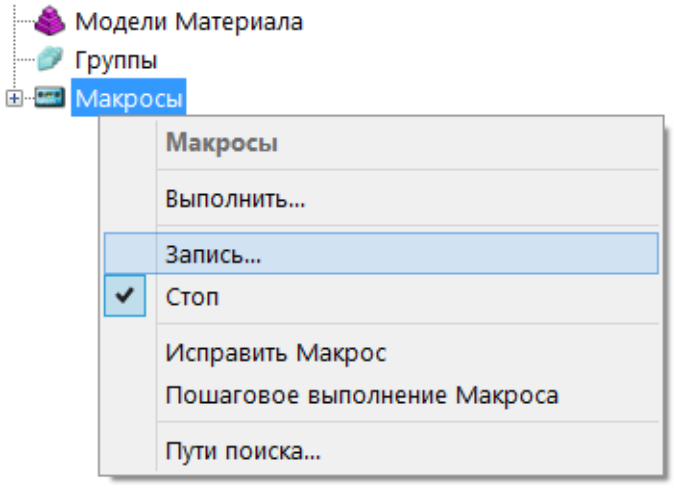
*Показываются только те пути, где есть хотя бы один макрос.*

# Запись макроса pmuser

Макрос **pmuser.mac** запускается автоматически, когда вы запускаете PowerMILL, выполняя ваши установки.

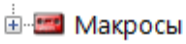
1 В меню **Инструменты** выберите **Сбросить диалоги**. Это гарантирует, что PowerMILL использует в диалогах параметры по умолчанию.

2 В контекстном меню **Макросы** выберите опцию **Запись**.




3 Выберите папку **pmill4** в каталоге **Home**. В диалоге **Выбор записываемого макроса** введите **pmuser** в поле **Имя файла** и нажмите **Сохранить**.

*На вопрос, хотите ли перезаписать существующий файл, ответьте **Да**.*

Иконка Макросы  изменит цвет на красный, показывая, что идет запись.

*Все опции в диалогах, которые вы хотите включить в макрос, должны быть выбраны во время записи. Если опция уже имеет нужное значение, введите его повторно.*

4 Задайте нужные параметры. Например:

a В меню **NC файлы** выберите **Параметры**.

b В диалоге **Параметры NC файлов** выберите **Постпроцессор** (например, **Heid400.pmopt**).

c Нажмите **Открыть**.

d Нажмите **Принять**.

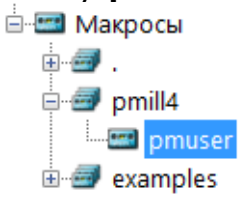
e Нажмите  на **Главной** панели инструментов, чтобы открыть диалог **Безопасные высоты**.

f Введите **Безопасную Z 10** и **Начальную Z 5**.

g Нажмите **Принять**.

5 В контекстном меню **Макросы** выберите пункт меню **Стоп**, чтобы завершить запись.

6 Разверните ветвь **Макросы**. Макрос **pmuser.mac** добавлен в папку **pmill4.**




7 Закройте и перезапустите PowerMILL, чтобы убедиться, что параметры из макроса **pmuser** активированы.

# Отключение сообщений об ошибках и предупреждениях и блокировка обновления экрана

## Сообщения об ошибках и предупреждениях

PowerMILL отображает ошибки и предупреждения, на которые необходимо ответить. Например, PowerMILL покажет сообщение с ошибкой, если вы попытаетесь активировать траекторию, которая не существует.

Обычно следует избегать написания макроса с ошибками или предупреждениями, но иногда это невозможно. В таких случаях, вы можете отключить эти сообщения следующим образом:

```
DIALOGS MESSAGE OFF
DIALOGS ERROR OFF
```

Чтобы вернуть назад отображение ошибок и предупреждений, наберите следующее:

```
DIALOGS MESSAGE ON
DIALOGS ERROR ON
```

## Графика

После запуска макроса PowerMILL обновляет экран каждый раз, как сделано изменение. Если PowerMILL обновляет экран очень часто, такое количество обновлений выглядит довольно неприятно. Чтобы PowerMILL не обновлял экран во время выполнения макроса, добавьте команду в начало макроса:

```
GRAPHICS LOCK
```

