Справочник по ActionScript

Данная глава не похожа на остальные главы этой книги, но мы надеемся, что и она окажется полезной. Вместо пошагового описания инструкций, применявшегося ранее в этой книге, мы организовали материал так, чтобы вы могли получать информацию по мере необходимости. Например, если существует объект, синтаксис которого вы хотели бы уточнить, или вы хотите что-либо сделать, но не знаете, какие элементы ActionScript лучше для этого использовать, — в этих и многих других случаях вам поможет данный справочник.

В разделах этой главы описываются следующие элементы языка ActionScript.

Встроенные (глобальные) функции. Встроенные команды ActionScript. Используются для навигации и управления временной шкалой.

Свойства. Элементы, которые устанавливают и извлекают параметры временных шкал, кнопок и клипов.

Операторы. Применяются для вычисления и изменения значений выражений.

Объектные ссылки и ссылки на временную шкалу. Средства, позволяющие получить доступ к элементам фильма.

Предопределенные объекты. Применяются для манипулирования данными, связанными со встроенными объектами (MovieClip, Sound и т. п.).

Благодаря хорошо продуманной структуре главы у вас не должно возникнуть трудностей, связанных с поиском. Подумайте, что вы хотите сделать. Если необходимо изменить вид чего-либо, тогда вам, скорее всего, нужен раздел "Свойства". Если нужно установить контроль над воспроизведением фильма главной временной шкалы, обратитесь к разделу "Встроенные функции". Может случиться, что вы, наоборот, точно знаете термин, но не помните, к какой категории он относится. Такая проблема тоже легко разрешается. Если вы, например, хотите найти функцию `startDrag()`, где вы начнете поиск? Подумайте, при помощи функции `startDrag()` вы можете перемещать клип. Это — разновидность управления временной шкалой, следовательно, вам нужен раздел "Функции". В самих разделах термины приводятся в алфавитном порядке.

Справа от каждого элемента в скобках указана версия Flash, начиная с которой его можно использовать.

Этот справочник по ActionScript не полон. До выхода Flash MX 2004 ActionScript уже был довольно сложным языком, а с появлением всех нововведений этой программы он стал еще сложнее. Полный справочник по этому языку, наверное, имел бы больший объем, чем вся эта книга. Здесь же вы найдете руководство по применению элементов ActionScript, которые, по нашему мнению, являются *самыми важными*. Без них вам не обойтись.

Встроенные (глобальные) функции

Втроенные (или глобальные) функции представляют собой встроенные команды ActionScript. С помощью функций можно создавать условные операторы и циклы, а также управлять воспроизведением Flash-фильмов.

break (4+)

Синтаксис

```
break
```

Эта функция может оказаться полезной, если вы хотите выйти из цикла до его завершения. Функция `break` выполняется внутри цикла, цикл разрывается и управление передается первому следующему за ним выражению.

Пример

```
onClipEvent(load){
   taxRate=.0875;
   grandTotal=subTotal*taxRate;
   for(i=1;i<<10;i++){
      if(eval("_root.selQty"+i)==0){
         this.attachMovie("alertClip","alert",1);
         break;
      }
      totalQty+=eval("_root.selQty"+i);
   }
}
```

В данном сценарии фукнция `break` используется для разрыва цикла и создается новый экземпляр клипа `alertClip`, если одна из переменных `selQty1`, `selQty2`...`selQty9` равна нулю. На практике экземпляр клипа `alertClip` представляет собой своего рода предупреждающее окно с сообщением, информирующим аудиторию о возникновении ошибки ввода данных и о том, как с ней можно справиться. Функция `break` размещается в сценарии так, чтобы выполнение последнего остановилось перед выполнением выражения, которое обновило бы значение переменной `totalQty`. Кроме того, функция `break` вызывается *после* добавления клипа `alert`. См. также описание функции `continue`.

clearInterval (6+)

Синтаксис

```
clearInterval(идентификатор_интервала)
```

Эта функция прекращает выполнение выражений для интервала, созданного с помощью функции `setInterval()`, которому был присвоен идентификатор интервала. Пример использования функции `clearInterval()` вы найдете в описании функции `setInterval()`.

continue (4+)

Синтаксис

```
continue
```

Эта функция прерывает выполнение текущей итерации цикла. Затем для операторов `while` или `do...while` условие цикла проверяется до его возобновления. Для оператора `for` сначала увеличивается счетчик цикла, а затем проверяется условие цикла. В конструкции `for...in` (см. ниже) выполнение цикла возобновляется для следующего свойства. Во избежание ошибок при использовании функции `continue` внимательно следите за выполнением оператора цикла.

Пример

```
onClipEvent(load){
   taxRate=.0875;
   grandTotal=subTotal*taxRate;
   for(i=1;i<<10;i++){
      if(eval("_root.selQty"+i)==0){
          continue;
      }
      totalQty+=eval("_root.selQty"+i);
   }
}
```

В этом сценарии функция `continue` используется для прерывания текущей итерации цикла, если одна из переменных `selQty1`, `selQty2`...`selQty9` равна нулю. См. также описание функции `break`.

do...while (4+)

Синтаксис

```
do{
   выражение;
} while(условие);
```

В этом цикле истинность условия проверяется после выполнения очередной итерации. Другими словами, тело цикла будет гарантированно выполнено один раз.

Пример

```
X=0;
trace("целые числа от 0 до 9:");
do{
   trace(x);
   x++
}while(x<<10);
```

Данный сценарий выводит на экран фразу `целые числа от 0 до 9:`, за которой следуют числа `0 1 2 3 4 5 6 7 8 9`. При выполнении цикла в одиннадцатый раз значение `x` становится равным 10. Условие `x<<10` уже не является истинным, и выполнение цикла прекращается.

duplicateMovieClip (4+)

Синтаксис

```
duplicateMovieClip("источник","копия_экземпляра",уровень)
```

При помощи функции `duplicateMovieClip()` создается копия экземпляра клипа. Функция принимает три аргумента: *источник* — имя экземпляра, с которого следует создать копию (строка); *копия_экземпляра* — имя, которое присваивается копии экземпляра (тоже строка); *уровень* — уровень главного фильма, на который следует поместить полученную копию. Аргумент *уровень* используется не для того, чтобы присвоить новому клипу свойство фильма `_level`, а чтобы установить уровень копии по отношению к клипу *источник*. Если, например, клип *источник* находится на уровне 1, его копию можно поместить ниже на уровень 0 или выше на уровень 2. Клипы, находящиеся на уровнях, номер которых больше, будут перекрывать клипы с меньшим номером уровня, то есть появляться поверх них. Копии клипов наследуют свойства оригиналов, но не переменные из временной шкалы клипа *источника*. Чтобы удалить копию клипа, воспользуйтесь функцией `removeMovieClip()`.

eval (5+)

Синтаксис

```
eval(выражение)
```

Когда выражение является строкой или возарвщает строку, функция `eval()` преобразует его в идентификатор с переменным значением, значением свойства или ссылкой на клип, содержащейся в строке. Если выражение не содержит информации, функция возарщается без идентификатора.

Пример

Функция применяется для управления динамически генерируемыми переменными или экземплярами:

```
j=1;
while(j<=5){
   this.duplicateMovieClip("cloud "+j,j);
   eval("_root.cloud "+j)._x=100*j;
   eval("_root.cloud "+j)._y=100*j;
   eval("_root.cloud "+j)._alpha=20*j;
   j++;
}
```

В этом примере создается пять копий клипа. Имя каждого нового экземляра (`cloud1`, `cloud2` и т. д.) возвращается как строка и преобразуется в ссылку клипа, которую можно использовать.

for (5+)

Синтаксис

```
for(инициализирующее_выражение;условное_выражение;модифицирующее_выражение){
   выражение;
}
```

Функция `for` предлагает альтернативный вариант цикла `while`. Вместо двух дополнительных строк, в которых инициализируется переменная и увеличивается счетчик цикла, в цикле `for` используется одна-единственная строка, в которую помещаются все элементы цикла.

При обнаружении в сценарии цикла `for` первым выполняется *инициализирующее_выражение*, в котором обычно устанавливается счетчик цикла. Это происходит только один раз перед запуском цикла. Затем анализируется *условное_выражение*, содержащее условие прекращения цикла. Пока оно равно `true`, цикл не прекращается. Каждый раз после прохождения всех строк тела цикла выполняется *модифицирующее_выражение*, в котором происходит изменение счетчика цикла. Как только проверка *условного_выражения* даст результат `false`, все строки тела цикла и *модифицирующее_выражение* будут пропущены и управление будет передано первому выражению, следующему за телом цикла.

Если нужно, чтобы в цикле было несколько переменных (то есть более одного счетчика), разделите их запятыми.

Пример

Два нижеприведенных цикла эквивалентны. Каждый из них выполняется до тех пор, пока переменные не станут равны 5. Обратите внимание на то, насколько компактнее синтаксис цикла `for`.

```
for(hi=10,lo=10;hi!=lo;hi--,lo++){
   trace("hi: "+hi);
   trace("lo: "+lo);
}

hi=10;
lo=0;
while(hi!=lo){
   trace("hi: "+hi);
   trace("lo: "+lo);
   hi--;
   lo++;
}
```

for...in (5+)

Синтаксис

```
for(переменная in объект){
   выражение;
}
```

Цикл `for…in` выполняет *выражение* в отношении ряда свойств, принадлежащих объекту или массиву. Цикл выполняет *выражение* для каждого свойства (сохраненного в *переменной*), относяще-

гося к указанному *объекту*. Этот цикл можно использовать для того, чтобы установить или обновить все свойства объекта либо извлечь из них необходимую информацию.

Пример

```
for(item in _level0){
   clip=_root[item];
   if(typeof clip=="movieClip"}{
      trace("Путь: "+targetPath(clip));
      //Действия для подготовки клипов и т. д.
   }
}
```

Данный сценарий проверяет все элементы временной шкалы объекта. Если элемент является клипом, путь к нему выводится в окне **Output** (Выходные данные). Кроме того, сценарий можно использовать как шаблон для выполнения некоторых других действий с клипами, например для того, чтобы установить координаты клипа, сделать его видимым или невидимым и т. д.

fscommand (3+)

Синтаксис

```
fscommand("аргумент","значение")
```

Функцию `fscommand()` можно использовать для установки связи с приложением, служащим хостом для вашего Flash-фильма. С ее помощью обеспечивается взаимодействие между фильмом и автономным приложением Flash Player. Дополнительную информацию об использовании функции `fscommand()` в ваших проектах вы найдете в главе 32. Эта функция также может обеспечивать связь между фильмом и веб-браузером при помощи команд JavaScript или VBScript.

function (5+)

Синтаксис

```
function имя(аргумент1,аргумент2…аргументX){
   выражение;
}
```

При помощи функции `function` создается пользовательская функция. Функция — это подпрограмма обработки данных, которую можно выполнять много раз. Функции применяются для создания стандартных процедур, выполняющих часто встречающие задачи. Чтобы создать функцию, необходимо указать ее *имя* и функции, которые она будет выполнять (*выражение*). В случае необходимости вы также можете указать аргументы для передачи функции каких-либо параметров.

Пример

Простая функция не требует аргументов. Следующая функция перемещает головку воспроизведения основной временной шкалы к первому кадру.

```
function rewind(){
   gotoAndStop(1);
}
```

Для более сложных функций необходимо указывать аргументы, такие функции более гибки и предоставляют больше возможностей разработчику.

Нижеприведенная функция сперва проверяет, указаны ли все необходимые аргументы. Если это не было сделано, в окне **Output** (Выходные данные) появляется соответствующее сообщение и выполнение функции прекращается. Если аргументы указаны правильно, функция возвращает результат.

```
function times (op1, op2){
   if(op1==undefined || op2==undefined){
      trace("Требуются аргументы");
      return;
   }else{
      return op1*op2;
   }
}
```

При втором вызове в вышеприведенном примере функция `return` возвращает значение. Чтобы функция возвратила результат, который можно использовать, присвойте ее переменной, как это сделано в следующем выражении:

```
prod=_root.times(2,2);
```

При выполнении функции из предыдущего примера переменная `prod` будет иметь значение 4.

getTimer (4+)

Синтаксис

```
getTimer()
```

Функция `getTimer()` возвращает число миллисекунд, прошедших с начала вопроизведния фильма. Она может применяться для назначения на время событий или вызова функций.

Пример

```
onClipEvent(load){
   start=getTimer();
}
onClipEvent(enterFrame){
   if(start+10000<getTimer()){
      orb.gotoAndStop(orb._currentframe+1);
      start=getTimer();
   }
}
```

Это сценарий дает хороший пример простого таймера. В переменной начинается запись миллисекунд, когда клип загружается впервые. В каждом кадре текущее число миллисекунд проверяется и сравнивается с исходным значением времени плюс 10 000. Если текущее число миллисекунд больше чем исходного + 10 000, значит, фильм воспроизводится в течение 10 секунд, тогда воспроизведение клипа `orb` переходит к следующему кадру и прекращается, а значение счетчика сбрасывается. Через 10 секунд воспроизведение клипа `orb` перейдет к следующему кадру, значение счетчика будет сброшено и т. д.

newline (4+)

Синтаксис

```
newline
```

Эта функция обеспечивает вставку знака разрыва строки (Возврата каретки) в троке текста. Функция `newline` применяется для вставки знака разрыва строки в текстовых полях.

getURL (2+/4+)

Синтаксис

```
getURL("имя_файла","окно","метод_передачи")
```

Функция `getURL()` позволяет Flash работать за пределами фильма. С ее помощью можно:

открыть документ, находящийся по указанному URL в сети Internet;

передать данные на сервер;

установить связь с фильмом программы Director.

Функция `getURL()` принимает до трех аргументов. Аргумент *`имя_файла`* указывает файл, который должен быть открыт, или команду, которая должна быть передана. Аргумент `окно` позволяет обратиться к определенному окну в браузере или фрейму. Все значения этого аргумента указываются как строки.

_self — документ загружается в текущем окне, где в настоящий момент воспроизводится фильм;
_blank — документ загружается в новом окне;
_parent — данное значение указывает на текущее окно и позволяет загрузить документ во фрейм, в котором в настоящий момент воспроизводится фильм;
_top — данный элемент целесообразно использовать тогда, когда ваш фильм воспроизводится во фрейме, а новый документ, на который указывает URL, должен заполнить все окно браузера.

Третий аргумент *`метод_передачи`* (который поддерживают только четвертая и более поздние версии Flash Player) позволяет передавать информацию приложению на сервер при помощи методов `GET` и `POST`.

gotoAndPlay (2+)

Синтаксис

```
gotoAndPlay(номер_кадра)
gotoAndPlay("метка_кадра")
gotoAndPlay("имя_сцены",номер_кадра)
gotoAndPlay("имя_сцены","метка_кадра")
```

Эта функция позволяет переместиться к определенному кадру, сцене или метке кадра и возобновить воспроизведение фильма с этого места. Аргумент *`номер_кадра`* задается как целое число, а аргументы *`метка_кадра`* и *`имя_сцены`* — как строки (то есть в кавычках). Если аргумент *`имя_сцены`* не указан, функция выполнится по отношению к текущей сцене.

gotoAndStop (2+)

Синтаксис

```
gotoAndStop(номер_кадра)
gotoAndStop("метка_кадра")
gotoAndStop("имя_сцены",номер_кадра)
gotoAndStop("имя_сцены","метка_кадра")
```

Эта функция позволяет переместиться к определенному кадру, сцене или метке кадра и остановить воспроизведение фильма в этом месте. Аргумент *`номер_кадра`* задается как целое число, а аргументы *`метка_кадра`* и *`имя_сцены`* — как строки (то есть в кавычках). Если аргумент *`имя_сцены`* не указан, функция выполнится по отношению к текущей сцене.

if (4+)

Синтаксис

```
if(условие){
   выражение;
}
```

Функция `if()` используется для создания условного оператора. *`Условие`* является выражением, возвращающим значение типа `boolean`. Если *`условие`* истинно, выполняется *`выражение`*. В противном случае *`выражение`* игнорируется. Функцию `if()` можно использовать в сочетании с функцией `else`, позволяющей указать выражения, которые будут выполняться, когда *`условие`* равно `false`.

loadMovie (3+)

Синтаксис

```
loadMovie("URL","уровень","метод_передачи")
```

Функция `loadMovie` применяется для загрузки дополнительных SWF-файлов. Аргументы этой функции всегда указываются как строки (то есть в кавычках). Аргумент `URL` позволяет ввести полный либо относительный путь к файлу, который вы хотите загрузить. В аргументе *`уровень`* вы можете указать, куда следует загрузить фильм. Фильм можно загрузить как новый документ (уровень `_level1`, `_level4` и т. д.) или заменить им фильм на одном из уже существующих уровней. Например, фильм основной временной шкалы можно заменить, загрузив новый фильм на уровнях `_level0` или `_root`. В аргументе *`уровень`* также можно указать имя клипа, в который нужно вставить новый SWF-файл. При загрузке фильма можно также передавать ему переменные. В аргументе *`метод_передачи`* укажите метод `GET` или `POST`, чтобы передать переменные в новый фильм. Метод `GET` рекомендуется использовать при загрузке небольшого количества переменных, в противном случае лучше воспользоваться методом `POST`.

loadMovieNum (4+)

Синтаксис

```
loadMovieNum("URL",уровень,"метод_передачи")
```

Функция `loadMovieNum()` почти идентична функции `loadMovie()`. Основное различие между ними состоит в том, что в первом случае аргумент *`уровень`*, определяющий уровень загрузки документа, задается как число, а не как строка. Это облегчает динамическую загрузку фильмов, но делает невозможной загрузку новых фильмов в экземпляры клипов. Если указанный уровень не существует, то программа создаст его для нового фильма. Если такой уровень все-таки есть, фильм, который находится на нем, будет заменен.

