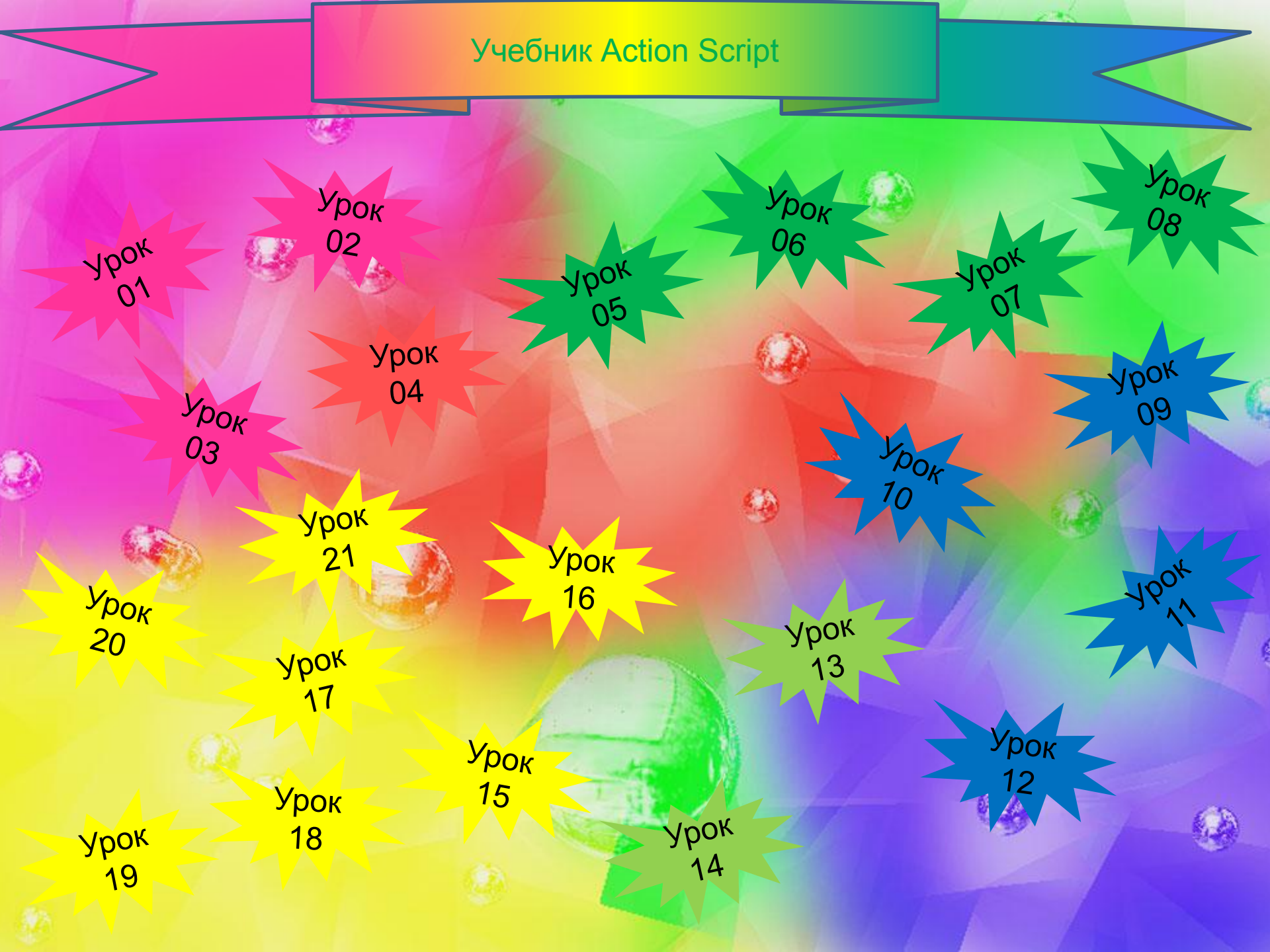
Учебник Action Script



Что такое код ActionScript

ActionScript - это язык программирования, используемый программой Flash MX. По сравнению с главной временной шкалой, позволяющей создавать лишь линейную анимацию, ActionScript расширяет возможности программирования. При помощи ActionScript ваш ролик будет реагировать, например, на выбор, делаемый пользователем, или на другие события. ActionScript позволяет управлять элементами, которые видит пользователь, и вместо простой анимации показывать нелинейные презентации, интерактивные приложения или игры.

ActionScript, используемый Flash MX, был создан на основе двух источников. Первый - это набор макрокоманд, взятый из предыдущих версий Flash, в основном Flash 4, который содержал схожий, но очень простой язык программирования. Другим источником оказался JavaScript - язык, использующийся для создания небольших программ для HTML-страниц в Internet Explorer и Netscape. Новый язык стал популярен среди разработчиков Web-страни, которые являются основными пользователями Flash MX. Для того чтобы упростить изучение ActionScript, многие новые команды и синтаксис были приведены к виду, напоминающему JavaScript.
Программы ActionScript представляют собой списки инструкций, которые выполняются программой Flash и могут быть помешены в различные места Flash-ролика. Если вы знаете, куда поместить сценарий, это уже полдела. Давайте рассмотрим несколько мест, куда может быть помещен сценарий, а также в каких случаях он будет использован программой.

Сценарии кадра

Вы можете помешать сценарии в ключевые кадры главной временной шкалы вашего ролика. Для этого выделите ключевой кадр на главной временной шкале и нажмите F9. На экране появится диалоговое окно Actions - Frame (Действия - Кадр).

Сценарии кадра могут содержать два типа элементов. Первый - это набор команд, исполняемых при воспроизведении кадра ролика. Команды исполняются друг за другом, пока не будет достигнут конец сценария.

Вторым типом элемента, включаемого в кадровый сценарий, является функция. Функции - это элементы кода, которые могут быть многократно использованы командами в сценариях кадра, а также другими сценариями ролика.

Сценарии для кнопок

Вы также можете задавать сценарии для кнопок. Прежде всего вам следует создать кнопку как элемент библиотеки. Затем, если необходимо, перетащите кнопку на рабочее поле. Выделите ее и нажмите F9 для того, чтобы вызвать диалоговое окно Actions. В случае, если диалоговое окно уже на экране, шенарий кнопки будет отображен при ее выделении.