и затем, чтобы обновить экран только один раз после выполнения всех команд, добавьте в конце макроса:

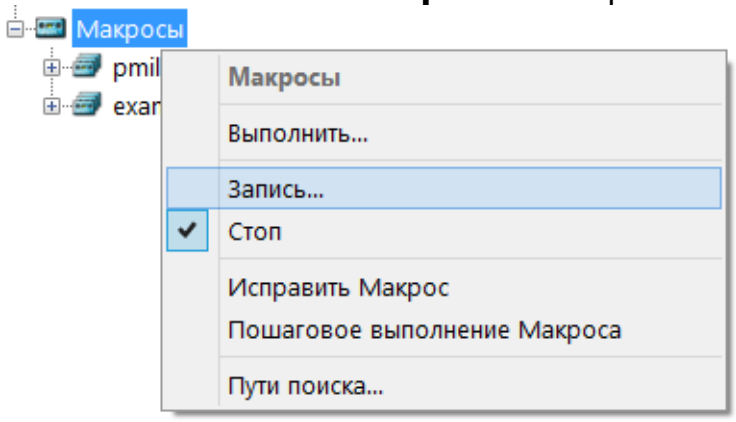
```
GRAPHICS UNLOCK
```

*Когда макрос завершится, PowerMILL восстановит настройки графики и сообщений в то состояние, в котором они были до запуска макроса. Не забывайте добавлять обратные команды, чтобы графика и сообщения не были случайно отключены навсегда.*

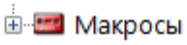
# Запись макроса для установки параметров NC файлов

Этот пример показывает как записать макрос, который задает параметры NC файла для контроллеров Heid400.

1. В меню **Инструменты** выберите **Сбросить диалоги**. Это гарантирует, что PowerMILL будет использовать в диалогах параметры по умолчанию.
2. В контекстном меню **Макросы** выберите **Запись**.




3. Выберите папку **pmill4** в каталоге **Home**. В диалоге **Выбор записываемого макроса** введите **Heid400_prefs** в поле **Имя файла** и нажмите **Сохранить**.

   Иконка Макросы 

изменит цвет на красный, показывая, что идет запись.

   *Все опции в диалогах, которые вы хотите включить в макрос, должны быть выбраны во время записи. Если опция уже имеет нужное значение, введите его повторно.*

4. В контекстном меню **NC файлы** выберите **Параметры**.
5. В диалоге **Параметры NC файлов** выберите **Heid400.pmopt** в поле **Постпроцессор** на вкладке **Вывод**.
6. Нажмите на вкладку **Траектория** и выберите **Всегда** в поле **Смена Инструмента**.
7. Нажмите **Принять**.
8. В контекстном меню **Макросы** выберите пункт меню **Стоп**, чтобы завершить запись.

# Советы при создании макросов

В этом разделе приведены советы, которые помогут при создании макросов.

- Макрос записывает любые значения, которые вы явным образом изменяете в диалоге, но не записывает текущие значения по умолчанию. Например, если допуск по умолчанию составляет **0.1** мм, и вы хотите допуск **0.1** мм, то необходимо повторно ввести **0.1** в поле допуска во время записи макроса. В противном случае PowerMILL будет использовать любое текущее значение допуска, которое не обязательно будет тем значением, что нужно.
- В меню **Инструменты** выберите **Сбросить диалоги**. Это гарантирует, что PowerMILL будет использовать в диалоговых окнах параметры по умолчанию.
- При отладке макроса важно выключить MacroFixer. Для этого используйте следующую команду:

  ```
  UNSET MACROFIX
  ```