Пример

Два следующих выражения эквивалентны.

```
loadMovieNum("act1.swf",1,"GET");
loadMovie("act1.swf","_level1","GET");
```

В следующем цикле перебираются все уровни с фильмами от 0 до 8 и осуществляется проверка, все ли они существуют. В переменной `top` сохраняется номер последнего уровня, на котором имеется какой-то фильм. По завершении цикла на следующий уровень загружается файл `high.swf`.

```
l=0;
top=0;
do{
   if(eval("_level"+l(){
      trace("Найдено@"=l);
      top++;
   } else {
      trace("Не найдено@"+l);
   }
} while(l++<<8);
trace("Верхний уровень"+(top-1));
loadMovieNum("high.swf",top);
```

Если бы во время выполнения сценария на уровнях 0 и 1 были фильмы, файл `high.swf` загрузился бы на уровень `_level2`.

loadVariables (4+)

Синтаксис

```
loadVariables("URL","уровень","метод_передачи")
```

Функция `loadVariables()` позволяет загрузить переменные из файла, указанного в аргументе *URL*, в фильм или клип, определенный в аргументе *уровень* (следует указать либо уровень клипа, например `_level2`, либо его имя). Flash может считывать различные данные из текстовых документов, файлов CGI, ASP, PHP и других серверных файлов. Все переменные передаются как строки и должны находиться в том же домене, что и фильм, который запрашивает их при вызове его в веб-браузере. В аргументе *метод_передачи* укажите метод `GET` или `POST`, чтобы передать переменные в фильм-получатель. Метод `GET` рекомендуется использовать при загрузке небольшого количества переменных, в противном случае лучше воспользоваться методом `POST`.

loadVariablesNum (4+)

Синтаксис

```
loadVariablesNum("URL",уровень,"метод_передачи")
```

Функции `loadVariables()` и `loadVariablesNum()` практически идентичны. Основное различие между ними состоит в том, что для функции `loadVariablesNum()` аргумент *уровень*, определяющий уровень загрузки документа, задается как число, а не как строковое значение. Это облегчает динамическую загрузку фильмов, принимающих переменные. Однако функция `loadVariablesNum()` не позволяет передавать переменные в клипы.

nextFrame (2+)

Синтаксис

```
nextFrame()
```

С помощью этой функции головка воспроизведения перемещается на один кадр вперед и останавливает в новой позиции. Эта функция является альтернативой использованию выражения `gotoAndStop(_currentframe+1)`.

play (2+)

Синтаксис

```
play()
```

Функция `play()` выполняет одну из наиболее важных задач: с ее помощью воспроизводится анимация. Воспроизведение продолжается до тех пор, пока его не остановит другая функция или пока не закончатся кадры на временной шкале.

prevFrame (2+)

Синтаксис

```
prevFrame()
```

С помощью данной функции головка воспроизведения перемещается на один кадр назад и останавливается. Эта функция является альтернативой использованию выражения `gotoAndStop(_currentframe-1)`.

print (5+)

Синтаксис

```
print("имя_клипа","печатаемая_область")
```

Эта функция применяется для вывода на печать всего фильма или его части. В аргументе *имя_клипа* вы указываете имя клипа или фильма, кадры которого хотите напечатать. Чтобы определить, какие именно кадры должны быть напечатаны, пометьте каждый такой кадр меткой `#p` на панели **Properties** (Свойства). Если этого не сделать, будут напечатаны все кадры клипа или фильма, указанного в аргументе *имя_клипа*. Принтеры PostScript способны работать как с векторными, так и с растровыми изображениями. Все остальные принтеры перед печатью преобразуют векторные изображения в растровые. Функция `print()` не позволяет напечатать элементы, отображаемые с использованием альфа-канала, и отобразить цветовые эффекты — для этого воспользуйтесь функцией `printAsBitmap()`.

Аргумент *печатаемая_область* может принимать три следующих строковых значения.

"bmovie". Flash использует содержимое помеченного кадра для определения печатаемой область. Чтобы пометить кадр, определяющий печатаемую область, укажите для него метку #b и поместите в него графические элементы, которые будут служить в качестве обрамляющей фигуры, границы которой соответствуют печатаемой области.

"bmax". Применяет комбинацию содержимого всех печатаемых кадров, чтобы определить печатаемую область. Этим значением можно воспользоваться, если данные, выводимые на печать, создаются динамически и отличаются от кадра к кадру.

"bframe". Обрабатывается каждый печатаемый кадр отдельно. Изображение будет заполнять всю страницу, поэтому масштаб элементов каждого кадра будет соответственно изменяться.

Пример

В этом сценарии на печать выводятся все кадры экземпляра клипа `directions`. Печатаемые области в них определяются кадром с меткой `#b`.

```
print("_root.directions","bmoive");
```

Если вы хотите напечатать отдельные кадры, укажите для них метку `#p`.

printAsBitmap (5+)

Синтаксис

```
printAsBitmap("имя_клипа","печатаемая_область")
```

Эта функция подобна функции `print()`, но за одним исключением. С ее помощью вместо векторных печатаются растровые изображения. Несмотря на невысокое качество печати, именно эту функцию следует использовать, если вы хотите сохранить при печати эффекты прозрачности (напечатать элементы, для которых указаны значения альфа-канала) и цветовые эффекты. Характеристику аргументов функции см. в описании функции `print()`.

printAsBitmapNum (5+)

Синтаксис

```
printAsBitmapNum(уровень,"печатаемая_область")
```

Данная функция практически идентична функции `printAsBitmap()`, только в аргументе *уровень* вместо строкового значения вы указываете номер уровня. Эта функция используется, если необходимо вывести на печать фильм, находящийся на некотором уровне. Вместо того чтобы указывать номер уровня как целое число, вы можете воспользоваться переменной, принимающей числовые значения.

Пример

В этом примере номер уровня задается как переменная `printNfo`, содержащая числовые данные, возвращаемые функцией `output()`.

```
printNfo=output();
printAsBitmapNum(printNfo,"bframe");
```

printNum (5+)

Синтаксис

```
printNum(уровень,"печатаемая_область")
```

Эта функция практически идентична функции `print()`, только в аргументе *уровень* вместо строкового значения вы указываете номер уровня. Данная функция используется, если необходимо вывести на печать фильм, находящийся на некотором уровне. Вместо того чтобы указывать номер уровня как целое число, вы можете воспользоваться переменной, принимающей числовые значения.

Пример

Два следующих выражения эквивалентны.

```
printNum(3,"bmax");
print("_level3","bmax");
```

removeMovieClip (4+)

Синтаксис

```
removeMovieClip("имя_клипа")
```

Функция `removeMovieClip()` используется для удаления экземпляра клипа из Flash-фильма. В аргументе *имя_клипа* укажите имя экземпляра клипа, который вы хотите удалить, или путь к нему. Эта функция выполняется только для клипов, которые вы поместили в фильм при помощи функции `attachMovie()` или `duplicateMovieClip()`.

Пример

Если этот сценарий связать с клипом, созданным посредством функции `duplicateMovieClip()`, то экземпляр этого клипа при щелчке на нем мышью будет удален. Поскольку `this` относится к текущему клипу, его имя указано *не* как строковое значение.

```
onClipEvent(mouseUp){
   if(this.hitTest(_roor._xmouse,_root._ymouse,true){
      removeMovieClip(this);
   }
}
```

setInterval (6+)

Синтаксис

```
setInterval(функция,интервал,параметр1,параметр2,...,параметрX)
```

Эта встроенная функция выполняет другие функции через регулярные интервалы. В аргументе *функция* указывается, которая будет выполняться, в аргументе *интервал* — значение в миллисекундах. Интервал определяет, с какой частотой будет выполняться функция. Если функции требуются дополнительные параметры, их можно указать после аргумента *интервал*.

Пример

```
function annoy(){
   trace("Это сделано?");
}
setInterval(annoy,500);
```

Функция `annoy` выводит фразу на панели **Output** (Выходные данные). Функция `setInterval()` используется для выполнения функции `annoy` каждые пол секунд (500 миллисекунд). Функция `annoy` никогда не перестанет выполняться, поскольку интервал никогда не будет удален.

В следующем примере пример показано, как можно удалить интервал. Идентификатор интервала `countoff` создает интервал, который будет выполнять функцию `tick` каждую секунду (1000 милисекунд). Когда функция `tick` будет выполнена четыре раза, интервал `countoff` будет удален с помощью функции `clearInterval()`.

Когда идентификатор интервала удаляется, сам интервал прекращает существование.

```
clock =new Sound();
clock.attachSound("click1 ");
var count=1;
function tick(sample){
   sample.start();
   if (count==4){
      clearInterval(countoff);
   }
   count++;
}
countoff=setInterval(tick,1000,clock);
```

Обратите внимание, что аргумент функции `tick`, указан как третий параметр функции `setInterval()`.

startDrag (4+)

Синтаксис

```
startDrag("имя_клипа",блокирование,влево,вверх,вправо,вниз)
```

Функция `startDrag()` делает клип, указанный в аргументе *имя_клипа*, *перемещаемым*. То есть этот экземпляр клипа можно перемещать на области действия, щелкнув на нем мышью и перетаскивая его. Если аргумент *блокирование* имеет значение `true`, указатель блокируется на точке регистрации перемещаемого клипа. Если аргумент имеет значение `false`, можно перемещать клип, предварительно поместив указатель в любою точку на нем. Это необязательный аргумент. Если вы его не укажете, по умолчанию будет использоваться значение `false`. Аргументы *влево*, *вверх*, *вправо*, *вниз* определяют координаты границ перемещения клипа. Эти координаты определяются относительно временной шкалы, на которой размещается клип, указанный в аргументе *имя_клипа*. Коор-

динаты определяются относительно начала (0;0) временной шкалы, на которой размещается клип. Если клип относится к основной временной шкале, началом будет левый верхний угол области действия. Если клип находится на экземпляре другого клипа, началом будет служить точка регистрации этого другого клипа. Аргументы *влево*, *вверх*, *вправо*, *вниз* тоже необязательные. Чтобы отменить возможность перемещения клипа, воспользуйтесь функцией `stopDrag()`.

Пример

См. описание функции `stopDrag()`.

stop (2+)

Синтаксис

```
stop()
имя_клипа.stop()
```

При вызове функции `stop()` воспроизведение фильма останавливается. Чтобы остановить воспроизведение указанного клипа, введите ссылку на него в аргументе *имя_клипа*.

stopDrag (4+)

Синтаксис

```
stopDrag()
```

Функция `stopDrag()` прекращает выполнение функции `startDrag()` в фильме. Поскольку во Flash может быть только один перемещаемый клип, этой функции не требуются аргументы. Она влияет на текущий клип, с которым связана функция `startDrag()`.

Пример

В следующем сценарии экземпляр клипа `blob` делается перемещаемым. Клип остается перемещаемым до тех пор, пока пользователь не отпустит кнопку мыши, то есть не закончит перемещение клипа.

```
blob.onMouseDown=function(){
   startDrag("blob",false,0,0,550,400);
}

blob.onMouseUp=function(){
   stopDrag();
}
```

switch (6+)

Синтаксис

```
switch(условие){
case(a):
   //выражения, выполняемые, если условие=a
   break;
case(b):
   //выражения, выполняемые, если условие=b
   break;
case(n):
   //выражения, выполняемые, если условие=n
   break;
default:
   //выражения, выполняемые по умолчанию, если проверка условия возвращает false
}
```

Функция `switch()` представляет собой условную конструкцию, которая является альтернативой конструкции `if...else`. Эта функция применяется, когда существует несколько вариантов истинности условия, каждый из которых имеет ряд выражений, выполняемых последовательно. Аргумент *условие* — это проверяемое условное выражение. В каждую ветвь `case` поместите выражения, которые будут выполняться при истинности *условия*. Кроме того, в сценарий следует включить функцию `break`, которые не позволят перейти к следующей ветви, если проверка условия возвращает `true`. Выражения ветви `default` будут выполняться, если ни в одном из предыдущих случаев проверка условия не возвратила `true`.

Пример

В этом примере в клипе отслеживается текущее положение головки воспроизведения на основной временной шкале. В зависимости от номера текущего кадра основной временной шкалы клип должен задержаться в кадрах 1, 2 или 3 своей временной шкалы.

```
onClipEvent(enterFrame){
   switch(_root.currentframe){
   case(1):
      gotoAndStop(1);
      break;
   case(10):
      gotoAndStop(2);
      break;
   case(20);
      gotoAndStop(3);
      break;
   default:
      stop();
   }
}
```

targetPath (5+)

Синтаксис

```
targetPath(экземпляр_клипа)
```

Эта функция возвращает абсолютный путь к экземпляру клипа. Путь возвращается в виде строки с точечным синтаксисом.

Пример

В этом сценарии переменной me присваивается значение в виде абсолютного пути к клипу. Вы можете использовать его, когда необходимо указать путь.

```
onClipEvent(load){
   me=targetPath(this);
}
```

unloadMovie (3+)

Синтаксис

```
unloadMovie("имя_клипа")
```

Функция `unloadMovie()` применяется для удаления уровней, содержащих фильмы, которые загружены при помощи функций `loadMovie()` и `loadMovieNum()`. Функция `unloadMovie()` также может помочь удалить содержимое экземпляра клипа из области действия. В аргументе *имя_клипа* (указывается как строка) задается фильм или экземпляр клипа, который вы хотите удалить.

Пример

В этом выражении удаляется фильм на уровне 2:

```
unloadMovie("_level2");
```

В следующем выражении удаляется содержимое экземпляра клипа:

```
unloadMovie("feature1");
```

При удалении содержимого экземпляра клипа сам экземпляр остается в области действия в виде пустого кадра. Этот кадр можно заполнить новым фильмом (SWF) при помощи функции `loadMovie()`.

unloadMovieNum(4+)

Синтаксис

```
umloadMovieNum(уровень)
```

Функция `unloadMovieNum()` почти идентична функции `unloadMovie()`. Основное различие между ними состоит в том, что для функции `unloadMovieNum()` уровень загрузки документа определяется как число, а не как строка. Из-за этого данную функцию нельзя применять для удаления содержимого экземпляра клипа. Она применяется для удаления уровней вместе с их фильмами.

Пример

Два следующих выражения эквивалентны.

```
unloadMovieNum(2);
unloadMovie("_level2");
```

updateAfterEvent (5+)

Синтаксис

```
updateAfterEvent()
```

Функция `updateAfterEvent()` обновляет экран тотчас после совершения некоторого события. Не будучи связанным с частотой смены кадров фильма, данная функция наделяет программу способностью реагировать на нажатие клавиш клавиатуры и действия мыши, происходящие *между* кадрами. Функцию можно использовать для ввода данных с клавиатуры или других событий, связанных с вводом пользователем данных.

Пример

Приведенная ниже функция, связанная с экземпляром `cursor`, превращает этот экземпляр в пользовательский указатель мыши.

```
cursor.onMouseMove=function(){
   cursor._x=_xmouse;
   cursor._y=_ymouse;
   updateAfterEvent();
}
```

while (4+)

Синтаксис

```
while(условие){
   выражение;
}
```

При помощи функции `while()` создаются циклы. В цикле `while()` необходимо указать *условие*, которое будет проверяться. Если проверка условия возвращает `true`, будет выполняться *выражение*, которое содержится в теле цикла. Если проверка условия возвращает значение `false`, выражение, содержащееся в теле цикла, пропускается. В цикле `while()` условие всегда проверяется перед выполнением любых действий.