Теперь окно Actions имеет заголовок Actions - Button (Действия - Кнопка). Любой сценарий, введенный здесь, будет выполнен кнопкой. Однако вы не можете просто ввести набор команд. Вы должны запрограммировать реакцию кнопки на выполнение различных событий, например на ее нажатие, помещение над ней курсора и т.д. Код, обрабатывающий подобные события, называется программой-обработчиком. Сценарий кнопки представляет собой набор из одного или нескольких обработчиков.

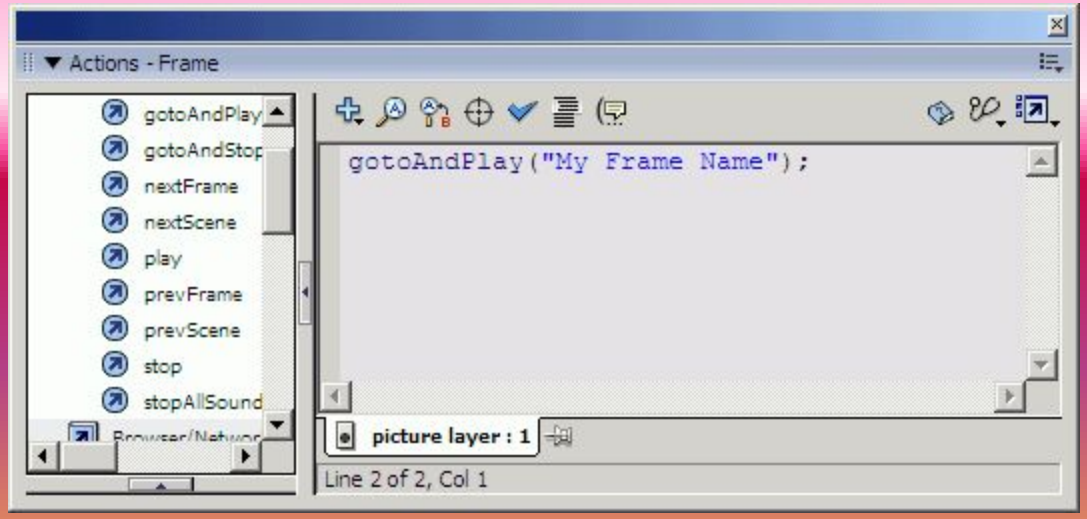
Сценарии клипов

Вы также можете назначить сценарий клипу. Для этого прежде всего создайте клип и сохраните его как элемент библиотеки. Затем поместите копию клипа на рабочее поле. Выделив копию клипа на рабочем поле, вызовите окно Actions, оно будет иметь заголовок Actions - Movie Clip (Действия - Клип).

Аналогично сценарию кнопки вы не можете просто ввести набор команд. Команды должны быть помешены в программы-обработчики, реагирующие на события клипов. Однако, в отличие от кнопок, клипы реагируют на совершенно другой набор событий. В следующих разделах мы рассмотрим пример

Использовани е окна Actions

Для программирования в ActionScript вы будете использовать окно Action-Script. При выборе команд, функций, операторов и других элементов синтаксиса в левой части окна элемент автоматически помешается в программу ActionScript в правой части окна.
Однострочная программа, содержащая команду gotoAndPlay, была помешена в программу в правой части окна двойным щелчком мыши по ее названию в списке слева. При выборе команды ее описание появляется под указателем мыши. В нашем примере был выбран gotoAndPlay(Frame);.



В вашем распоряжении имеется список команд левой части окна, которые могут быть добавлены в программу двойным щелчком мыши.

Ваша первая программа на ActionScript

Первая команда, которую вы изучите, - trace, которая предназначена для передачи информации в окно Flash Output. Окно Output представляет собой небольшое текстовое окно, выводимое на экран во время предварительного просмотра ролика во Flash. Обычно оно используется для вывода отладочных сообщений при разработке программы.

Хотя в действительности данная команда не будет применяться в законченном Flash-ролике, приводимый здесь пример наглядно демонстрирует, что при помощи ActionScript вы можете заставить Flash выполнить указанные вами действия.

Для начала создайте новый Flash-ролик. При этом автоматически создается временная шкала с одним слоем и одним ключевым кадром. Выделив его и нажав F9, вызовите окно Actions. Затем при помощи всплывающего меню в правом верхнем углу окна переключите его в экспертный режим.