  Это обеспечит, что все синтаксические и макро ошибки будут передаваться в PowerMILL напрямую. Вы можете использовать команду `SET MACROFIX` чтобы включить обратно MacroFixer.
- Если вы получаете синтаксическую ошибку в циклах (**DO-WHILE**, **WHILE**, **FOREACH**) или в условных операторах (**IF-ELSEIF-ELSE**, **SWITCH**) проверьте, имеется ли пробел перед каждой открывающей фигурной скобкой (`{`). Для цикла **DO-WHILE** убедитесь, что закрывающая фигурная скобка (`}`) имеет пробел после неё и перед словом **WHILE**.
- Блоки кода должны располагаться в фигурных скобках. Они должны иметь одинаковое количество открывающих (`{`) и закрывающих (`}`) фигурных скобок.
- В слове **ELSEIF** не должно быть пробела между **ELSE** и **IF**.
- Если вы столкнулись с ошибками выражения, проверьте, чтобы круглые скобки и кавычки имели одинаковое количество (открывающие и закрыывающие).
- В качестве десятичного разделителя в числах должна использоваться точка (**.**), а не запятая (**,**).
- В присваиваниях, переменная в левой части знака **=** должна иметь префикс `$`. Таким образом:

  ```
  $myvar = 5
  ```

  переменная `myvar` задана правильно, а:

  ```
  myvar = 5
  ```

  неправильно, так как отсутствует префикс `$`.

- Локальные переменные подменяют параметры PowerMILL. Если макрос содержит:

  ```
  REAL Stepover = 10
  ```

  то во время выполнения макроса любое использование **Stepover** будет использовать значение **10**, независимо от того, какое значение задано в пользовательском интерфейсе. Также как и команда:

  ```
  EDIT PAR "Stepover" "Tool.Diameter*0.6"
  ```

  изменит значение локальной переменной **Stepover** , а не значение параметра PowerMILL **Stepover**.

# Частые вопросы

### Как создать цикл для всех траекторий ?

Функция папки folder() возвращает все элементы в папках Проводника. Например **Machine Tools**, **Toolpaths**, **Tools**, **Boundaries** и тд.. Самый простой способ перебора всех элементов заключается в использовании оператора FOREACH:

```
FOREACH tp IN folder('Toolpath') {
  PRINT = tp.name
}
```

Функция папки возвращает все элементы в заданной папке и во всех подпапках. Можно ограничить количество выводимых элементов, указав конкретную подпапку для цикла:

```
FOREACH tp IN folder('Toolpath\semi-finishing') {
  PRINT = tp.name
}
```

### Как создать цикл только для элементов в основной папке (исключая любые подпапки)?

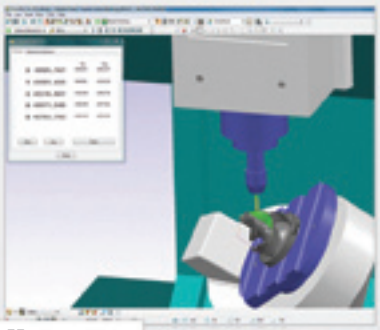
Как написано выше, функция папки folder() возвращает все элементы из родительской папки и всех подпапок. Если вы хотите перебрать все элементы только в родительской папке, необходимо отфильтровать результаты. Например, если у вас есть папка 'Semi-finishing' и подпапка 'Not used', которая содержит временные или альтернативные траектории, то чтобы получить доступ только к траекториям в папке 'Semi-finishing', необходимо использовать функции pathname и dirname:

```
STRING fld = 'Toolpath\semi-finishing'
FOREACH tp IN folder($fld) {
  IF dirname(pathname(tp)) == $fld {
    PRINT = tp.name
  }
}
```

Перевод на русский язык – **Eksodus**

# PowerMILL 2012

PowerMILL 2012 - Что нового



PowerMILL 2012 - Что нового

Delcam TV

www.delcam.tv

PowerMILL Learning Zone


PowerMILL Learning Zone

PowerMILL Website

www.powermill.com

THE QUEEN'S AWARDS FOR ENTERPRISE: INNOVATION 2010

THE QUEEN'S AWARDS FOR ENTERPRISE: INTERNATIONAL TRADE 2011

## Powering your productivity

Delcam Headquarters | Small Heath Business Park | Birmingham | B10 0HJ | UK
+44 (0)121 766 5544 | marketing@delcam.com | www.delcam.com

To contact your local reseller, visit **www.delcam.com/resellers**