Пример

Сперва принято указывать переменную(-ые) цикла, а уже затем условие и само тело цикла. С помощью приведенного ниже цикла создаются копии клипа и произвольным образом помещаются на область действия.

```
var z;
z=0;
while(z<<10){
   ry=Math.random()*400;
   duplicateMovieClip("lite","lite"+z,z);
   _root["lite"+z]._x=50*z;
   _root["lite"+z]._y=ry;
   z++;
}
```

with (5+)

Синтаксис

```
with(объект){
   выражение;
}
```

Появившись во Flash 5, функция `with()` заменила функцию `tellTarget()`, применявшееся во Flash 3 и 4. Эта функция используется для получения доступа и управления свойствами объекта или экземпляра клипа, указанного в аргументе *объект*. Если вы хотите получить доступ к свойствам экземпляров клипа, функция `with()` позволяет избежать указания длинных путей к объекту в ссылках.

Пример

Этот сценарий устанавливает новое значение прозрачности (значение альфа-канала) и новые размеры для экземпляра клипа `menu`.

```
with(menu){
   _alpha=40;
   _xscale+=25;
   _yscale+=25;
}
```

А вот идентичный сценарий без использования функции `with()`.

```
_root.navigation.bar.menu._alpa=40;
_root.navigation.bar.menu._xscale+=25; _root.navigation.bar.menu._yscale+=25;
```

Свойства

В ActionScript *свойства* — это элементы, при помощи которых вы получаете доступ к данным о состоянии, положении, характеристиках и т. д. различных элементов фильма. Свойства можно ассоциировать практически со всеми компонентами фильма. Когда вы указываете значения свойств для отдельных элементов, каждый из них получает свой уникальный набор характеристик.

_alpha (4+)

Синтаксис

```
экземпляр._alpha=значение
```

При помощи свойства `_alpha` устанавливаются значения прозрачности для экземпляров объектов `Button` и клипов. Это свойство может иметь значения от 100 (объект непрозрачен) до 0 (объект прозрачен). Прозрачные кнопки и клипы остаются активными в области действия и действуют как обычные объекты.

_currentFrame (4+)

Синтаксис

```
экземпляр_клипа._currentFrame
```

Это свойство возвращает номер кадра клипа, который воспроизводится на момент вызова свойства. Если не указать значение аргумента *экземпляр_клипа*, свойство вернет номер текущего кадра фильма, с которым связан этот сценарий.

_dropTarget(4+)

Синтаксис

```
перемещаемый_экземпляр._dropTarget
```

Это свойство доступно только для чтения. Оно возвращает путь к клипу, в который был помещен другой клип, указанный в аргументе *перемещаемый_экземпляр*. Клип считается помещенным в другой, если точка регистрации перемещаемого клипа находится в границах главного клипа. Свойство возвращает путь, записанный с использованием символов (/). Если вы хотите, чтобы для записи пути использовался точечный синтаксис, воспользуйтесь функцией `eval()`. Например:

```
onClipEvent(mouseUp){
   trace("Использование символа /: "+this._dropTarget);
   trace("Точечный синтаксис: "+eval(this._dropTarget));
}
```

Пример

Когда вы "отпускаете" клип, который перетаскивали мышью, следует проверка, находится ли он в экземпляре `hit`. Если нет, то перемещаемый клип (его точка регистрации) помещается в точку с координатами (100;100).

```
onClipEvent(mouseUp){
   stopDrag();
   if(eval(this.dropTarget)!=_root.hit){
      this._x=100;
      this._y=100;
   }
}
```

_framesloaded (4+)

Синтаксис

```
_framesloaded
имя_файла._framesloaded
```

Это свойство возвращает количество кадров, которые были загружены из основной временной шкалы фильма или внешнего SWF-файла. С его помощью перед началом воспроизведения можно проверить, все ли кадры фильма загружены.

Пример

```
if(_framesloaded==_totalframes){
      gotoAndPlay("start");
   }else{
      gotoAndPlay(1);
}
```

В приведенном сценарии контролируется процесс загрузки кадров фильма основной временной шкалы. Если все кадры загружены, головка воспроизведения переходит к кадру с меткой `start`; в противном случае к кадру 1.

_height (4+)

Синтаксис

```
экземпляр._height=значение
```

Это свойство применяется, чтобы проверить или установить высоту отображения экземпляра клипа или кнопки. Аргумент *значение* данного свойства следует указывать в пикселах.

_name (4+)

Синтаксис

```
экземпляр._name="имя";
```

С помощью этого свойства указываются имена экземпляров клипов и кнопок. Вы можете проверить, имеет ли экземпляр имя и изменить его в случае необходимости. Чтобы присвоить экземпляру новое имя, введите соответствующее строковое значение в аргументе *имя*.

NaN (5+)

Синтаксис

```
NaN
```

NaN означает "not a number" ("не номер"). Это свойство применяется для указания элементов, которые не являются числами или которые нельзя обрабатывать как числовые данные.

_quality (5+)

Синтаксис

```
_quality="значение"
```

Это свойство устанавливает параметры качества изображения для вашего фильма. Качество изображения определяется сглаживанием, которое применяется в фильме. Аргумент *значение* принимает одно из четырех возможных строковых значений.

"LOW": отменяются любые функции сглаживания и устанавливается приоритет скорости воспроизведения над качеством.
"MEDIUM": сглаживание применяется только по отношению к векторным изображениям.
"HIGH": сглаживание применяется к векторным и к растровым изображениям, если в последних нет анимации.
"BEST": в этом случае фильм воспроизводится с наилучшим качеством, которое только достижимо, без учета возможности компьютера поддерживать требуемую скорость воспроизведения.

_rotation (4+)

Синтаксис

```
экземпляр._rotation=значение
```

С помощью свойства `_rotation` осуществляется поворот экземпляра кнопки или клипа на указанное в аргументе *значение* число градусов. Если аргумент *значение* является положительным числом, экземпляр поворачивается по часовой стрелке; если отрицательным — против. Вращение осуществляется относительно исходной ориентации кнопок и клипов на их временной шкале.

_soundbuftime (4+)

Синтаксис

```
_soundbuftime=значение
```

Свойство `_soundbuftime` определяет, сколько секунд потокового звука следует загрузить перед началом его воспроизведения. Значение по умолчанию составляет 5 секунд.

_totalframes (4+)

Синтаксис

```
экземпляр_клипа._totalframes
```

Свойство `_totalframes` возвращает количество кадров в клипе или в фильме основной временной шкалы. Это свойство применяется при создании анимационных изображений, которые воспроизводятся, пока идет процесс загрузки всех кадров фильма.

Пример

См. в описании свойства `_framesloaded`.

_url (4+)

Синтаксис

```
экземпляр._url
```

Свойство _url возвращает URL для SWF-файла, содержащего экземпляр кнопки или клипа, указанный в аргументе *экземпляр*. Возвращаемое свойством строковое значение содержит полный путь к требуемому файлу.

Пример

```
trace(this._url);
```

Если файл загружается с жесткого диска, будет выведен следующий результат:

```
file:///C/windows/Desktop/pathTest.swf
```

Для в файла, загружаемого с веб-узла, результат будет таким:

```
http://www.domain.com/folder/file.swf
```

_visible (4+)

Синтаксис

```
экземпляр._visible=Boolean
```

Данное свойство определяет, виден ли экземпляр кнопки или клипа, указанный в аргументе *экземпляр*, в области действий. Если это свойство имеет значение true (используется по умолчанию), экземпляр является видимым. Если свойство имеет значение false, экземпляр является скрытым. Скрытые (невидимые) экземпляры *клипов* реагируют на события и действия, а экземпляры *кнопок*, которые были сделаны невидимыми при помощи свойства _visible, — нет.

_width (4+)

Синтаксис

```
экземпляр._width=значение
```

Это свойство применяется, чтобы установить ширину экземпляра клипа или кнопки, указанного в аргументе *экземпляр*. Аргумент *значение* следует указывать в пикселах.

_x (3+)

Синтаксис

```
экземпляр._x
экземпляр._x=значение
```

С помощью свойства _x возвращается (в первом случае) или устанавливается (во втором случае) значение координаты положения экземпляра клипа или кнопки, на которые указывает аргумент *экземпляр*, по оси x. Координата определяется для точки регистрации и, следовательно, зависит от того, где находится экземпляр (см. ниже).

> Если экземпляр находится в фильме основной временной шкалы, координата определяется относительно левого края области действия. Свойство _x клипа или кнопки, удаленного на 225 пикселов от левого края области действия, будет иметь значение 225.
>
> Если экземпляр находится в клипе, его координаты определяются относительно точки регистрации этого клипа. Например, если он находится на 50 пикселов правее точки регистрации, его свойство _x будет иметь значение 50; если на 50 пикселов левее — -50.

_xmouse (5+)

Синтаксис

```
экземпляр._xmouse
```

Это свойство возвращает координату положения указателя мыши по оси x относительно экземпляра клипа или кнопки.

_root._xmouse возвращает координату положения указателя мыши относительно левого края области действий.
moverClip._xmouse возвращает координату положения указателя мыши относительно точки регистрации объекта moverClip.

Пример

Приведенный сценарий позиционирует экземпляр клипа на одной горизонтальной линии с указателем мыши относительно основной временной шкалы.

```
onClipEvent(enterFrame){
   _x=_root._xmouse;

}
```

_xscale(4+)

Синтаксис

```
экземпляр._xscale=значение
```

Это свойство применяется для горизонтального масштабирования экземпляра клипа или кнопки. Все значения масштабирования основываются на начальном размере экземпляра. Они вычисляются в процентном отношении к начальному размеру, и изображение выравнивается симметрично относительно точки регистрации экземпляра.

Пример

В приведенном сценарии при совершении события `enterFrame` экземпляр клипа масштабируется, то есть его ширина увеличивается на 50%. Поскольку масштабирование осуществляется симметрично, по 25% приходится на увеличение ширины влево и вправо.

```
onClipEvent(enterFrame){
   _xscale=_xscale+50;
}
```

_y (3+)

Синтаксис

```
экземпляр._y
экземпляр._y=значение
```

С помощью свойства `_y` возвращается (в первом случае) или устанавливается (во втором случае) значение координаты клипа или кнопки, указанного в аргументе `экземпляр`, по оси y. Значение координаты относится к точке регистрации и зависит от того, где находится экземпляр (см. ниже).

Если экземпляр находится в фильме основной временной шкалы, координата определяется относительно верхнего края области действия. Свойство _y экземпляра клипа или кнопки, находящегося на 100 пикселов ниже верхнего края области действия, будет иметь значение 100.

Если экземпляр находится в клипе, его координаты определяются относительно точки регистрации этого клипа. Например, если экземпляр находится на 50 пикселов ниже точки регистрации, его свойство _y будет иметь значение 50; если на 50 пикселов выше — -50.

_ymouse (5+)

Синтаксис

```
экземпляр._ymouse
```

Это свойство возвращает координату положения указателя мыши по оси y относительно экземпляра клипа или кнопки:

_root._ymouse возвращает координату положения указателя мыши относительно верхнего края области действия;
launcher._ymouse возвращает координату положения указателя мыши относительно точки регистрации кнопки launcher.

_yscale (4+)

Синтаксис

```
экземпляр._yscale=значение
```

Это свойство применяется для вертикального масштабирования экземпляра клипа или кнопки. Все значения масштабирования основываются на начальном размере экземпляра. Они вычисляются в процентном отношении к начальному размеру, и изображение выравнивается симметрично относительно точки регистрации экземпляра.

Операторы

Операторы применяются для вычисления и изменения значений выражений.

--(декрементирование) (4+)

Синтаксис

```
--A
A--
```

Оператор `--` можно использовать как перед некоторым выражением (в данном случае `A`), так и после него. В первом случае из выражения `A` вычитается 1 и возвращается новое значение; во втором тоже вычитается 1, но возвращается исходное значение выражения `A`.

Пример

```
X=2;
y=--x;
```

Оператор декрементирования помещен перед выражением, в результате получим y = 1, x = 1.

```
X=2;
y=x--;
```

Оператор декрементирования помещен после выражения и в результате получим y = 2, x = 1.

++(инкрементирование) (4+)

Синтаксис

```
++A
A++
```

Этот оператор можно использовать как перед выражением (в данном случае A), так и после него. В первом случае к выражению A прибавляется 1 и возвращается новое значение; во втором прибавляется 1, но возвращается исходное значение выражения A.

Пример

```
X=1;
y=++x;
```

Оператор инкрементирования помещен перед выражением, в результате получим y = 2, x = 2.

```
X=1;
y=x++;
```

Оператор инкрементирования помещен после выражения, в результате получим y = 1, x = 2.

!(логическое НЕ) (4+)

Синтаксис

```
!A
```

Оператор ! всегда возвращает значение типа Boolean, противоположное значению следующего за ним выражения A.

Пример

!1 возвращает 0.

!0 возвращает 1.

!=(неравенство) (5+)

Синтаксис

```
A!=B
```

Оператор != проверяет, действительно ли два выражения A и B не равны, и возвращает соответствующее логическое значение. Если выражения не равны, оператор возвращает true, в противном случае — false.

Пример

Оператор != можно использовать в сценарии анимации, которая отображает процесс загрузки. Если не все кадры фильма загрузились, головка воспроизведения переходит к кадру 1.

```
if(_framesloaded!=_totalframes){
   gotoAndPlay(1);
}else{
   gotoAndPlay(5);
}
```

!== (строгое неравенство) (6+)

Синтаксис

```
A!==B
```

Этот оператор, также как и оператор `!=`, проверяет, действительно ли два выражения `A` и `B` не равны, и возвращает соответствующее логическое значение. Единственно отличие состоит в том, что оператор `!==` учитывает также тип данных выражений.

Пример

У нас есть следующие выражения:

```
str="1";
num=new Number(1);
```

В первом случае возвращается значение `false`, поскольку значения переменных `str` и `num` *равны*:

```
str!=num;
```

Во втором случае возвращается значение `true`, поскольку переменные `str` и `num` не удовлетворяют строгому равенству — они относятся к различным типам данных (строка и число):

```
str!==num;
```

% (деление по модулю) (4+)

Синтаксис

```
A%B
```

Этот оператор возвращает остаток, полученный при делении выражения `A` на выражение `B`.

Пример

`17%4` возвращает 1.

Если 17 разделить на 4, получим 4 и в остатке 1. Оператор `%` возвращает остаток деления двух выражений.

%= (присвоение по модулю) (4+)

Синтаксис

```
A%=B
```

Этот оператор возвращает остаток, полученный при делении выражения `A` на выражение `B`, и присваивает его выражению `A`.

Пример

```
X=2;
y=13;
y%=x;
trace(y);
```

В этом примере в окне Output будет выведен результат 1. Переменная y делится на переменную x и остаток (1) присваивается переменной y.

&& (логическое И) (4+)

Синтаксис

```
A&&B
```

Этот оператор оценивает два выражения A и B и возвращает логическое значение true или false в зависимости от значения каждого выражения.

true&&false возвращает false.

true&&true возвращает true.

false&&false возвращает false.

Чтобы оператор && возвратил true, оба выражения должны быть равны true.

- (вычитание) (4+)

Синтаксис

```
A-B
```

Этот оператор либо вычитает выражение B из выражения A, либо присваивает выражению противоположное значение.

Пример

1,5-0,4 возвращает 1,1.

4-3 возвращает 1.

-(4-3) возвращает -1.

-(1-2) возвращает 1.

* (умножение) (4+)

Синтаксис

```
A*B
```

Этот оператор умножает выражение A на выражение B.

*= (присвоение результата умножения) (4+)

Синтаксис

```
A*=B
```

Этот оператор умножает выражение A на выражение B и присваивает произведение выражению A.

Пример

Запись `a*=b` эквивалентна `a=a*b`.

В следующем коде переменной `a` присваивается новое значение 200.

```
a=10
b=20
a*=b
```

?: (условный) (4+)

Синтаксис

```
условие?выражение1:выражение2
```

Условный оператор `?:` является альтернативой конструкции `if…else` и требует всего одну строку кода. Оператор производит оценку *условия* и, если оно истинно, возвращает *выражение1*, в противном случае возвращает *выражение2*.