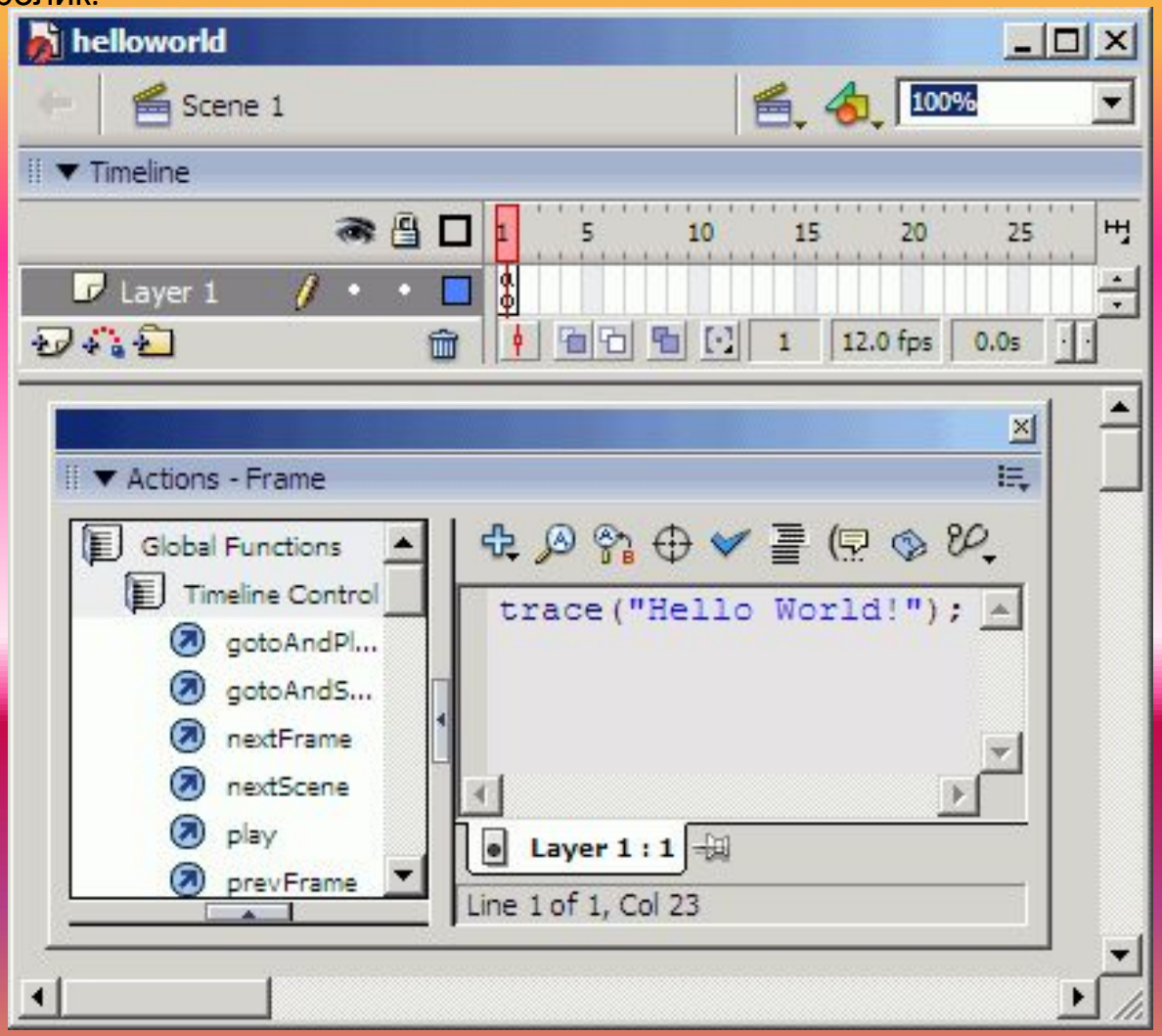
Теперь вы сможете поместить курсор в пустое программное поле в правой части окна Actions. Введите следующую строку:

```
trace ("Hello World!");
```

Точка с запятой ставится в конце каждой команды, как в примере с trace.

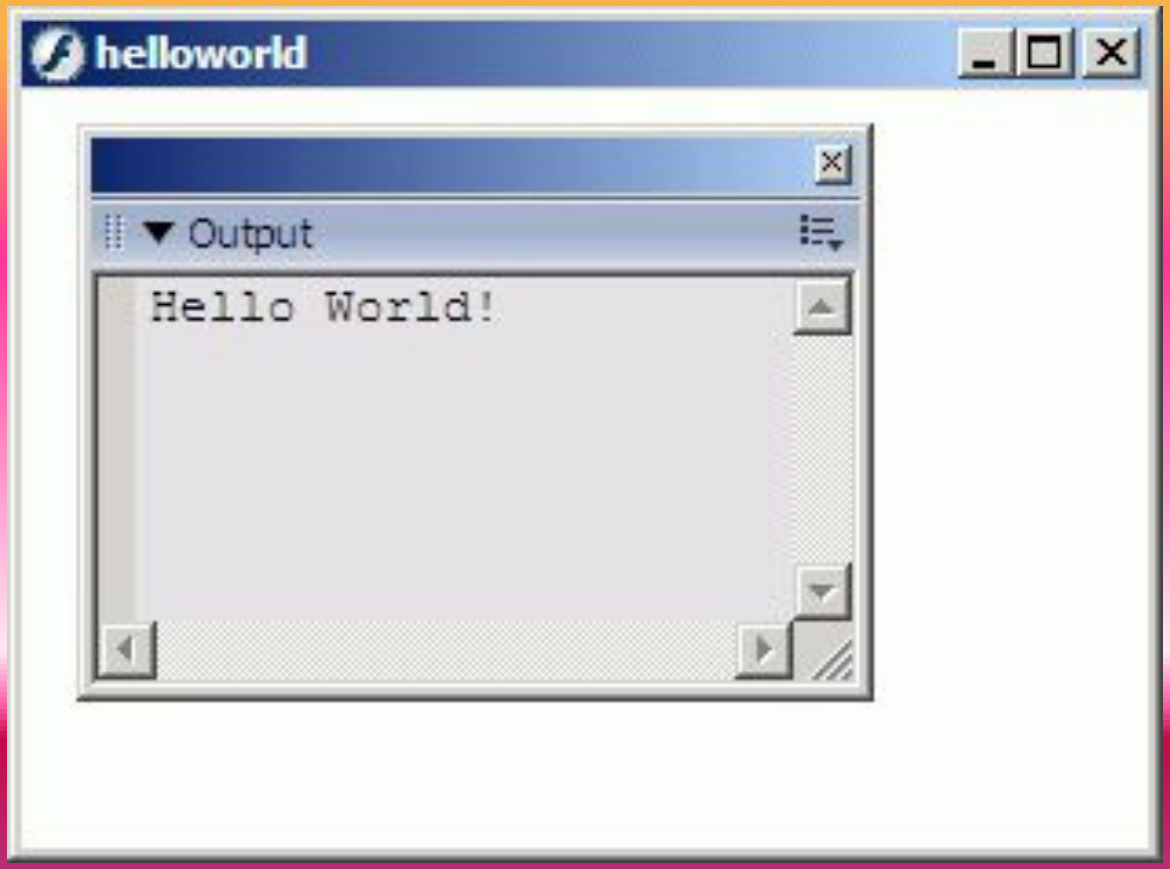
Команда trace - это встроенная функция Flash. В круглых скобках указываются параметры функции. Параметры - это входные данные, необходимые функциям для выполнения их задачи. Команда trace использует один параметр: строку, которая помещается в окно Output. Некоторые функции содержат

Ваш экран должен выглядеть примерно так, как изображено на этом рисунке. Здесь вы видите один слой и один кадр на временной шкале. Окно Actions имеет заголовок Actions - Frame. Это обозначает, что содержащийся в окне код ActionScript относится к выбранному в данный момент ключевому кадру. Единственным элементом программного листинга является команда trace. Попробуйте самостоятельно создать данный ролик.