Пример

```
if(Key.getCode()==Key.RIGHT){
   nextFrame();
}else{
   stop();
}
```

В этом сценарии головка воспроизведения переходит на следующий кадр, если нажата правая клавиша мыши. Однако его можно записать и в следующем виде:

```
Key.RIGHT?nextFrame():stop();
```

// (однострочный комментарий) (1+)

Синтаксис

```
//комментарий
```

Если в коде сценария присутствует оператор `//`, это значит, что текст между символом оператора и символом конца строки, на которой находится оператор, следует игнорировать. В комментарии вы можете внести свои замечания и пояснения к сценарию. По умолчанию текст комментариев окрашивается в светло-серый цвет.

/*...*/ (многострочный комментарий) (5+)

Синтаксис

```
/*начало_комментария…конец_комментария*/
```

Этот оператор используется, если вы помещаете в сценарий большое количество текста, например блок из нескольких строк, которые должны игнорироваться. Программа будет пропускать весь текст между первой и последней звездочками. Не пропустите один из этих операторов при включении в код комментариев.

/= (присвоение результата деления) (4+)

Синтаксис

```
A/=B
```

Этот оператор возвращает значение, полученное в результате деления выражения A на выражение B, и присваивает его выражению A.

Пример

Запись x/=y эквивалентна x=x/y.

Переменной x присваивается значение частного, полученного при выполнении операции x/y.

|| (логическое ИЛИ) (4+)

Синтаксис

```
A||B
```

Этот оператор проверяет выражение A и выражение B и возвращает значение типа Boolean. Если одно из выражений является истинным, оператор возвращает true. Если оба выражения ложны, возвращается значение false.

Для 1||0 оператор возвращает значение true.

Для 0||1 оператор возвращает значение true.

Для 1||1 оператор возвращает значение true.

Для 0||0 оператор возвращает значение false.

+ (сложение) (4+)

Синтаксис

```
A+B
"строка1"+"строка2"
```

Оператор возвращает сумму выражений A и B. Его также можно использовать для объединения строк (только во Flash Player 5 и более поздних версиях).

+= (присвоение результата сложения) (4+)

Синтаксис

```
A+=B
C+="строка"
```

Этот оператор возвращает сумму выражений A и B и присваивает ее значение выражению A. Данный оператор работает с числами и строками.

Выражение x+=1 эквивалентно x=x+1.

```
var x=10, y=20
```

x+=y возвращает 30.

Оператор может также соединить строку с переменной и создать тем самым новое значение:

```
x="Action";
trace(x+="Script");
```

В приведенном примере в окне **Output** (Выходные данные) будет выведено слово ActionScript.

<< (меньше) (4+)

Синтаксис

```
A<<B
```

Если выражение A меньше выражения B, оператор << возвращает true. В противном случае (то есть если A больше или равно B) он возвращает false. Этот оператор может также обрабатывать строки. Прописные буквы, расположенные ближе к A, имеют меньшие значения, наибольшие значения имеют строчные буквы, расположенные ближе к z.

<<= (меньше или равно) (4+)

Синтаксис

```
A<<=B
```

Если выражение A меньше или равно выражению B, оператор <<= возвращает true. В противном случае (то есть если A больше B) он возвращает false. Оператор может также обрабатывать строки. Прописные буквы, расположенные ближе к A, имеют меньшие значения, наибольшие значения имеют строчные буквы, расположенные ближе к z.

= (присвоение) (4+)

Синтаксис

```
идентификатор=A
```

Этот оператор присваивает значение выражения A некоторому *идентификатору*. Аргумент идентификатор представляет собой имя, присвоенное свойству объекта, переменной или элементу массива.

- = (присвоение результата вычитания) (4+)

Синтаксис

```
A-=B
```

Этот оператор возвращает разность выражений A и B и присваивает ее значение выражению A.

Пример

Выражение y-=x эквивалентно y=y-x.

```
var x=10, y=20
```

y-=x возвращает 10.

== (равенство) (4+)

Синтаксис

```
A==B
```

Если выражение A равно выражению B, оператор возвращает true. В противном случае оператор возвращает false.

=== (строгое равенство) (6+)

Синтаксис

```
A===B
```

Подобно оператору равенства этот оператор сравнивает выражения A и B и возвращает логическое значение в зависимости от того, равны они ли нет. Их отличие состоит в том, что оператор строгого равенства учитывает также тип данных выражений.

Пример

Возьмем следующие выражения:

```
str="1";
num=new Number(1);
```

Оператор равенства str==num возвращает значение true, так как значения переменных равны (единица). Но оператор строгого равенства str===num возвращает значение false, поскольку переменные str и num содержат данные различных типов.

>> (больше) (4+)

Синтаксис

```
A>>B
```

Если выражение A больше выражения B, оператор >> возвращает true. В противном случае (то есть если A меньше или равно B) оператор возвращает false. Оператор может также обрабатывать строки. Заглавные буквы, расположенные ближе к A, имеют меньшие значения, наибольшие значения имеют строчные буквы, расположенные ближе к z.

>>= (больше или равно) (4+)

Синтаксис

```
A>>=B
```

Если выражение A больше выражения B или равно ему, оператор >>= возвращает true. В противном случае оператор возвращает false. Оператор может также обрабатывать строки. Заглавные буквы, расположенные ближе к A, имеют меньшие значения, наибольшие значения имеют строчные буквы, расположенные ближе к z.

Пример

В следующих выражениях оператор возвращает значение true:

```
2>=1
3>=3
"shockwave ">="flash "
```

В следующем выражении оператор возвращает значение false:

```
"Flash ">="flash "
```

Выражение Flash не будет больше или равно выражению flash, поскольку считается, что заглавные буквы имеют меньшее значение, чем строчные.

typeof (5+)

Синтаксис

```
typesof A
```

Оператор typeof возвращает тип данных, содержащихся в выражении A. Это могут следующие типы данных: Boolean, function, movieclip, number, object и string.

Пример

```
obj = new Object;
trace("2: "+typeof 2);
trace("text: "+typeof "text");
trace("true: "+typeof true);
trace("obj: "+typeof obj);
```

В результате выполнения этого кода в окне будет отображено следующее:

```
2: number
text: string
true: Boolean
obj: object
```

То есть 2 — это число, text — значение типа string, true — логическое значение, а obj — пользовательский объект. Для элементов, не содержащих данных, оператор возвращает значение undefined или null.

Объектные ссылки и ссылки на временную шкалу

Объектные ссылки — это элементы, указывающие на клипы, уровни фильма, переменные, объекты и функции Flash-фильмов. Правильное использование этих элементов является залогом успеха в указании точных путей к различным объектам. Если встал вопрос об указании пути к какому-либо объекту, его правильность всегда можно проверить при помощи панелей **Movie Explorer**, **Debugger** (Отладчик) или кнопки **Insert Target Path** (Вставить путь) панели **Actions** (Действия).

_global (6+)

Синтаксис

```
global.имя_переменной
global.имя_функции
```

Эта ссылка позволяет создавать глобальные переменные, объекты или функции. Определение *глобальный* означает, что переменная, объект или функция доступны из любой временной шкалы вашего фильма. Вместо того чтобы указывать путь, например _root.myFunction, ссылка _global позволяет просто вызвать функцию myFunction. Для того чтобы к функции или переменной можно было получить доступ по имени из любой временной шкалы, при их инициализации укажите ссылку _global.

Пример

В приведенном коде переменная startGame объявляется как глобальная.

```
_global.startGame=getTimer();
```

Эту переменную можно вызвать в любом месте фильма, просто указав ее имя, `startGame`. Если вы не воспользовались ссылкой `_global` при создании этой переменной в фильме главной временной шкалы, к ней придется обращаться, указывая путь `_root.startGame`.

_level (4+)

Синтаксис

```
_levelX
```

Эта ссылка указывает на временную шкалу фильма, загруженного на уровне `X`. Приложение Flash Player может одновременно хранить и воспроизводить множество фильмов. Главная временная шкала (первый загруженный фильм) находится на уровне 0. Все последующие фильмы загружаются на более высоких уровнях: уровне 1, уровне 2 и т. д. Ссылка `_level` применяется, чтобы загрузить или выгрузить дополнительный фильм или управлять его воспроизведением.

_lockroot (7)

Синтаксис

```
временная_шкала._lockroot=boolean
```

Эта ссылка определяет значение ссылки на главную временную шкалу `_root`. Если ссылке присвоить значение `false`, во всех сценариях ссылка `_root` будет указывать на временную шкалу `_level0`. Если ссылке присвоить значение `true`, ссылка `_root` будет указывать на главную временную шкалу фильма, в котором была вызвана ссылка `_lockroot`, при загрузке одного фильма на уровень или в экземпляр клипа другого фильма. С помощью ссылки `_lockroot` предотвращается неадекватное функционирование ссылки `_root`. (Примеры см. в главе 19.)

_parent (4+)

Синтаксис

```
экземпляр._parent
```

При помощи ссылки `_parent` можно обратиться к временной шкале фильма, в котором содержится клип или кнопка. Например, при помощи ссылки `_parent` можно обратиться к временной шкале главного фильма из экземпляра клипа или кнопки, в нем находящегося. В фильме главной временной шкалы эту ссылку использовать нельзя, поскольку он расположен на самом низком уровне из всех возможных (`_level0` или `_root`). Ссылка `_parent` особенно полезна при использовании относительных путей к объекту.

Пример

Если экземпляр клипа `arrow` находится в фильме главной временной шкалы, вызвав нижеприведенный сценарий во временной шкале клипа `arrow`, вы остановите воспроизведение фильма.

```
_parent.stop();
```

Если в клипе `arrow` содержится вложенный экземпляр клипа `fletcher`, вызвав следующий сценарий во временной шкале клипа `fletcher`, вы остановите воспроизведение фильма главной временной шкалы, сделаете экземпляр `arrow` полупрозрачным и повернете на 180° экземпляр `head`, также находящийся в клипе `arrow`.

```
_parent._parent.stop();
_parent._alpha=50;
_parent.head._rotation=180;
```

_root (4+)

Синтаксис

```
root.экземпляр_клипа
root.функция
_root.свойство
```

Ссылка `_root` указывает на главную временную шкалу фильма (`level0`). Эту ссылку можно использовать для создания полного пути к объекту, указанному в аргументе *экземпляр_клипа*, или для управления главной временной шкалой из любой части фильма.

this (5+)

Синтаксис

```
this.функция
this.свойство
this.nestedClipA.nestedClipB
```

Ссылка `this` используется как самовызов экземпляров символов для текущей временной шкалы. Если вызвать `this` из обработчика событий мыши, связанного с кнопкой, она укажет на временную шкалу фильма, в котором эта кнопка находится. Если эта ссылка вызвана из обработчика событий клипа, она укажет на временную шкалу этого клипа. Ссылку `this` можно использовать для управления воспроизведением, установкой/получением свойств текущей временной шкалы. Она также применяется при создании пути к объекту, начинающемуся с указания текущей временной шкалы.

Предопределенные объекты

Предопределенные объекты (`Array`, `Color`, `Date`, `Key`, `Math`, `Mouse`, `MovieClip`, `Object` (`Generic`), `Sound`, и `TextField`) управляют свойствами и данными, связанными со встроенными Flash-объектами. Полный список встроенных объектов вы можете получить, воспользовавшись панелью **Help** (Справка) (вызывается нажатием клавиши **F1**).

Объект Array (5+)

Синтаксис:

```
new Array()
new Array(размер)
new Array (элемент0,элемент1,элемент2…элементх)
```

Массив представляет собой упорядоченный набор данных. Свойства и методы объекта `Array` можно использовать для хранения, извлечения и обработки элементов, находящихся в массиве.

Для создания объекта `Array` можно воспользоваться функцией `new Array()`. Например:

```
list=new Array();
```

Указав эту функцию без аргументов, вы создадите новый пустой массив. Однако функция может принимать и необязательные аргументы. Указав целое число в аргументе *размер*, вы установите размер массива, то есть количество элементов в нем.

```
week=new Array(7);
```

В данном случае создается пустой массив `week`, в который можно записать максимум 7 элементов. Также допускается заполнение массива сразу при его создании.

```
groceries=new Array("pasta","bread","milk");
```

Массив `groceries` содержит три элемента типа `string`. В массиве могут содержаться и смешанные данные.

```
home=new Array();
home[0]="Chicago";
home[1]=60657;
home[2]=["Lakeview","Wrigleyville"];
```

В этом примере при создании массива `home` применяется оператор доступа к элементу массива (`[]`). Массив `home` содержит три элемента: `"Chicago"` (тип `string`), `60657` (тип `number`) и другой массив с элементами `"Lakeview"` и `"Wrigleyville"` типа `string`. Все элементы массива сохраняются в памяти последовательно, и обращаться к элементу массива следует по смещению его адреса, то есть индексу (индекс первого элемента — 0, второго — 1 и т. д.). При доступе к элементу массива по индексу следует указывать оператор `[]`. Например, выражение `trace(home[1])` возвращает значение `60657`, то есть второй элемент массива.

Свойства:

`Array.length` возвращает количество элементов в массиве.

Методы:

`Array.concat()` создает новый массив, добавляя дополнительные элементы в уже существующий.

`Array.pop()` удаляет последний элемент массива, но возвращает значение этого последнего элемента.

`Array.push()` добавляет дополнительные элементы в конец массива.

`Array.shift()` удаляет первый элемент массива, но возвращает значение этого первого элемента.

`Array.slice()` создает новый массив, используя некоторую часть уже существующего.

`Array.sort()` упорядочивает элементы массива в соответствии с правилами, установленными пользователем.

`Array.toString()` возвращает элементы массива в одной строке, разделяя их при помощи запятых.

`Array.unshift()` добавляет дополнительные элементы в начало массива.

Array.concat (5+)

Синтаксис:

```
имя_массива.concat(элемент1, элемент2…элементx)
```

Этот метод используется для добавления дополнительных элементов в конец существующего массива. При этом создается совершенно новый массив.

Пример

Создадим массив `letter1`.

```
letter1=new Array("a","b","c");
```

Создадим массив `letter2`, добавив два элемента в массив `letter1`.

```
letter2=letter1.concat("d","e");
```

В массиве `letter2` содержатся следующие элементы.

```
["a","b","c""d","e"]
```

Array.length (5+)

Синтаксис:

```
имя_массива.length
```

Это свойство можно использовать как для установки, так и для извлечения размера массива, то есть количества элементов, содержащихся в массиве, указанном в аргументе *имя_массива*. Поскольку нумерация индексов элементов в массиве начинается с 0, размер массива всегда на 1 больше, чем индекс последнего его элемента. Значение свойства `length` изменяется, если в массив добавляются элементы или удаляются из него. Таким образом, свойство `length` всегда отражает текущее состояние массива, даже если он на данный момент пуст.

Пример

Создадим массив `alphabet`.

```
alphabet=new Array("a","b","c");
```

В окне **Output** (Выходные данные) будет выведено значение `3`, поскольку в массиве `alphabet` всего три элемента.

```
trace(alphabet.length);
```

Array.pop (5+)

Синтаксис:

```
имя_массива.pop()
```

Этот метод применяется для удаления последнего элемента массива, указанного в аргументе *имя_массива*, и возвращает его значение. При вызове данного метода последний элемент массива удаляется, а значение, находившееся в этом элементе, возвращается. Метод противоположен методу `shift()`, который удаляет первый элемент массива и возвращает его значение.

Пример

В следующем выражении удаляется последний элемент массива `alphabet`.

```
alphabet=new Array("a","b","c");
popped=alphabet.pop();
```

В окне **Output** (Выходные данные) будет выведена надпись `c удален`, поскольку именно элемент `c` был последним в массиве:

```
trace(popped+"удален");
```

Array.push (5+)

Синтаксис:

```
имя_массива.push(элемент1, элемент2…элементx)
```

Этот метод добавляет указанные элементы в массив. В отличие от метода `concat()` новый массив при этом не создается, а в отличие от метода `unshift()` элементы помещаются в конец, а не в начало массива.

Пример

Создадим массив alphabet.

```
alphabet=new Array("a","b","c");
```

В следующем выражении в массив alphabet добавляются элементы d, e и f.

```
alphabet.push("d","e","f");
```

Array.shift (5+)

Синтаксис:

```
имя_массива.shift()
```

Этот метод применяется для удаления первого элемента массива, указанного в аргументе *имя_массива*, и возвращает значение удаляемого элемента. При вызове данного метода первый элемент массива удаляется, а значение этого элемента возвращается. Метод противоположен методу Array.pop(), который удаляет последний элемент массива и возвращает его значение.

Пример

В следующем выражении удаляется первый элемент массива alphabet.

```
alphabet=new Array("a","b","c");
shifted=alphabet.shift();
```

В окне **Output** (Выходные данные) будет выведена строка a удален, поскольку именно элемент a был первым элементом массива.

```
trace(shifted+"удален"              );
```

Array.slice (5+)

Синтаксис:

```
имя_массива.slice(индекс_1,индекс_2)
```

Метод Array.slice() применяется для создания нового массива на основе определенной части ужу существующего, указанного в аргументе *имя_массива*. Чтобы выделить эту часть массива, вам следует указать два аргумента: *индекс_1* и *индекс_2*. Оба аргумента содержат индексы, заданные с учетом того, что начальный элемента массива имеет индекс 0. Аргумент *индекс_1* представляет первый элемент выделенной части существующего массива, которая будет включена в новый массив. Если этот аргумент содержит отрицательное число, отсчет будет вестись с конца существующего массива. Например, -1 — номер последнего элемента массива, -2 — номер предпоследнего элемента и т. д. Аргумент *индекс_2* — это последний элемент выделенной части существующего массива, который будет включен в новый массив. Аргумент *индекс_2* может быть задан как отрицательное число — так же как и аргумент *индекс_1*. Если не указать этот аргумент, будет использовано его значение по умолчанию — *имя_массива*.length — и все оставшиеся элементы существующего массива перейдут в новый массив. Короче говоря, в новый массив входят элементы, находящиеся между элементом *индекс_1* и элементом *индекс_2* (включительно) существующего массива.

Пример

Создадим новый массив:

```
nums=new Array("one","two","three","four","five");
```

Следующие выражения эквивалентны. Во всех создается новый массив с двумя последними элементами массива — four и five.

```
sub1=nums.slice(3,5);
sub2=nums.slice(3);
sub3=nums.slice(-2,5);
```

Array.sort (5+)

Синтаксис:

```
имя_массива.sort()
имя_массива.sort(функция_сортировки)
```

При помощи метода Array.sort() элементы массива, указанного в аргументе имя_массива, сортируются в определенном порядке. Если функция сортировки не указана, элементы будут сортироваться в таком порядке: вначале — целые числа, затем — символы верхнего регистра (в алфавитном порядке) и, наконец, символы нижнего регистра.

Если вы укажете функцию сортировки, массив будет отсортирован в соответствии с правилами, устанавливаемыми этой функцией. Функция сортировки должна удовлетворять вашим нуждам. При ее создании следует соблюдать некоторые простые правила. Функция должна предоставлять возможность сравнивать два значения. Если вы хотите, чтобы при сортировке первое значение предшествовало второму, функция должна возвращать -1. Если при сортировке второе значение должно предшествовать первому, функция должна возвращать 1. Для равных значений, не требующих сортировки, функция будет возвращать 0. Элементы массива сортируются в таком порядке: вначале отрицательные числа, затем — положительные. Например, отсортируем значения А и Б в возрастающем порядке.

А<<Б — функция возвращает -1.

А>>Б — функция возвращает 1.

А==Б — функция возвращает 0.

Чтобы отсортировать значения в убывающем порядке, вам просто нужно поменять значения, возвращаемые функцией. Когда А<<Б, функция будет возвращать 1; если А>>Б — -1; при равенстве значений — 0.

Array.unshift (5+)

Синтаксис:

```
имя_массива.unshift(элемент1, элемент2…элементх)
```

Этот метод добавляет указанные элементы в массив, указанный в аргументе *имя_массива*. В отличие от метода concat() новый массив при этом не создается, а в отличие от метода push() элементы помещаются в начало, а не в конец массива.

Пример

Создадим массив alphabet.

```
alphabet=new Array("d","e","f");
```

В следующем выражении в массив alphabet добавляются элементы a, b и c.

```
alphabet.unshift("a","b","c");
```

Объект Color (5+)

Синтаксис:

```
new Color(имя_клипа)
```

В объекте `Color` хранится информация, позволяющая работать со значениями составляющих цвета и прозрачности клипа или фильма, указанного в аргументе *имя_клипа*. Перед вызовом методов объекта `Color` необходимо создать новый объект посредством функции `new Color()`.

Методы:

`Color.getRGB()` возвращает текущее RGB-значение объекта `Color`.

`Color.getTransform()` возвращает текущее значение параметра трансформации объекта `Color`.

`Color.setRGB()` задает текущее RGB-значение объекта `Color`.

`Color.setTransform()` задает текущее значение параметра трансформации объекта `Color`.

Color.getRGB (5+)

Синтаксис:

```
объектColor.getRGB
```

Этот метод возвращает RGB-значение для указанного объекта `Color`.

Пример

При щелчке мышью на клипе, с которым связан приведенный ниже сценарий, RGB-значения цвета указанного объекта переносятся из одного клипа в другой. Метод `getRGB()` извлекает значение цвета объекта `orbColor` клипа `orb` и применяет его к текущему экземпляру клипа.

```
onClipEvent(mouseUp){
   if(this.hitTest(_root._xmouse,_root._ymouse,true)){
      colorTrans=new Color(this);
      colorTrans.setRGB(_root.orb.orbColor.getRGB());
   }
}
```

Color.getTransform (5+)

Синтаксис:

```
объектColor.getTransform()
```

Этот метод извлекает текущие параметры трансформации, установленные для указанного объекта `Color`.

Color.setRGB (5+)

Синтаксис:

```
colorObject.setRGB
```

Этот метод применяется, чтобы задать RGB-значение для указанного объекта `Color`.

Пример

В приведенном сценарии устанавливается красный цвет для экземпляра клипа, с которым он связан.

```
onClipEvent(load){
   clipColor=new Color(this);
   clipColor.setRGB(0xFF000000);
```

См. также описание метода `Color.getRGB()`.

Color.setTransform (5+)

Синтаксис:

```
объектColor.setTransform(Объект1)
```

Метод `setTransform()` позволяет манипулировать цветовыми составляющими объекта `Color`: красной, зеленой, синей и альфа-значением. Вы можете устанавливать долю (в процентах) или значение каждой составляющей. Чтобы воспользоваться этим методом, сначала необходимо создать объект и присвоить ему свойства, приведенные в табл. С.1. Созданный объект становится объектом, указанным в аргументе `Объект1`. При вызове метода `setTransform()` его свойства применяются к объекту `Color`.

Таблица С.1. Свойства объекта Объект1

Свойство	Диапазон	Описание
ra	от -100 до 100	Доля красного
rb	от -255 до 255	Значение красного
ga	от -100 до 100	Доля зеленого
gb	от -255 до 255	Значение зеленого
ba	от -100 до 100	Доля синего
bb	от -255 до 255	Значение синего
aa	от -100 до 100	Степень прозрачности (значение альфа-канала в процентах)
ab	от -255 до 255	Альфа-значение

Доля в процентах — это значение, применяемое к исходному значению цветовой составляющей. Например, если установить значение `ba=50`, голубая составляющая цвета объекта станет равной всего лишь половине своей исходной величины. Значение также можно увеличивать или уменьшать. Если установить `gb=255`, значение зеленой составляющей цвета объекта подскакивает до максимума.

Объект Date (5+)

Синтаксис:

```
new Date()
new Date(год,месяц,число,часы,минуты,секунды,миллисекунды);
```

В ActionScript объект `Date` позволяет получить текущую дату в соответствии с местным временем или временем по Гринвичу. Местное время определяется настройками системы компьютера, на котором воспроизводится фильм. В этом справочнике не рассматриваются методы объекта `Date`, связанные с временем по Гринвичу.

Объект `Date` позволяет получить текущие дату и время, сохранить их и использовать в фильмах. Существует два способа создания объекта `Date`. Чтобы создать объект для текущих даты и времени, можно воспользоваться функцией `new Date()`. Если необходимо создать объект для указанной даты, используют аргументы объекта `Date`.

Год — целое число, обозначающее год. Чтобы указать год с 1900 по 1999, используйте двузначные числа от 0 до 99 (например, число 73 означает 1973 год). Года с 1000 по 1899, а также с 2000 и далее обозначаются четырехзначными числами. Трехзначные числа применяются для обозначения годов до 1000 года н. э.

Месяц — целое число от 0 (январь) до 11 (декабрь), обозначающее месяц.

Число — целое значение от 1 до 31, обозначающее число месяца. Это — необязательный аргумент.

Часы — целое число от 0 (полночь) до 23 (11 часов вечера), обозначающее часы. Аргумент *часы* необязателен.

Минуты — целое число от 0 до 59, обозначающее минуты. Это —необязательный аргумент.

Секунды — целое число от 0 до 59, обозначающее секунды. Это —необязательный аргумент.

Миллисекунды — целое число от 0 до 999, обозначающее миллисекунды, необязательный аргумент. В ActionScript миллисекунды имеют большое значение, поскольку вся информация, касающаяся дат, хранится в виде количества миллисекунд, прошедших начиная с полуночи 1 января 1970 года. Методы объекта Date позволяют манипулировать этими длинными громоздкими числами и переводить их в понятные даты с днями, месяцами, годами и т. д.

Методы

Создав объект `Date`, вы можете воспользоваться следующими его методами, позволяющими извлекать или устанавливать дату.

Date.getDate() возвращает число месяца.

Date.getDay() возвращает день недели.

Date.getFullYear() возвращает год в виде четырехзначного числа.

Date.getHour() возвращает часы.

Date.getMilliseconds() возвращает миллисекунды.

Date.getMinutes() возвращает минуты.

Date.getMonth() возвращает месяц года.

Date.getSeconds() возвращает секунды.

Date.getTime() возвращает количество миллисекунд, прошедших с 1 января 1970 года.

Date.getYear() возвращает год относительно 1900 года.

Date.setDate() позволяет задать число.

Date.setFullYear() позволяет задать год в виде четырехзначного числа.

Date.setHour() позволяет задать часы.

Date.setMilliseconds() позволяет задать миллисекунды.

Date.setMinutes() позволяет задать минуты.

Date.setMonth() позволяет задать месяц года.

Date.setSeconds() позволяет задать секунды.

Date.setTime() задает дату в миллисекундах относительно полуночи 1 января 1970 года.

Date.setYear() задает год в виде как двузначного, так и четырехзначного числа.

Date.toString () возвращает строку, содержащую дату как по местному времени, так и по Гринвичу.

Пример

В этом сценарии создается новый объект `Date` для текущих даты и времени. Затем в нем используются различные методы для форматирования даты и в соответствующих текстовых полях выводится значение времени.

```
//Создаем новый объект Date;
today=new Date();
//Форматируем каждую единицу времени
//в отдельном текстовом поле
hours=today.getHours();
minutes=today.getMinutes();
seconds=today.getSeconds();
```

Следующий сценарий связан с кнопкой. В нем вычисляется количество дней, остающихся до дня рождения, указанного в текстовых полях `month` и `day`.

```
on(release){
   today=new Date();
   bDay=new Date(today.getFullYear(),month-1,day)
   var wait=Math.ceil((bDay-today)/86400000);
   if(wait<<0){
      printout="Поздравляю с прошедшим днем рождения!";
   }else{
      printout=wait+" дней осталось до следующего дня рождения.";
   }
}
```

В приведенном примере используются два объекта `Date`: один — для текущей даты (`today`), а другой — для даты дня рождения, указанной в полях `month` и `day`. Обратите внимание: поскольку в ActionScript нумерация месяцев начинается с 0 (0 обозначает январь), необходимо вычесть 1 из значения, введенного в поле `month`. Разница между значениями двух дат вычисляется в миллисекундах. Чтобы получить результат в более понятных числах, его делят на 86400000 (количество миллисекунд в одном дне) и округляют при помощи метода `Math.ceil()` до следующего большего целого числа. Результат выводится в поле `printout`.

Date.getDate (5+)

Синтаксис:

```
объектDate.getDate()
```

Метод `getDate()` возвращает значение дня месяца для объекта `Date`. Данное значение представляет собой целое число от 1 до 31.

Date.getDay (5+)

Синтаксис:

```
объектDate.getDay()
```

Метод `getDay()` возвращает значение дня недели для объекта `Date`. Значение дня недели представляет собой целое число от 0 (воскресенье) до 6 (суббота).

Date.getFullYear (5+)

Синтаксис:

```
объектDate.getFullYear()
```

Метод `getFullYear()` возвращает значение года для объекта `Date` в виде четырехзначного числа.

Date.getHours (5+)

Синтаксис:

```
объектDate.getHours()
```

Метод `getHours()` возвращает значение часов для объекта `Date`. Это значение представляет собой целое число от 0 (полночь) до 23 (11 часов вечера). Обратите внимание на то, что объект `Date` не поддерживает обозначения A.M. (до полудня) и P.M (после полудня).

Date.getMilliseconds (5+)

Синтаксис:

```
объектDate.getMilliseconds()
```

Метод `getMilliseconds()` возвращает значение миллисекунд для объекта `Date`. Значение миллисекунд представляет собой целое число от 0 до 999.

Date.getMinutes (5+)

Синтаксис:

```
объектDate.getMinutes()
```

Метод `getMinutes()` возвращает значение минут для объекта `Date`. Это значение представляет собой целое число от 0 до 59.

Date.getMonth (5+)

Синтаксис:

```
объектDate.getMonth()
```

Метод `getMonth()` возвращает значение месяца для объекта `Date`. Значение месяца представляет собой целое число от 0 (январь) до 11 (декабрь).

Date.getSeconds (5+)

Синтаксис:

```
объектDate.getSeconds()
```

Метод `getSeconds()` возвращает значение секунд для объекта `Date`, которое представляет собой целое число от 0 до 59.

Date.getTime (5+)

Синтаксис:

```
объектDate.getTime()
```

Во Flash все объекты `Date` сохраняются как одно число, представляющее собой количество миллисекунд, прошедших с полуночи 1 января 1970 года. Метод `getTime()` возвращает это количество миллисекунд для объекта `Date`. Такой способ обработки дат позволяет легко их сравнивать.

Date.getYear (5+)

Синтаксис:

```
объектDate.getYear()
```

Метод `getYear()` применяется для получения значения года для объекта `Date` относительно 1900 года. Например, для 1973 года метод возвратит значение 73, а для 2004 года — 104.

Date.setDate (5+)

Синтаксис:

```
объектDate.setDate(число)
```

Метод `setDate()` позволяет задать значение числа месяца для объекта `Date`. Значение указывается в аргументе `число` как целое число от 1 до 31.

Date.setFullYear (5+)

Синтаксис:

```
объектDate.setFullYear(год,месяц,число)
```

При помощи метода `setFullYear()` можно задать точную дату для объекта `Date`. Метод принимает следующие аргументы:

год — четырехзначное число, обозначающее год;

месяц — число от 0 (январь) до 11 (декабрь), обозначающее месяц;

число — число месяца от 1 до 31.

Date.setHours (5+)

Синтаксис:

```
объектDate.setHours()
```

Данный метод применяется, чтобы установить значение часов для объекта `Date`. Это значение указывается как целое число от 0 (полночь) до 23 (11 часов вечера).

Date.setMilliseconds (5+)

Синтаксис:

```
объектDate.setMilliseconds()
```

Этот метод применяется, чтобы установить значение миллисекунд для объекта `Date`. Значение должно быть целым числом от 0 до 999.

Date.setMinutes (5+)

Синтаксис:

```
объектDate.setMinutes()
```

Метод `setMinutes()` используется для того чтобы задать значение минут для объекта `Date`. Значение минут указывается в виде целого числа от 0 до 59. Если указать значение, выходящее за диапазон чисел от 0 до 59, значение часа объекта `Date` будет изменено на предыдущий или следующий час соответственно. Например, значение 60 минут эквивалентно первой минуте следующего часа.

Date.setMonth (5+)

Синтаксис:

```
объектDate.setMonth()
```

Метод `setMonth()` задает значение месяца года для объекта `Date`. Значение месяца указывается в виде целого числа от 0 (январь) до 11 (декабрь). Если указать значение, выходящее за диапазон чисел от 0 до 11, значение года объекта `Date` будет изменено на предыдущий или следующий год соответственно.

Date.setSeconds (5+)

Синтаксис:

```
объектDate.setSeconds()
```

Этот метод позволяет задать значение секунд для объекта `Date`, которое представляет собой целое число от 0 до 59. Если указать значение, выходящее за диапазон чисел от 0 до 59, значение минут объекта `Date` будет изменено на предыдущую или следующую минуту соответственно. Например, значение секунд -1 эквивалентно последней (59-ой) секунде предыдущей минуты.

Date.setTime (5+)

Синтаксис:

```
объектDate.setTime()
```

Этот метод можно использовать, чтобы задать количество миллисекунд для объекта `Date` относительно 1 января 1970 года. Количество миллисекунд указывается в виде целого числа.