Программа "Hello World" описывает первый и единственный кадр

Для того чтобы запустить программу, выберите команду Control -> Test Movie (Контроль -> Запустить пробное воспроизведение ролика). В течение секунды Flash создаст swf-файл и откроет его в окне предварительного просмотра. Окно останется пустым, так как в ролике нет никакой графики. Затем будет открыто окно Output со словами "Hello World!" . Это был важный шаг в изучении ActionScript. Мы дали Flash команду. В ответ Flash показал, что он понимает ее и слушается вас.



В окне Output содержится результат выполнения команды trace

Программа "Hello World" появилась вместе с первыми языками программирования и стала традиционной. Это самая элементарная программа, которая обычно приводится как первый пример при обучении любому языку программирования, будь то ассемблер громадной ЭВМ, Basic, Pascal, C, Java или ActionScript. Только что вы прошли по следам миллионов программистов, начинающих изучать свое ремесло.

Контроль воспроизведения ролика

В уроке 2 "Использование окна Actions" мы бегло ознакомились с командой ActionScript gotoAndPlay. Эта команда приказывает Flash не принимать во внимание следующий кадр временной шкалы, а вместо этого перейти к совершенно другому кадру. При помощи команды gotoAndPlay вы можете контролировать воспроизведение Flash-ролика.

Создайте новый Flash-ролик. В этот раз он будет содержать несколько кадров. Создайте последовательность из четырех ключевых кадров и присвойте им имена начиная с "parti" и заканчивая "part4". Эти кадры представляют собой четыре части анимации. Кроме этого, в каждый из четырех кадров следует поместить статический текст, сообщающий пользователю о том, какую часть он в данный момент видит на экране.

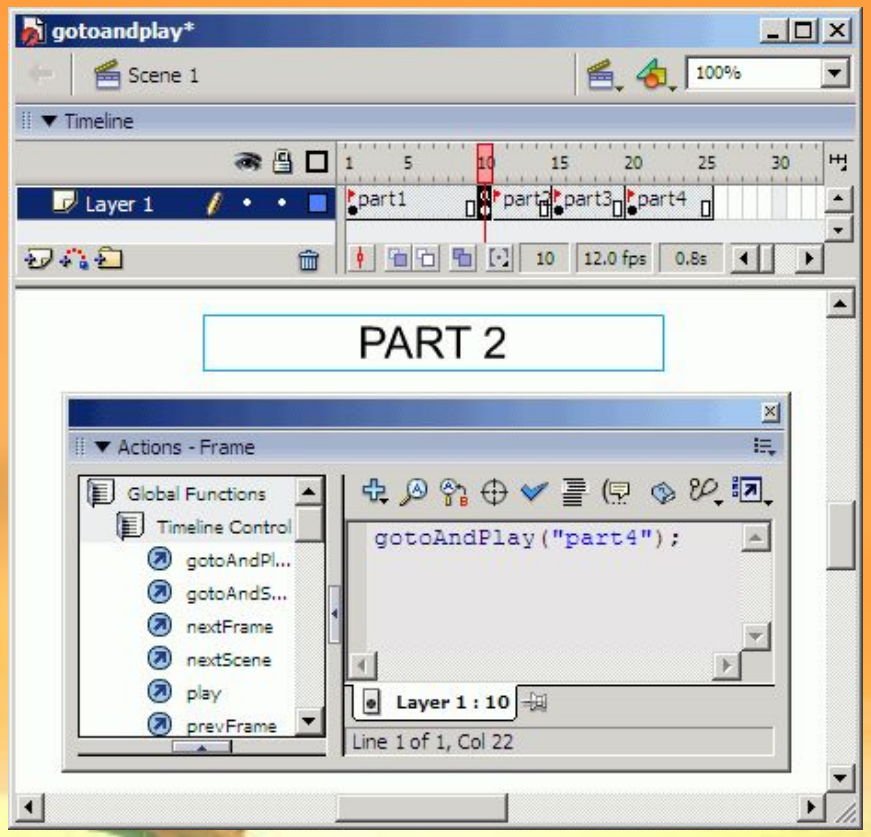
В примере Gotoandplay.fla в каждый из четырех ключевых кадров был помешен статический текст "PART 1"- "PART 4". Текст "PART 1" появится в ключевом кадре "part1", текст "PART 2" - в ключевом кадре "part2" и т.д.

Ключевые кадры разделены несколькими кадрами, чтобы были видны названия меток на главной временной шкале. На следущем рисунке будет показана главная временная шкала, содержащая четыре метки. Выбран второй ключевой кадр, поэтому на рабочем поле отображен текст "PART 2".

Выделив вт оманду:

```
gotoAndPlay ("part4");
```

Обратите внимание, что после добавления в ключевой кадр кода Action-Script на временной шкале в соответствующем кадре появляется строчная буква "а".



Ролик содержит четыре ключевых кадра и сценарий, помешенный во второй ключевой кадр. Выбран второй ключевой кадр, и в окне Actions отображается соответствующий сценарий