Date.setYear (5+)

Синтаксис:

```
объектDate.setYear(год,месяц,число)
```

Метод `setYear()` позволяет установить точную дату для объекта `Date`. Он принимает следующие аргументы.

Год — целое число, обозначающее год: от 0 до 99 — годы с 1900 по 1999; четырехзначные числа — годы с 1000 по 1899 и с 2000 и далее, трехзначные числа — годы до 1000 года н. э.

Месяц — число от 0 (январь) до 11(декабрь), обозначающее месяц.

Число — число от 1 до 31, обозначающее день месяца.

Date.toString (5+)

Синтаксис:

```
объектDate.toString()
```

Этот метод применяется для преобразования информации объекта `Date` в строку: день недели, месяц, число, часы, минуты, секунды, часовой пояс и год.

Объект Key (5+)

Синтаксис:

```
Key.метод()
Key.свойство
```

С помощью объекта `Key` контролируется нажатие клавиш в фильме. Он предоставляет великолепное средство для создания интерактивных элементов управления в фильме, которые вызываются с клавиатуры. Методы объекта `Key` применяются для идентификации нажатых клавиш, а его свойства используются при создании комбинаций клавиш быстрого вызова команд с клавиатуры. (Полный список кодов клавиш вы найдете в табл. С.4 в конце этого раздела.)

Методы:

`Key.getCode()` возвращает значение кода последней нажатой клавиши.

`Key.isDown()` проверяет, нажата ли в данный момент указанная клавиша.

Свойства

В табл. С.2 приведены свойства объекта `Key`.

Табл. С.2. Свойства объекта Key.

Синтаксис	Клавиша	Код
Key.BACKSPACE	Backspace	8
Key.CAPSLOCK	Caps Lock	20
Key.CONTROL	Control	17
Key.DELETEKEY	Delete	46
Key.DOWN	Down arrow	40
Key.END	End	35
Key.ENTER	Enter (цифровая клавиатура)	13
Key.ESCAPE	Esc	27
Key.HOME	Home	36
Key.INSERT	Insert	45
Key.LEFT	Left arrow	37
Key.PGDN	Page Down	34

Key.PGUP	Page Up	33
Key.RIGHT	Right arrow	39
Key.SHIFT	Shift	16
Key.SPACE	Spacebar	32
Key.TAB	Tab	9
Key.UP	Up arrow	38

Key.getCode (5+)

Синтаксис:

```
Key.getCode()
```

Данный метод возвращает значение кода последней нажатой клавиши. Коды клавиш — это стандартные значения, представляющие все клавиши в цифровом виде. (Полный список кодов клавиш вы найдете в табл. С.4 в конце справочника.)

Пример:

Следующая функция может применяться для того, чтобы извлечь значение кода клавиши и ее ASCII-значение.

```
keyCheck=new Object();
keyCheck.onKeyDown=function(){
   trace("Код: "+Key.getCOde());
   trace("ascii-значение: "+Key.getAscii());
}
Key.addListener(keyCheck);
```

Объект `keyCheck` идентифицирует нажатые клавиши или, как в данном случае, реагирует на событие `onKeyDown`.

Key.isDown (5+)

Синтаксис:

```
Key.isDown(код_клавиши)
```

В отличие от методов `Key.getAscii()` и `Key.getCode`, этот метод проверяет, удерживается ли нажатой клавиша, определенная аргументом *код_клавиши*. Аргумент *код_клавиши* может вводиться как цифровой код или как свойство объекта `Key` (например, `Key.SPACE`).

Пример

В приведенном сценарии представлена простая навигационная система клипа с использованием клавиш управления курсором.

```
onClipEvent(enterFrame){
   if(Key.isDown(Key.UP)){
      this._y-=10;
   }else if(Key.isDown(key.DOWN)){
      this._y+=10;
   }
   if(Key.isDown(Key.LEFT)){
      this._x-=10;
   }else if(key.isDown(Key.RIGHT)){
      this._x+=10;
   }
}
```

Обратите внимание на то, что код к клавишам перемещения курсора в вертикальном (со стрелками вверх и вниз) и горизонтальном направлениях (со стрелками влево и вправо) находится в отдельных блоках `if…else`. Это позволяет пользователю выполнять перемещение клипа в двух направлениях одновременно. Клип может перемещаться одновременно вверх или вниз *и* вправо или влево, то есть его можно перемещать по диагонали.

Объект Math (5+)

Синтаксис:

```
Math.свойство
Math.метод(выражение)
```

Объект `Math` позволяет создавать сценарии, в которых выполняются математические расчеты. При помощи методов этого объекта вы получаете доступ к математическим функциям; последние можно использовать для обработки чисел в фильмах.

Методы:

`Math.abs()` вычисляет модуль числа.

`Math.ceil()` округляет число до следующего большего целого числа.

`Math.floor()` округляет число до предыдущего меньшего целого числа.

`Math.max()` вычисляет большее из двух чисел.

`Math.min()` вычисляет меньшее из двух чисел.

`Math.pow()` возводит число в степень.

`Math.random()` возвращает произвольное число от 0 до 1.

`Math.round()` округляет число до ближайшего целого числа.

Math.abs (5+)

Синтаксис:

```
Math.abs(выражение)
```

Метод `Math.abs()` применяется для получения *абсолютной величины* `выражения`. *Абсолютная величина* действительного числа равна этому числу, если оно положительно, равна противоположному числу, если оно отрицательно, и равна нулю, если число равно нулю. Этот метод преобразует отрицательные значения в положительные, а положительные оставляет без изменений.

Math.ceil (5+)

Синтаксис:

```
Math.ceil(выражение)
```

Метод `Math.ceil()` преобразует число с плавающей точкой, указанное в `выражении`, в ближайшее большее целое число.

`Math.ceil(0.00001)` возвращает 1.

`Math.ceil(1.9)` возвращает 2.

Math.floor (5+)

Синтаксис:

```
Math.floor(выражение)
```

Метод `Math.floor()` преобразует число с плавающей точкой, указанное в *выражении*, в ближайшее меньшее целое число.

`Math.floor(0.00001)` возвращает 0.

`Math.floor(1.9)` возвращает 1.

Math.max (5+)

Синтаксис:

```
Math.max(число1,число2)
```

Метод `Math.max()` возвращает большее из двух чисел, указанных в аргументах *число1* и *число2*.

Math.min (5+)

Синтаксис:

```
Math.min(число1,число2)
```

Метод `Math.min()` возвращает меньшее из двух чисел, указанных в аргументах `число1` и `число2`.

Math.pow (5+)

Синтаксис:

```
Math.pow(число,exp)
```

При помощи этого метода *число* возводится в степень *exp*.

Math.random (5+)

Синтаксис:

```
Math.random()
```

Этот метод возвращает произвольное число, большее 0, но меньшее 1. Метод `Math.random()` применяется для произвольного выбора чисел. Эти числа затем можно использовать в функциях, которые произвольным образом размещают клипы на области действий и т. д.

Пример

Приведенное ниже выражение возвращает произвольное целое число от 0 до 10.

```
Math.floor(Math.random()*11);
```

Метод `Math.floor()` округляет результат до ближайшего меньшего целого числа. Поскольку метод `Math.random()` никогда не возвращает 1,то для того, чтобы в результате получить 10, мы умножаем число, возвращаемое этим методом, на 11, которое на единицу больше верхней границы диа-

пазона (0;10). Пусть, например, метод `Math.floor()` возвратил 0,999. Умножив это число на 11 и округлив до ближайшего меньшего целого, мы получим 10.

Точно так же можно определить как верхнюю, так и нижнюю границу для вычисления произвольного числа. В следующем сценарии возвращается произвольное число от 50 до 100, которое используется для установки значения свойства `_alpha`.

```
onClipEvent(load){
   vis=Math.floor(Math.random()*101)+50;
   this._alpha=vis;
}
```

Math.round (5+)

Синтаксис:

```
Math.round(выражение)
```

Этот метод применяется для округления числа с плавающей точкой до ближайшего целого числа. Если значение десятых в дроби равно или больше 0,5, выражение округляется до следующего большего целого числа; если значение десятых меньше 0,5, выражение округляется до ближайшего меньшего целого числа.

Пример

`Math.round(1.5)` возвращает 2.

`Math.round(1.4)` возвращает 1.

`Math.round(-1.5)` возвращает -1.

`Math.round(-1.6)` возвращает -2.

Объект Mouse (5+)

Синтаксис:

```
Mouse.метод
```

Объект `Mouse` позволяет отображать или не отображать указатель мыши в приложении Flash Player. Методы этого объекта обычно применяются для замены указателя мыши, используемого системой по умолчанию (стрелка), пользовательским указателем.

Методы:

`Mouse.hide()` — указатель мыши не отображается.

`Mouse.show()` — указатель мыши отображается.

Mouse.hide (5+)

Синтаксис:

```
Mouse.hide()
```

Этот метод помогает скрыть указатель мыши, находящийся на окне приложения Flash Player (в границах Flash-фильма). Если указатель мыши поместить за пределы области отображения кадров фильма, отображаться будет указатель, применяемый в системе по умолчанию.

Пример

Если с клипом связать следующий сценарий, этот клип станет настраиваемым указателем мыши. Следует воспользоваться событием `mouseMove`, чтобы обеспечить перемещение клипа, представляющего собой настраиваемый указатель, вместе с указателем мыши.

```
onClipEvent(load){
   Mouse.hide();
}
onClipEvent(mouseMove){
   this._x=_root._xmouse;
   this._y=_root._ymouse;
   updateAfterEvent();
}
```

Mouse.show (5+)

Синтаксис:

```
Mouse.show()
```

С помощью этого метода отображается указатель мыши, применяемый по умолчанию, после того как он был скрыт при помощи метода `Mouse.hide()`.

Объект MovieClip (5+/6)

Синтаксис:

```
экземпляр_клипа.метод
путь.экземпляр_клипа.метод
```

Объект `MovieClip` не предназначен для создания новых клипов. Во Flash для этого применяется команда **New Symbol** (Новый символ) меню **Insert** (Вставка), при этом переключатель типа символа устанавливается в положение **Movie Clip** (Клип). Объект `MovieClip` предоставляет целую серию методов для управления воспроизведением экземпляров клипов. Многие из этих методов подобны глобальным функциям, предназначенным для управления воспроизведением клипов и временной шкалой фильмов. Однако некоторые методы предоставляют дополнительные функциональные возможности, не поддерживаемые функциями ActionScript. Методы объектов `MovieClip` могут применяться как по отношению к экземплярам клипов, так и по отношению к временной шкале фильмов. Это относится и к главному фильму, и к SWF-файлам, которые загружаются на новый уровень фильма. Чтобы применить методы объекта `MovieClip` к клипу, свяжите с ним сценарий при помощи соответствующего обработчика событий `onClipEvent()`. Если нужно обеспечить управление клипом из другого элемента фильма, например из кнопки, экземпляру клипа должно быть присвоено уникальное имя. Это можно сделать на панели **Properties** (Свойства) либо при создании копии клипа, либо при перемещении его из библиотеки. Кроме того, часто приходится указывать путь к клипу, с которым вы хотите работать. Путь необходим, если этот клип находится на другом уровне (`_root`, `_level1`, `_level2` и т. д.) или вложен в другой клип. О способах получения доступа к клипам и временной шкале фильма можно узнать из главы 19.

Методы:

`MovieClip.attachMovie()` создает новый экземпляр клипа при помощи клипа, экспортируемого из библиотеки.

`MovieClip.duplicateMovieClip()` создает копию указанного клипа.

`MovieClip.getBytesLoaded()` возвращает количество байтов указанного фильма, которое было загружено.

`MovieClip.getBytesTotal()` возвращает размер указанного фильма в байтах.

`MovieClip.getURL()` загружает внешний файл, найденный по указанному URL.

`MovieClip.gotoAndPlay()` перемещает головку воспроизведения указанного фильма или экземпляра клипа к некоторому кадру его временной шкалы и начинает воспроизведение с этого места.

`MovieClip.gotoAndStop()` перемещает головку воспроизведения указанного фильма или экземпляра клипа к некоторому кадру его временной шкалы и останавливает воспроизведение на этом месте.

`MovieClip.hitTest()` возвращает логическое значение `true` или `false` в зависимости от того, находится ли в области воспроизведения указанного клипа некоторая координатная точка или область воспроизведения другого клипа. (Далее мы будем говорить — *пересекается* ли клип с другим клипом.)

`MovieClip.loadMovie()` загружает внешний SWF-файл в указанный экземпляр клипа.

`MovieClip.nextFrame()` переводит головку воспроизведения на один кадр вперед в указанном клипе или фильме и останавливает воспроизведение.

`MovieClip.play()` воспроизводит указанный клип или фильм.

`MovieClip.prevFrame()` переводит головку воспроизведения на один кадр назад в указанном клипе или фильме и останавливает воспроизведение.

`MovieClip.removeMovieClip()` удаляет экземпляр клипа, созданный при помощи метода `duplicateMovieClip()` или `attachMovie()`.

`MovieClip.startDrag()` делает экземпляр клипа перемещаемым.

`MovieClip.stop()` останавливает воспроизведение указанного клипа или фильма.

`MovieClip.stopDrag()` отменяет возможность перемещения клипа.

`MovieClip.unloadMovie()` выгружает внешний SWF-файл, доступ к которому был получен с помощью функции `loadMovie()` или метода `MovieClip.loadMovie()`.

`MovieClip.valueOf()` возвращает полный путь к указанному экземпляру клипа с использованием точечного синтаксиса записи пути.

MovieClip.attachMovie (5+)

Синтаксис:

```
экземпляр_клипа.attachMovie("id","имя",уровень)
```

Метод `attachMovie()` вызывает новый экземпляр клипа, указанного в аргументе *имя*, и размещает его на области действия в точке начала координат клипа, указанного в аргументе *экземпляр_клипа*. Точкой начала координат является левый верхний угол области отображения кадров главного фильма или точка регистрации экземпляра клипа. Чтобы вызвать новый клип, сначала необходимо экспортировать его из библиотеки, выбрав команду **Linkage** (Связь) меню **Options** (Параметры). Экспортируемое имя (указанное в окне **Linkage Properties** (Свойства связи)) используется в аргументе `id`, положение в главном фильме задается в аргументе *уровень*. Аргументы *id* и *имя* вводятся как строки, а аргумент *уровень* — как целое число.

Пример

Приведенный сценарий связан с фильмом, который экспортировался из библиотеки под именем `sun` и имеет имя экземпляра `yellow`. Фильм размещается на уровне главного фильма (`_root`) в левом верхнем углу области действия. В этом примере фильм `yellow` перемещается в центр области действия, имеющей размеры 550x400 пикселов.

```
_root.attachMovie("sun","yellow",2);
_root.yellow._x=275;
_root.yellow._y=200;
```

MovieClip.duplicateMovieClip (5+)

Синтаксис:

```
экземпляр_клипа.duplicateMovieClip("новое_имя",уровень)
```

Этот метод создает копию указанного экземпляра клипа (аргумент *экземпляр_клипа*), присваивает ей новое имя (аргумент *новое_имя*) и определяет ее положение в главном фильме (аргумент *уровень*). Аргумент *новое_имя* вводится как строка, а *уровень* — как целое число. Несмотря на некоторые отличия в синтаксисе, функциональные возможности этого метода идентичны возможностям функции `duplicateMovieClip()`.

Пример

См. описание функции `duplicateMovieClip()`.

MovieClip.getBytesLoaded (5+)

Синтаксис:

```
имя_фильма.getBytesLoaded()
```

Этот метод применяется, чтобы определить количество загруженных байтов для фильма, указанного в аргументе *имя_фильма*. Если вы хотите получить количество загруженных байтов главного фильма, этот аргумент можно не указывать, но неплохо бы указать путь `_root` или `this`. Поскольку клип считается частью фильма, в котором он находятся, нельзя узнать количество загруженных байтов для экземпляра клипа. Метод `MovieClip.getBytesLoaded()` предназначен для отслеживания процесса загрузки главных фильмов и SWF-файлов.

Пример

С помощью следующего сценария в поле `loaded` выводится количество килобайтов, соответствующее размеру загрузившейся части главного фильма. Чтобы получить значение в килобайтах, количество байтов, возвращенное методом `MovieClip.getBytesLoaded()`, было разделено на 1024.

```
loaded=Math.round(_root.getBytesLoaded()/1024)+"kB";
```

MovieClip.getBytesTotal (5+)

Синтаксис:

```
экземпляр_клипа.getBytesTotal()
```

Этот метод возвращает размер указанного фильма или экземпляра клипа в байтах.

Пример

С помощью следующего сценария в поле `total` выводится количество килобайтов, соответствующих размеру главного фильма (общий размер фильма в килобайтах). Чтобы получить значение в килобайтах, количество байтов, возвращенное методом `MovieClip.getBytesTotal()`, было разделено на 1024.

```
total=Math.round(_root.getBytesTotal()/1024)+"kB";
```

MovieClip.getURL (5+)

Синтаксис:

```
имя_получателя.getURL("имя_источника","окно","метод_передачи")
```

Этот метод объекта `MovieClip` выполняет те же функции, что и функция `getURL()`. Он позволяет передавать переменные из одного фильма (аргумент *имя_источника*) непосредственно в другой фильм (аргумент *имя_получателя*). Полное описание метода и пример к нему вы найдете в описании функции `getURL()`.

MovieClip.gotoAndPlay (5+)

Синтаксис:

```
экземпляр_клипа.gotoAndPlay(номер_кадра)
экземпляр_клипа.gotoAndPlay("метка_кадра")
```

Этот метод переводит головку воспроизведения указанного фильма или экземпляра клипа (аргумент *экземпляр_клипа*) в определенный кадр его временной шкалы и начинает воспроизведение с этого места. Вы можете определить нужный кадр, указав либо его номер (целое число), либо метку (строка).

MovieClip.gotoAndStop (5+)

Синтаксис:

```
экземпляр_клипа.gotoAndStop(номер_кадра)
экземпляр_клипа.gotoAndStop("метка_кадра")
```

Этот метод переводит головку воспроизведения указанного фильма или экземпляра клипа (аргумент *экземпляр_клипа*) в определенный кадр его временной шкалы и останавливает воспроизведение в этом месте. Вы можете определить нужный кадр, указав либо его номер (целое число), либо метку (строка).

MovieClip.hitTest (5+)

Синтаксис:

```
экземпляр_клипа.hitTest(x,y,значение)
экземпляр_клипа.hitTest("путь")
```

Этот метод проверяет, пересекается ли *экземпляр_клипа* с некоторой координатной точкой или другим экземпляром клипа, и возвращает соответствующее значение типа `Boolean`. Чтобы проверить наличие пересечения с точкой, необходимо указать ее координаты x и y и ввести логическое значение в аргументе *значение*. Если этот аргумент имеет значение `false` (это значение используется по умолчанию), пересечение будет определяться с учетом обрамления указанного экземпляра клипа. Если аргумент равен `true`, при определении пересечения будут учитываться только пикселы изображения экземпляра клипа. Аргумент *значение* следует задавать равным `true` при работе с клипами неправильной формы или с теми, в которых есть большие пустые участки между изображениями. Чтобы установить наличие пересечения экземпляра клипа с другим клипом, необходимо указать только *путь* к этому клипу как строковое значение.

Пример

В приведенном примере метод MovieClip.hitTest() используется для проверки наличия пересечения текущего клипа с экземпляром клипа border. Если они пересекаются (true), в поле display появится надпись "Пересечение"; в противном случае надпись будет отсутствовать.

```
onClipEvent(enterFrame){
   if(this.hitTest("_root.border")){
      _root.display="Пересечение";
   }else _root.display="";
}
```

MovieClip.loadMovie (5+)

Синтаксис:

```
экземпляр_клипа.loadMovie("URL","метод_передачи")
```

Этот метод выполняет те же функции, что и функция loadMovie(). Единственное его отличие заключается в том, что он не принимает аргумент *уровень*. Вместо этого метод объекта MovieClip загружает SWF-файл, на который указывает URL, в *экземпляр_клипа*. В разделе, посвященном функции loadMovie(), вы найдете соответствующий пример, а также узнаете о том, как можно передавать переменные при помощи необязательного аргумента *метод_передачи*.

MovieClip.nextFrame (5+)

Синтаксис:

```
экземпляр_клипа.nextFrame()
```

Метод MovieClip.nextFrame() переводит головку воспроизведения указанного экземпляра клипа или фильма (аргумент *экземпляр_клипа*) на один кадр вперед по временной шкале и останавливает воспроизведение.

MovieClip.play (5+)

Синтаксис:

```
экземпляр_клипа.play()
```

Метод MovieClip.play() применяется для воспроизведения указанного экземпляра клипа или фильма.

MovieClip.prevFrame (5+)

Синтаксис:

```
экземпляр_клипа.prevFrame()
```

Метод MovieClip.prevFrame() переводит головку воспроизведения указанного экземпляра клипа или фильма (аргумент *экземпляр_клипа*) на один кадр назад по временной шкале и останавливает воспроизведение.

MovieClip.removeMovieClip (5+)

Синтаксис:

```
экземпляр_клипа.removeMovieClip()
```

Этот метод объекта `MovieClip` выполняет те же функции, что и функция `removeMovieClip()`. С его помощью из области действий удаляется любой экземпляр клипа, созданный посредством функций `MovieClip.attachMovie()` и `MovieClip.duplicateMovie()`. Метод не принимает аргумент *имя_клипа*, вместо которого следует указывать путь к *экземпляру_клипа*, чтобы удалить его с области действия.

Пример

В данном сценарии клип `alert` удаляется из текущего фильма.

```
onClipEvent(mouseUp){
   if(this.hitTest(_root._xmouse,_root._ymouse,true)){
      _root.alert.removeMovieClip();
   }
}
```

MovieClip.startDrag (5+)

Синтаксис:

```
экземпляр_клипа.startDrag(блокирование,влево,вверх,вправо,вниз)
```

Как и функция `startDrag()`, метод `MovieClip.startDrag()` делает клип, указанный в аргументе *экземпляр_клипа*, перемещаемым, позволяя перетаскивать его мышью на области действия. Аргумент *блокирование* содержит значение типа `Boolean`, которое определяет, должна ли точка регистрации клипа при первом щелчке мышью переместиться и совпасть с указателем мыши (`true`) или остаться в своем исходном положении (`false`). Аргументы *влево*, *вверх*, *вправо*, *вниз* вводятся как целые числа и определяют координаты границ перемещения клипа. В главе 20 вы найдете пример использования функции `startDrag()` для создания бегунка.

MovieClip.stop (5+)

Синтаксис:

```
экземпляр_клипа.stop()
```

Метод `MovieClip.stop()` позволяет остановить воспроизведение указанного экземпляра клипа или фильма.

MovieClip.stopDrag (5+)

Синтаксис:

```
экземпляр_клипа.stopDrag()
```

Как и функция `stopDrag()`, метод `MovieClip.stopDrag()` отменяет возможность перемещения клипа в текущем фильме.

Пример

См. описание функции `startDrag()`.

MovieClip.unloadMovie (5+)

Синтаксис:

```
экземпляр_клипа.unloadMovie()
```

Этот метод позволяет выгрузить SWF-фильм, доступ к которому был получен с помощью функции `loadMovie()` или метода `MovieClip.loadMovie()`.

MovieClip.valueOf (5+)

Синтаксис:

```
экземпляр_клипа.valueOf()
```

Данный метод возвращает полный путь к указанному клипу, используя для его записи точечный синтаксис.

Методы рисования объекта MovieClip (6+)

Методы рисования для объекта `MovieClip` появились во Flash MX. Они предназначаются для динамического рисования клипов с помощью ActionScript. Дополнительную информацию об этих методах можно получить, обратившись к панели **Help** (Справка) (вызывается нажатием клавиши **F1** или в документации к Flash MX 2004.

`MovieClip.beginFill()` задает характеристики заливки и окрашивает выбранным цветом клип.

`MovieClip.createEmptyMovieClip()` создает новый пустой клип.

`MovieClip.curveTo()` создает кривую, имеющую характеристики (пунктирная, сплошная или какая-то другая, толщина, цвет и др.), установленные на текущий момент.

`MovieClip.endFill()` применяет заливку, использованную при последнем вызове метода `beginFill()`.

`MovieClip.lineStyle()` определяет характеристики линий.

`MovieClip.lineTo()` создает прямую линию, имеющую характеристики, установленные на текущий момент.

`MovieClip.moveTo()` задает исходную точку для создания линий или кривых.

MovieClip.beginFill (6+)

Синтаксис:

```
экземпляр_клипа.beginFill(rgb,alpha)
```

Этот метод применяется для заполнения цветом экземпляра клипа, указанного в аргументе *экземпляр_клипа*. Для заполняемого цвета необходимо указать два следующих элемента: шестнадцатеричное RGB-значение в формате 0xrrggbb (аргумент *rgb*) и значение от 0 до 100, определяющее степень прозрачности заливки (аргумент *alpha*).

MovieClip.createEmptyMovieClip (6+)

Синтаксис:

```
экземпляр_клипа1.createEmptyMovieClip("экземпляр_клипа2",уровень)
```

Этот метод используется на начальном этапе создания клипа с помощью ActionScript. Посредством метода `createEmptyMovieClip()` на временной шкале указанного клипа (аргумент *экземпляр_клипа1*) создается новый клип. При этом необходимо указать два аргумента: имя создаваемого экземпляра клипа (аргумент *экземпляр_клипа2*) и глубину уровня (аргумент *уровень*). На одном уровне главного фильма может быть только один клип, поэтому следует указывать незанятый уровень.

MovieClip.curveTo (6+)

Синтаксис:

```
экземпляр_клипа.curveTo(упрх,упру,x,y)
```

Метод `curveTo()` предназначен для создания в клипе кривых и дуг. Кривая линия рисуется, начиная с текущей точки, определенной последним вызовом метода `MovieClip.lineTo()`, `MovieClip.curveTo()` или `MovieClip.moveTo()`.Если не указаны иные координаты, кривая начинается в точке с координатами (0;0). Чтобы нарисовать кривую, нужно указать четыре следующих элемента: координаты x и y (*упрх*, *упру*) для управляющей точки, с помощью которой кривой придается ее форма, и координаты (*x* и *y*) для точки окончания кривой. Кривая будет иметь такие же характеристики, какие были установлены при последнем вызове метода `MovieClip.lineStyle()`.

MovieClip.endFill (6+)

Синтаксис:

```
экземпляр_клипа.endFill()
```

Этот метод применяет заливку, выбранную при последнем вызове метода `MovieClip.beginFill()`, к указанному экземпляру клипа.

MovieClip.lineStyle (6+)

Синтаксис:

```
экземпляр_клипа.lineStyle(толщина,rgb,alpha)
```

С помощью этого метода можно задать характеристики линий в ваших векторных изображениях. Толщина линии (аргумент *толщина*) определяется как целое число от 0 до 255. В аргументе *rgb* задается шестнадцатеричное RGB-значение цвета (в формате 0xrrggbb), которое определяет цвет линии. Степень прозрачности (аргумент *alpha*) определяется целым числом от 0 до 100. Все характеристики линий фиксируются и сохраняются до нового вызова метода `MovieClip.lineStyle()`.

MovieClip.lineTo (6+)

Синтаксис:

```
экземпляр_клипа.lineTo(x,y)
```

Метод `MovieClip.lineTo()` предназначен для создания в указанном экземпляре клипа прямых линий. Прямая линия начинается в текущей точке, определенной последним вызовом метода `MovieClip.lineTo()`, `MovieClip.curveTo()` или `MovieClip.moveTo()`. Если не указаны иные координаты, линия начинается в точке (0, 0). Для точки завершения необходимо задать соответствующие координаты в аргументах *x* и *y*. Итоговая линия начинается в текущей точке и заканчивается в точке с координатами (x;y). Линия имеет такие же характеристики, какие были установлены при последнем вызове метода `MovieClip.lineStyle()`.

MovieClip.moveTo (6+)

Синтаксис:

```
экземпляр_клипа.moveTo(x,y)
```

С помощью метода `MovieClip.moveTo()` задается исходная точка для создания линий или кривых. Координаты этой точки X и Y (аргументы *x* и *y*) задаются относительно точки регистрации указанного экземпляра клипа. После того как была задана исходная точка, текущая точка всегда будет определяться координатами точки завершения, установленными при последнем вызове метода `lineTo()` или `curveTo()`.

Пример

В этом примере с помощью ActionScript динамически воспроизводится эффект сумерек. В первом блоке создается черный квадрат, названный `sky` (небо). Обратите внимание на то, что если форма рисуется только с помощью метода `MovieClip.beginFill()`, в ней нет обрамления.

```
_root.createEmptyMovieClip("sky",1);
_root.sky.beginFill(0x000000);
_root.sky.moveTo(0,0);
_root.sky.lineTo(0,200);
_root.sky.lineTo(200,200);
_root.sky.lineTo(200,0);
_root.sky.lineTo(0,0);
```

Во втором блоке рисуются звезды. Они произвольно размещаются в прямоугольнике размером 190×190 и разворачиваются на произвольное количество градусов в пределах от 0 до 45 градусов.

```
for(i=2; i<<23; i++){
   var ptX=Math.floor(Math.random()*191);
   var ptY=Math.floor(math.random()*191);
   var spin=Math.floor(math.random()*46);
   _root.createEmptyMovieClip("star"+i,i);
   with(eval("_root.star"+i)){
      _rotation=spin;
      linestyle(2,0xcccc66,100);
      beginFill(0xcccc00)
      moveTo(ptX+7,ptY-7);
      lineTo(ptX,ptY-8);
      lineTo(ptX+5,ptY-13);
      lineTo(ptX+4,ptY-20);
      lineTo(ptX+10,ptY-20);
      lineTo(ptX+15,ptY-13);
      lineTo(ptX+20,ptY-8);
      lineTo(ptX+13,ptY-7);
      lineTo(ptX+10,ptY);
      lineTo(ptX+7,ptY-7);
   }
}
```

В последнем блоке сценария создается полумесяц. Для этого мы воспользовались методом `MovieClip.curveTo()`. Контур месяца пурпурный, а заливка светло-пурпурная.

```
_root.createEmptyMovieClip("moon",i);
_root.moon.linestyle(8,0x663366,100);
_root.moon.beginFill(0x666699);
_root.moon.moveTo(40,50);
_root.moon.curveTo(116,12,150,40);
_root.moon.curveTo(190,73,160,160);
_root.moon.curveTo(162,109,126,74);
_root.moon.curveTo(85,35,40,50);
```

Компоненты объекта Object (5+)

Синтаксис:

```
new Object()
```

С помощью объекта `Object` создаются объекты для временного хранения информации, то есть пользовательские объекты. Для создания пользовательского объекта применяется функция `new Object()`. Примеры создания таких объектов вы найдете в описаниях методов `Color.setTransform()` и `Sound.setTransform`.

Объект Sound (5+/6+)

Синтаксис:

```
new Sound()
new Sound("target")
```

Объект `Sound` предназначен для управления воспроизведением звука во Flash-фильмах. Для создания этого объекта воспользуйтесь функцией `new Sound()`. С ее помощью вы получите глобальный объект `Sound`, именуемый `globalSound`. Такой объект может контролировать все звуки фильма.

```
globalSound=new Sound();
```

В нижеприведенном выражении создается звуковой объект `bkgdSound` для воспроизведения звуков фильма, находящегося на уровне 1.

```
bkgdSound=new Sound("_level1");
```

В следующем выражении создается звуковой объект `clipSound` для воспроизведения звуков клипа `note`.

```
clipSound=new Sound("_root.note");
```

Создав объект `Sound`, вы можете воспользоваться его методами для регулировки громкости и позиции панорамирования звука. Кроме того, из библиотеки вашего фильма можно экспортировать отдельные звуки и связать их с определенными объектам `Sound` при помощи метода `attachSound()`. Эта техника обеспечивает управление всеми параметрами воспроизведения звука, в том числе определяет часть фильма, в которой начинается воспроизведение звука, и количество повторений последнего.

События

`Sound.onSoundComplete` возникает при окончании воспроизведения звука.

`Sound.onLoad` возникает при загрузке звука из внешнего источника во Flash-фильм.

Методы:

`Sound.attachSound()` связывает звук, извлеченный из библиотеки, с объектом `Sound`.

`Sound.getPan()` возвращает значение позиции панорамирования для указанного объекта `Sound`.

`Sound.getTransform()` возвращает доли (в процентах) стереоканалов звука, которые на данный момент воспроизводятся в левом и правом динамиках.

`Sound.getVolume()` возвращает уровень громкости для указанного объекта `Sound`.

`Sound.setPan()` присваивает значение позиции панорамирования указанному объекту `Sound`.