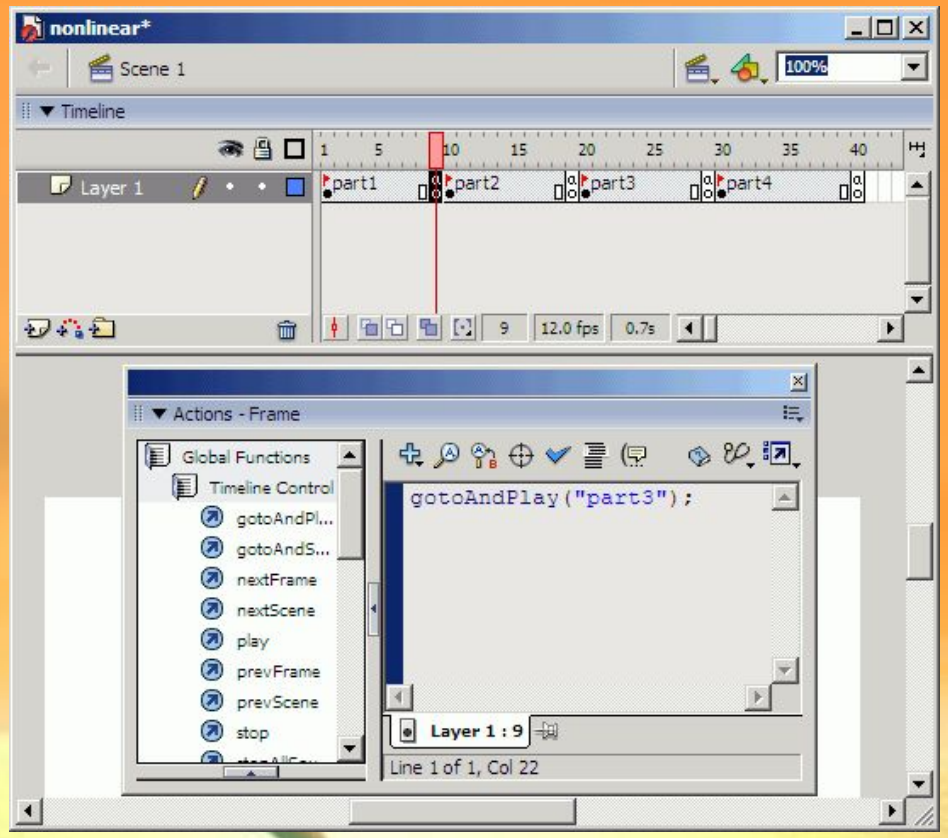
Это единственный код ActionScript в ролике. При воспроизведении ролик начинается с ключевого кадра "part 1" и на экране появится текст "PART 1". Затем ролик будет двигаться по временной шкале до ключевого кадра "part2", где находится код ActionScript. По команде Flash перейдет к ключевому кадру "part4'. При этом текст "PART 2" не успеет появиться на экране1. Ролик будет находиться в кадре "part4", на экране окажется текст "PART 4".

Ролик продолжит движение по временной шкале до конца. Затем возвратится к первому кадру и опять покажет текст "PART 1". Этот цикл повторяется бесконечно.

При помощи ActionScript вы заставили Flash отступить от последовательного воспроизведения анимации. Ролик должен был последовательно проигрываться с кадра "part 1" к "part2", затем к "part3" и к "part4" однако по вашему сценарию он пропустил сразу два кадра - "part 2" и "part 3'. С ActionScript вы получаете возможность

Точнее говоря, если в кадре имеется сценарий, программа сначала исполняет сценарий, а потом прорисовывает кадр.
Давайте рассмотрим другой пример. По той же схеме создайте небольшие ключевые кадры в конце каждой части ролика. Мы будем создавать сценарии, воспроизводимые не в начале каждой части, а в ее конце.

На этом рисунке показано устройство ролика, включающего четыре маркированных ключевых кадра, как и в предыдущем примере, а также четыре ключевых кадра в конце каждой части ролика. В этих немаркированных кадрах содержится код



Ролик состоит из четырех частей. Каждая часть начинается с маркированного ключевого кадра и заканчивается немаркированным ключевым кадром, содержащим небольшой сценарий

Ключевой кадр, следующий за "part1", содержит код:

gotoAndPlay ("part3");

Ключевой кадр, следующий за "part2", содержит код:

gotoAndPlay ("part4");

Ключевой кадр, следующий за "part3", содержит код:

gotoAndPlay ("part2");

Ключевой кадр, следующий за "part4"', содержит код:

gotoAndPlay ("part1");

При запуске ролика из файла Nonlinear.fla на экране появляется надпись "PART 1". Затем ActionScript задает переход ролика не к кадру "part2", а к "part3". Таким образом, ролик переходит от надписи "PART 1"к "PART 3". Затем, после кадра "'part3", ролик возвращается к "part2", а после него к "part4". Весь ролик воспроизводится в таком порядке - "PART 1", "PART 3", "PART 2" и "PART 4". Эта последовательность затем повторяется.

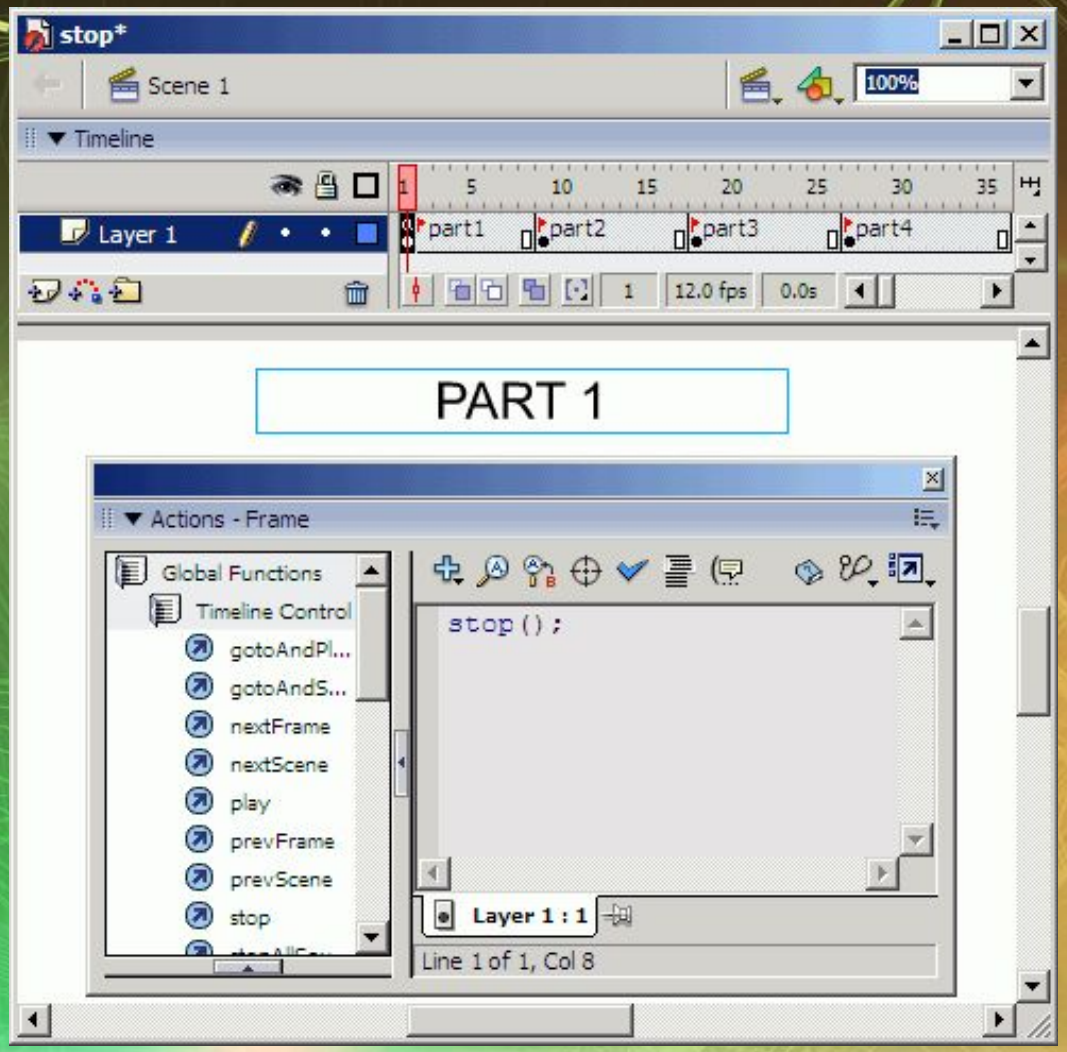
При помощи ActionScript вы полностью заменили обычную анимационную последовательность на свою собственную.

Приведенные выше примеры дают предсказуемый результат, однако у вас есть возможность использовать команду gotoAndPlay по-другому, например в сочетании с кнопками, чтобы предоставить пользователю

Создание кнопок для пользователя

В предыдущем примере мы рассмотрели возможность управления роликом при помощи ActionScript без всякого вмешательства пользователя. Давайте добавим несколько кнопок, щелкнув по которым, пользователь сможет запустить воспроизведение той или иной части ролика.

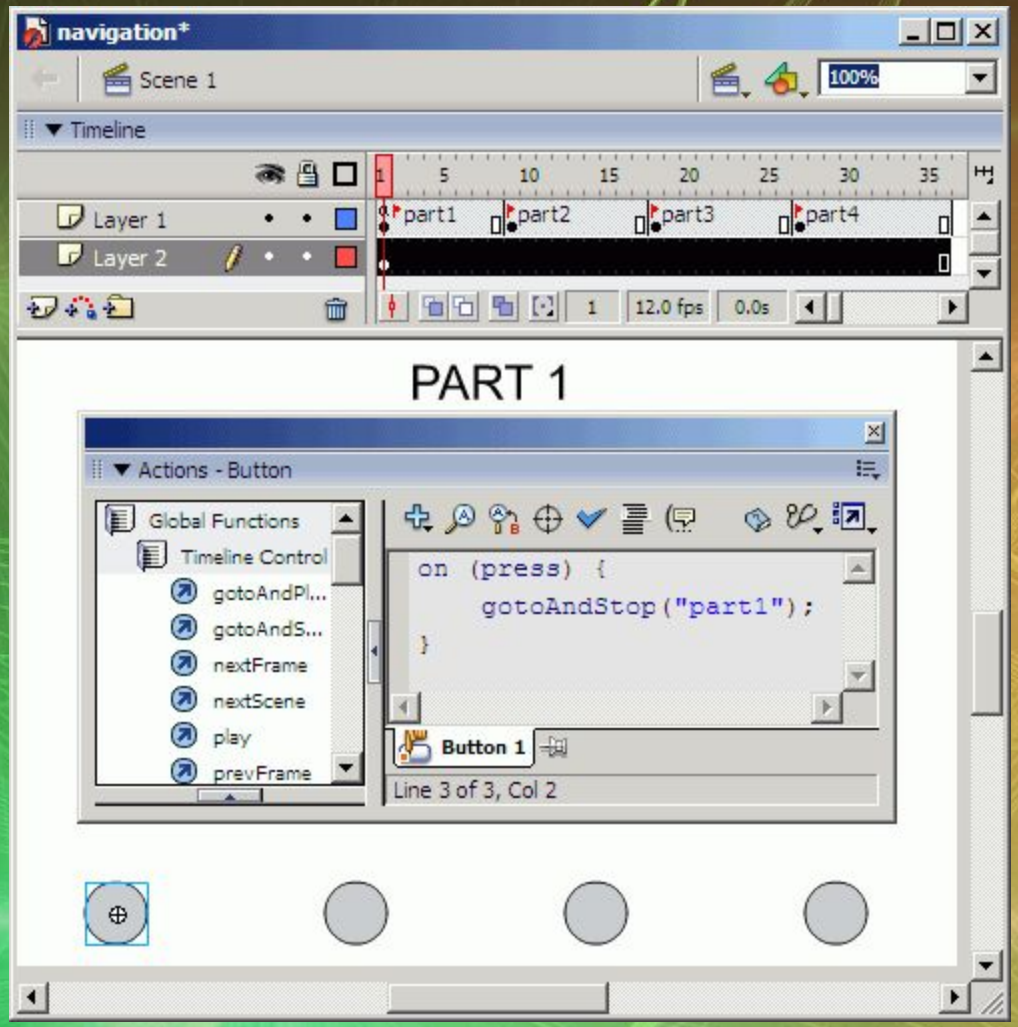
Предоставить пользователю больше контроля над воспроизведением ролика можно, прежде всего лишив этого контроля Flash. Flash начинает воспроизведение анимации с первого кадра, затем переходит к следующему и т.д. При помощи команды stop () вы можете остановить воспроизведение анимации на первом кадре.



Ролик показанный на этом рисунке состоит из четырех кадров, каждому из которых соответствует метка на временной шкале и текст на рабочем поле. В первый кадр включен небольшой сценарий с командой stop(). Воспроизведение ролика начинается и заканчивается в кадре 1.