`Sound.setTransform()` задает доли (в процентах) стереоканалов звука, воспроизводящихся в левом и правом динамиках.

`Sound.setVolume()` задает уровень громкости для указанного объекта `Sound`.

`Sound.start()` воспроизводит звук, связанный с указанным объектом `Sound`.

`Sound.stop()` выключает звук, связанный с указанным объектом `Sound`.

Свойства:

`Sound.duration` возвращает полную возможную длительность воспроизведения указанного звука в миллисекундах.

`Sound.position` возвращает время в миллисекундах, в течение которого в действительности происходило воспроизведение звука.

attachSound (5+)

Синтаксис:

```
объектSound.attachSound("идентификатор")
```

Метод `Sound.attachSound()` служит для связывания звука, который вы экспортировали из библиотеки, с определенным объектом `Sound`. Для того чтобы экспортировать звук из библиотеки фильма, выберите его на панели **Library** (Библиотека), а затем выберите команду **Linkage** (Связь) меню **Options** (Параметры). Установите флажок **Export for ActionScript** (Экспортировать для ActionScript) и присвойте звуку имя в поле **Identifier** (Идентификатор). При вызове метода `Sound.attachSound()` это имя указывается в аргументе *идентификатор* как строковое значение (то есть в кавычках). Таким образом, звук, который вы экспортировали из библиотеки, будет связан с объектом `Sound`. Теперь с помощью объекта `Sound` вы можете воспроизвести звук, остановить его воспроизведение, а также выполнить с ним другие действия. Если в фильме на разных уровнях используется много звуковых объектов, то, чтобы управлять звуками каждого уровня, необходимо указывать соответствующие аргументы.

Пример

В приведенном примере звук `brass` связывается с новым звуковым объектом `fanfare`, после чего этот звук воспроизводится.

```
fanfare=new Sound();
fanfare.attachSound("brass");
fanfare.start();
```

Sound.duration (6+)

Синтаксис:

```
объектSound.duration
```

С помощью этого свойства извлекается общая продолжительность воспроизведения звука (в миллисекундах), ассоциированного с указанным объектом `Sound`.

Пример

В следующем примере два свойства, `position` и `duration`, используются для создания индикатора воспроизведения, отслеживающего, какая часть звука уже воспроизведена. По мере воспроизведения звука индикатор воспроизведения (первоначально имеющий свойство `_xscale` с нулевым значением) приближается к 100.

```
onClipEvent(load){
   bar_mc.xscale=0;
}
onClipEvent(enterFrame){
   played=_root.riff.position;
   total=_root.riff.duration;
   bar_mc._xscale=100*(played/total);
}
```

Sound.getPan (5+)

Синтаксис:

```
объектSound.getPan()
```

С помощью этого метода можно извлечь значение позиции панорамирования для указанного объекта `Sound`. Значение позиции панорамирования объекта измеряется в пределах от -100 (крайняя левая) до 100 (крайняя правая).

Sound.getTransform

Синтаксис:

```
объектSound.getTransform(объект1)
```

Используя этот метод, мы назначаем свойства `ll`, `lr`, `rl` и `rr` , которое содержатся в *объекте1*, звукам в указанном объекте `Sound` или звукам, ассоциированным с этим объектом. Эти свойства содержат доли (в процентах) левого и правого каналов стереозвука для соответствующих динамиков, заданные методом `Sound.setTransform()` для *объекта1*.

Sound.getVolume (5+)

Синтаксис:

```
объектSound.getVolume()
```

Этот метод возвращает текущий уровень громкости для указанного объекта `Sound`. Возвращенные значения находятся в пределах от 0 до 100.

Пример

Приведенная функция устанавливает уровень громкости звуков в фильме равным 10, если они воспроизводятся с уровнем громкости больше 10, и устанавливает максимальную громкость для звуков, громкость которых меньше 10.

```
function volToggle(obj){
   if(obj.getVolume()<=10){
      obj.setVolume(100);
   }else obj.setVolume(10);
}
```

Sound.onLoad (6+)

Синтаксис:

```
объектSound.onLoad=функция
```

Событие `Sound.onLoad` возникает тогда, когда звук, ассоциированный с объектом `Sound`, полностью загрузится в фильм посредством метода `Sound.loadSound()`. Чтобы использовать это событие для запуска дополнительных сценариев, задайте *функцию*, которая будет выполнять нужные выражения.

Пример

Это событие можно использовать при создании анимации, отображающей процесс загрузки фильма.

```
track1.onLoad=function(){
   track1.start(0,2);
   status.nextFrame();
}
```

Звук `track1` воспроизводится с момента начала загрузки звукового файла два раза, а клип `status` переходит к следующему кадру.

Sound.onSoundComplete (6+)

Синтаксис:

```
объектSound.onSoundComplete=функция
```

Это событие происходит тогда, когда заканчивается воспроизведение звука, связанного с указанным объектом `Sound`. Если объект воспроизводится циклически, событие `Sound.onSoundComplete` не будет возникать до тех пор, пока цикл не будет завершен. Чтобы использовать это событие для запуска дополнительных сценариев, задайте *функцию*, которая будет выполнять нужные выражения.

Пример

Возможности применения этого события очень разнообразны. Например, его можно использовать для воспроизведения следующей по списку песни.

```
track1.onSoundComplete=function(){
   track2.start();
}
```

По окончании воспроизведения звука `track1` начнет воспроизводиться звук `track2`.

Sound.position (6+)

Синтаксис:

```
объектSound.position
```

Данное свойство возвращает время (в миллисекундах), в течение которого происходило воспроизведение звука, связанного с указанным объектом `Sound`. Если объект `Sound` должен воспроизводиться циклически, его свойство `position` будет возвращать ноль в начале каждого повтора цикла. Пример использования этого свойства вы найдете в описании свойства `Sound.duration`.

Sound.setPan (5+)

Синтаксис:

```
объектSound.setPan(позиция_панорамирования)
```

С помощью этого метода задается значение позиции панорамирования звука или звуков, связанных с указанным объектом `Sound`. Значение позиции панорамирования звука может находиться в пределах от -100 (крайняя левая) до 100 (крайняя правая). Если задать крайнюю левую позицию панорамирования, звук будет воспроизводиться только левым динамиком. И наоборот, крайняя правая позиция означает воспроизведение только в правом динамике.

Пример

При каждом нажатии кнопки, связанной с этим сценарием, воспроизведение звука объекта `moveSnd` постепенно переводится на левый динамик.

```
on(press){
   movSnd.setPan(moveSnd.getPan()-10);
}
```

В данном сценарии мы неявно задаем значение позиции панорамирования. Мы получаем ее текущее значение при помощи метода `getPan()`. После этого метод `setPan()` задает новое значение позиции панорамирования, вычитая из значения старой число 10. С такой же легкостью можно переводить воспроизведение звука на правый динамик, прибавляя 10 к текущему значению позиции панорамирования.

setTransform (5+)

Синтаксис:

```
объектSound.setTransform(объект1)
```

Метод `Sound.setTransform()` относится к наиболее интересным методам ActionScript. Он позволяет динамически перераспределять компоненты левых и правых каналов стереозвука в левом и правом динамике. Используя этот метод, можно полностью перестроить стереосостав звука. Например, можно воспроизводить все звуки левого канала в правом динамике (так, чтобы ни один из них не воспроизводился в левом), а большую часть звуков правого канала — в левом динамике и только 25% из них — в правом.

Чтобы воспользоваться этим методом, необходимо сначала создать *объект1* — пользовательский объект со свойствами `ll`, `lr`, `rl` и `rr`. Эти свойства отвечают за распределение компонентов левых и правых каналов в левом и правом динамиках. Описание каждого свойства дано в табл. С.3.

Таблица С.3. Свойства объекта1

Свойство	Диапазон	Описание
ll	0—100	Доля звуков левого канала (в процентах), предназначенная для воспроизведения в левом динамике
lr	0—100	Доля звуков правого канала (в процентах), предназначенная для воспроизведения в левом динамике
rr	0—100	Доля звуков правого канала (в процентах), предназначенная для воспроизведения в правом динамике
rl	0—100	Доля звуков левого канала (в процентах), предназначенная для воспроизведения в правом динамике

Создав *объект1*, присвойте значение каждому из четырех его свойств и передайте их значения указанному объекту `Sound` при помощи метода `Sound.setTransform()`.

Sound.setVolume (5+)

Синтаксис:

```
объектSound.setVolume(уровень_громкости)
```

Этот метод устанавливает уровень громкости звука или звуков, связанных с указанным объектом `Sound`. Уровень громкости может устанавливаться в пределах от 0 до 100. Если уровень громкости установить равным 0, звук будет воспроизводиться, но не будет слышен. Можно подумать, что воспроизведение остановилось. Хотя Macromedia и не документировала этот факт, но фактически максимального уровня громкости для звуков не существует. Однако, если до импортирования во Flash-

фильм звуки были должным образом нормализированы, установка уровня громкости больше 100 единиц может вызвать их искажение.

Пример

В приведенном сценарии громкость звука объекта `movSnd` постепенно уменьшается при каждом нажатии кнопки.

```
on(press){
   movSnd.setVolume(movSnd.getVolume()-10);
```

Вместо того чтобы явно задать уровень громкости, мы извлекли его текущее значение при помощи метода `getVolume()`, а после этого посредством метода `Sound.setVolume()` установили новое значение уровня громкости, уменьшив текущее на 10. В этом сценарии можно с такой же легкостью усиливать громкость звука, прибавляя 10 к текущему значению уровня громкости. См. также пример в описании метода `Sound.getVolume()`.

Sound.start (5+)

Синтаксис:

```
объектSound.start(количество_секунд,цикл)
```

Этот метод применяется для воспроизведения звука, связанного с объектом `Sound` при помощи метода `Sound.attachSound()`. Метод `Sound.start()` принимает два необязательных аргумента: *количество_секунд* и *цикл*. В первом аргументе указывается количество секунд, через которое следует начать воспроизведение звука. Если в этом аргументе указать, например, число 1,5, то воспроизведение звука начнется через полторы секунды после начала воспроизведения звукового файла. По умолчанию данный аргумент имеет значение 0, при котором звук проигрывается с самого начала. В аргументе *цикл* вы задаете, сколько раз звук будет повторяться. Он может иметь значение 1 (значение по умолчанию), 2 и т. д., звук соответственно будет воспроизводиться один раз, два раза и т. д.

Пример

В следующем примере демонстрируется способ применения метода `Sound.start()`. Во-первых, необходимо создать объект `Sound` и связать с ним звук из библиотеки.

```
groove=new Sound();
groove.attachSound("drums");
```

Затем мы воспроизводим звук один раз с самого начала.

```
groove.start();
```

Вы также можете воспроизвести звук, начиная с 1,8 секунды.

```
groove.start(1.8);
```

Sound.stop (5+)

Синтаксис:

```
объектSound.stop("идентификатор")
```

Этот метод предназначен для отключения звуков в фильме. С его помощью можно задать, воспроизведение каких звуков должно быть остановлено. Если объект `Sound` является глобальным звуковым объектом, в фильме будут отключены все звуки. Если вы хотите отключить звук, связанный с объектом `Sound` при помощи метода `Sound.attachSound()`, укажите его имя (как строку) в аргументе *идентификатор*.

Пример

Поскольку объект `bkgd` является глобальным звуковым объектом, это выражение остановит воспроизведение всех звуков в фильме.

```
bkgd.stop();
```

В нижеприведенном выражении выключается только звук `drums`, связанный с объектом `groove`.

```
groove.stop("drums")
```

Объект TextField (6+)

Объект `TextField` позволяет манипулировать характеристиками полей, предназначенных для размещения динамического текста и текста ввода с помощью ActionScript. Для этого текстовому полю следует присвоить уникальное имя экземпляра. Щелкните на текстовом поле и откройте панель **Properties** (Свойства) (вызвав команду **Window ▸ Properties** или нажав комбинацию клавиш **Cmd/Ctrl+F3**). Введите имя в поле **Instance Name** (Имя экземпляра) панели **Properties** (Свойства). Это имя и будет именем экземпляра поля.

Свойства:

`TextField.hscroll` задает и возвращает текущее положение бегунка прокрутки горизонтали.

`TextField.maxhscroll` возвращает максимальное положение бегунка прокрутки по горизонтали.

`TextField.maxscroll` возвращает максимальное положение бегунка прокрутки по вертикали.

`TextField.scroll` задает и возвращает текущее положение бегунка прокрутки по вертикали.

TextField.hscroll (6+)

Синтаксис:

```
текстовое_поле.hscroll=выражение
```

С помощью этого свойства можно как задавать, так и получать значение, соответствующее положению бегунка прокрутки по горизонтали для объекта `TextField`. Если значение свойства `hscroll` равно 0, текст в поле выравнивается по левому краю. Пример использования данного свойства вы найдете в описании свойства `scroll`.

TextField.maxhscroll (6+)

Синтаксис:

```
текстовое_поле.maxhscroll=выражение
```

Это свойство возвращает значение максимально возможной правой позиции символов в текстовом поле. Если символ текста в этой позиции поля отсутствует, свойство возвращает значение 0. Вы не можете устанавливать значение этого свойства.

Пример

В этом сценарии текст в поле `line` переместиться в его (поля) конец, то есть последний символ последнего слова будет занимать крайнюю правую позицию поля.

```
on(press){
   edge=line.maxhscroll;
   line.hscroll=edge;
}
```

TextField.maxscroll (6+)

Синтаксис:

```
текстовое_поле=выражение
```

Это свойство объекта `TextField` возвращает номер верхней видимой строки, при котором в поле видна самая нижняя строка. Если в текстовом поле видно 5 строк, а всего их в нем 20, свойство `maxscroll` будет иметь значение 16, потому что если внизу видна строка 20, в самом верху поля будет видна строка 16. Вы не можете устанавливать значение этого свойства.

TextField.scroll (6+)

Синтаксис:

```
текстовое_поле.scroll=выражение
```

С помощью этого свойства объекта `TextField` можно как устанавливать, так и получать номер самой верхней строки, видимой в текстовом поле. Свойство применяется для создания интерактивных и анимированных полей с прокруткой.

Пример

Любой из приведенных примеров можно поместить в обработчик события `on(mouseEvent)` или `onClipEvent()`. Кроме того, вы можете воспользоваться объектом `MovieClip` или `Button` для прокрутки содержимого поля. Чтобы прокрутить поле вверх, воспользуйтесь следующим сценарием.

```
verField.scroll+=1;
```

Чтобы прокрутить поле вниз — другим сценарием.

```
verField.scroll-=1;
```

Для прокрутки поля в горизонтальном направлении (вправо) воспользуйтесь сценарием

```
horizField.hscroll+=10;
```

Чтобы восстановить исходное положение текста, воспользуйтесь сценарием

```
horizField.hscroll=0;
```

TextField.text (6+)

Синтаксис

```
textField .text="characters "
```

Это свойство объекта TextField определяет текст (слово, словосочетание или фразу), находящийся в соответствующем текстовом поле. Введите текст как строку или укажите переменную, которая содержит строковые данные.

Пример

```
var greeting="welcome ";
intro1.text=greeting;
```

В этом примере текст, хранящийся в переменной `greeting`, помещается в текстовое поле `intro1` .

Значения кодов клавиш для объекта Key

В табл. С.4 содержится полный список значений кодов клавиш для объекта `Key`.

Таблица С.4. Значения кодов клавиш

Буквенные клавиши	**Код**
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
Цифровые клавиши	**Код**
0	48
1	49
2	50
3	51
4	52

5	53
6	54
7	55
8	56
9	57
Клавиши цифровой клавиатуры	**Код**
клавиша 0	96
клавиша 1	97
клавиша 2	98
клавиша 3	99
клавиша 4	100
клавиша 5	101
клавиша 6	102
клавиша 7	103
клавиша 8	104
клавиша 9	105
знак умножения (*)	106
знак сложения (+)	107
Enter	13
знак вычитания (-)	109
знак десятичной дроби (.)	110
знак деления (/)	111
Функциональные клавиши	**Код**
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123
Прочие клавиши	**Код**
Backspace	8

Tab	9
Clear (Macintosh)	12
Enter	13
Shift	16
Control	17
Alt	18
Caps Lock	20
Esc	27
Пробел	32
Page Up	33
Page Down	34
End	35
Home	36
Стрелка влево	37
Стрелка вверх	38
Стрелка вправо	39
Стрелка вниз	40
Insert	45
Delete	46
Help	47
Num Lock	144
;:	186
=+	187
-_	189
/?	191
'~	192
[{	219
|	220
]}	221
'"	222