Теперь мы можем предоставить немного самостоятельности пользователю. Создайте простой эталон кнопки и поместите его на рабочий стол. Не следует выдумывать ничего необычного. В примере кнопка представляет собой маленький кружок.

Следующим вашим шагом будет создание нового слоя с одним кадром. Теперь в ролике будут участвовать одни и те же элементы. В данный слой мы поместили четыре различных копии одной и той же кнопки из библиотеки. Результат изображен на рисунке:



Данный ролик состоит из двух слоев. Первый слой разделен на четыре части, которые содержатся в четырех кадрах, второй включает один ключевой кадр.

В первом кадре данного ролика будет использоваться та же самая команда stop(), которая остановит воспроизведение ролика сразу после его начала. В нижней части экрана размешены четыре кнопки. Каждой из них будет приписан отдельный сценарий. Первый сценарий изображен на предыдущем рисунке. Первая кнопка выбрана,и в окне ActionScript отображается ее сценарий:

```
on (press){
gotoAndStop("part1");
}
```

Так выглядит обычный сценарий кнопки. Поведение кнопки во время выполнения различных действий с ней задается программами-обработчиками. В нашем случае это действие press. Синтаксический элемент on обозначает начало программы-обработчика. В фигурные скобки заключаются команды реакции на события.

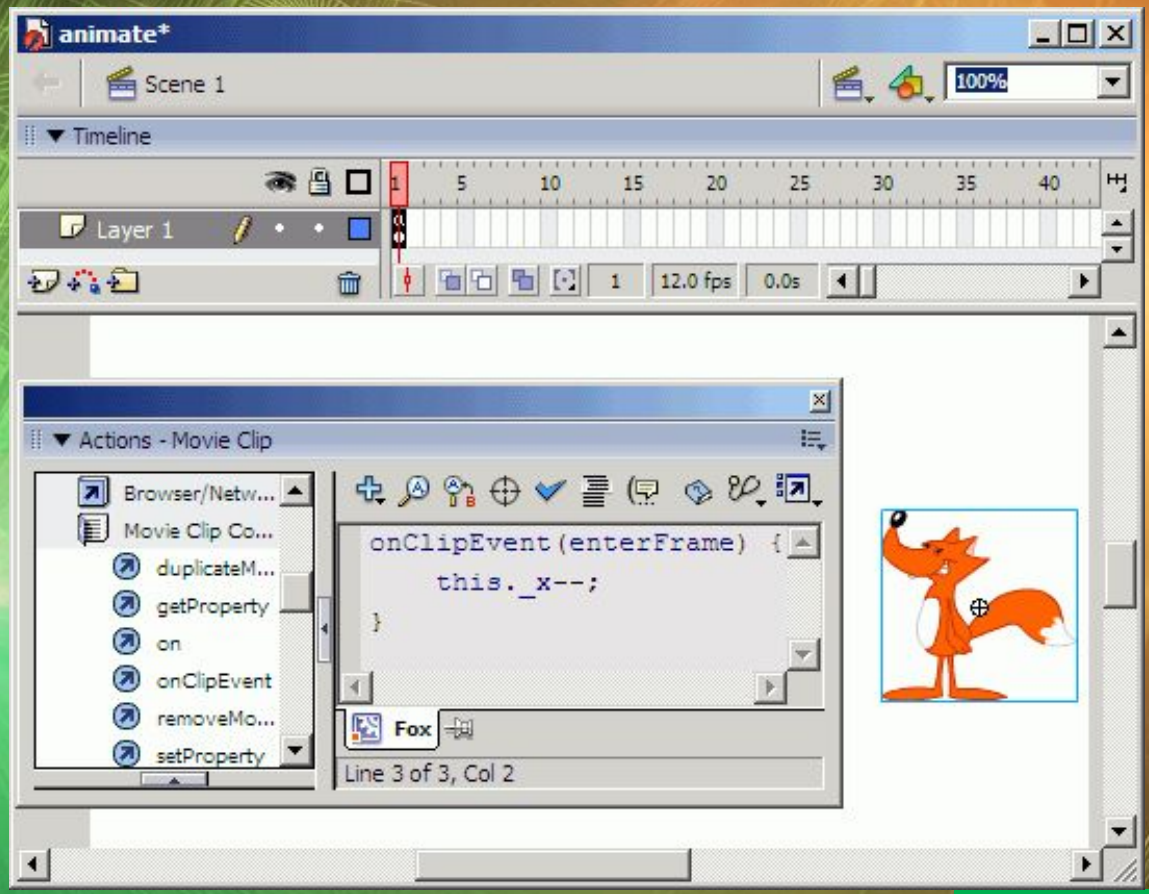
В нашем примере используется команда gotoAndStop - разновидность команды gotoAndPlay. В отличие от первой команды, задающей переход к новому кадру и остановку воспроизведения ролика, вторая команда переходит к другому кадру и запускает его воспроизведение.

Помимо этого сценария, три подобных приписаны трем другим кнопкам. Единственным их отличием является то, что они задают переход к кадрам "part 2", "part 3" и "part 4".

Анимация при помощи ActionScript

Теперь научимся перемешать предметы по рабочему полю при помощи ActionScript. Необходимо будет назначить клипу сценарий подобно тому, как мы сделали это для кнопки на нашем предыдущем занятии.

Когда употребляется термин "ролик" (movie), речь идет обо всем файле проекта. Термин "клип" употребляется для символа типа movie clip.
Вы можете поместить на рабочее поле экземпляр любого клипа, имеющегося в библиотеке. На данный момент наш фильм состоит из одного слоя и одного кадра, помешенного на рабочее поле



Сценарий, назначенный клипу, использует программу-обработчик подобно сценарию кнопки в предыдущем занятии. Чтобы задать программу обработки, вместо командной строки on введите строку onClipEvent, которая определяет события для клипов ролика. В данном случае событие будет следующим: enterFrame. Это автоматическое событие, происходящее при каждом обращении к данному кадру. Если ролик должен воспроизводиться со скоростью 12 кадр/с, оно должно посылаться в программу обработки 12 раз в секунду.

Если ролик остановлен командой stop () или просто состоит из одного кадра, то кадр "зацикливается", то есть перерисовывается с той скоростью, которая задана ролику Подобно программе on (press) в сценарии кнопки, команды реакции на событие onClipEvent (enterFrame) заключаются в фигурные скобки. В данной программе будет только одна команда, перемещающая клип на один пиксель влево. Давайте взглянем на сценарий и проанализируем, какие действия в нем выполняются.

```
OnClipEvent(enterFrame) {
this._ x -- ;
}
```

Команда this._x --, вероятно, будет совершенно непонятной для людей, не знакомых с языками программирования, поэтому разберем ее по частям.

Команда this обозначает обращение команды к объекту, который ее содержит. В данном случае this используется для обращения к клипу, которому назначен этот сценарий.

За командой this следует точка, обозначающая обращение к свойству объекта. В нашем примере _x относится к горизонтальному положению клипа.Итак, this._x определяет горизонтальное положение клипа. Символ -- является декрементом (командой уменьшения значения). Он уменьшает значение стоящей перед ней величины на 1 (пункт). Таким образом, команда this._x -- берет значение горизонтального положение клипа и вычитает 1, благодаря чему клип перемещается влево.

Если клип необходимо переместить вправо, используется команда ++, которая называется инкрементом (это команда увеличения значения). Если вы хотите переместить клип вправо или влево сразу на несколько пикселов, используйте соответственно += или -=:

Возможность управления клипами пользователем

А теперь давайте сделаем так, чтобы клип двигался вслед за курсором.
В предыдущем занятии мы узнали, как получить доступ к горизонтальной позиции клипа. При помоши параметра _y так же легко оперировать изменением координаты по вертикали. Теперь все, что нам нужно, - это привязать данные координаты к курсору.
Местоположение курсора определяется двумя параметрами: _xmouse и _ymouse, которые представляют собой координаты курсора по вертикали и горизонтали. Возникает следующий вопрос: чьи это параметры?
Это могут быть параметры любого клипа или самого ролика. Например, команда this ._xmouse определяет горизонтальное положение мыши по отношению к центру текущего клипа.
Нам нужно установить положение клипа по отношению к рабочему полю. Для того чтобы получить параметры рабочего поля, вместо идентификатора this следует использовать _root. Так, переменная _root ._xmouse определяет горизонтальное расположение мыши относительно левого верхнего угла рабочего поля.
Для того чтобы привязать координаты клипа к координатам курсора, необходимо изменить параметры _x и _y клипа в соответствии с параметрами _xmouse и _ymouse рабочего поля. Вот как будет выглядеть программа:

PHOBOS IMPACT

```
On.ClipEvent (enterFrame) {
this._x = _root._xmouse; this._y =
_root._ymouse;
}
```

На этом рисунке показан "собранный" ролик Usercontrol.fla. Он состоит из одного кадра со скриптом, содержащим команду stop(). Клипу назначен вышеописанный сценарий. Это все, что требуется. При воспроизведении ролика клип с лисой будет следовать за курсором.

Использовани
е переменных

Переменные представляют собой поименованные области памяти, содержащие какую-либо информацию, такую как числа или последовательность символов. Например, вы можете создать переменную и поместить в нее число 5. Если вы прибавите к данной переменной число 2, она будет содержать число 7. Вот как эти действия будут выглядеть в ActionScript:

```
myVariable = 5;
myVariable += 2;
```

myVariable - это имя созданной вами переменной. Для того чтобы создать переменную во Flash, необходимо просто один раз упомянуть ее в тексте программы. Flash автоматически создает переменную с соответствующим именем (в ActionScript не требуется какой-либо специальной инициализации переменных). В нашем примере была создана переменная myVariable со значением 5. В следующей строке к данной переменной прибавляется значение 2. Теперь переменная myVariable содержит число 7.

Очень важно присвоить переменной содержательное имя. Оно должно отражать выполняемую ей задачу. Например, если переменная предназначена для хранения имени пользователя, ее лучше назвать, допустим, "userName". Позднее, просматривая код, вы сразу поймете, для чего используется данная переменная.

В переменную можно также помещать строки. Строка - это последовательность символов наподобие “Hello World”. Так же как и число, строку можно сохранить в переменной при помощи **символа =. myVariable = "Hello World";**

Переменные во Flash не делятся на типы. Классификация переменных -это ограничение типа данных, хранимых в данной переменной, которое применяется в других языках программирования. Это означает, что если переменная предназначена для хранения чисел, она может использоваться только для хранения чисел, но не для хранения строк. В ActionScript нет такого ограничения, поэтому любая переменная может хранить любые объекты, например числа или строки.
Помимо чисел и строк переменные могут содержать другие элементы, например указатель клипа. Вспомните, как в двух предыдущих уроках использовался элемент this для ссылки на текущий клип. Вы можете присвоить значение this переменной и с ее помощью ссылаться на клип.
Использование подобных элементов помогает управлять роликами, где содержится более чем один клип.
Вы будете часто сталкиваться с подобными ситуациями в играх, рассматриваемых в этой книге. На данный же момент вам достаточно знать, что переменные могут содержать числа, строки и ссылки на клипы.

Все переменные, использованные в этом уроке и в остальной книге, это глобальные переменные. Это значит, что после того как вы их задали, к ним можно обращаться из ролика или других клипов. Также с помощью ключевого слова "var" вы можете определить локальные переменные. Локальные переменные, определенные в какой-либо функции, недоступны к использованию в остальном коде. Как локальную переменную, например, удобно определять счетчик цикла, используемого внутри функции.

Выполнение операций

Вам наверняка понадобится изменять данные, хранящиеся в переменной. Мы уже рассматривали, как с помощью команд ++ или += изменять значение переменной. В вашем распоряжении также имеется большой набор других операций.
Давайте начнем с переменных, содержащих числа. Вы можете выполнять большое количество математических операций с числами при помощи символов +, -, / и *. Приведем несколько примеров. Допустим, у вас есть две переменные: a и Ь. Вы можете сложить их и поместить результат в переменную c.

```
a = 7;
b = 5;
c = a + b;
```

В предыдущем примере c будет в результате содержать число 12. Вот еще один подобный пример

```
c = a - b;
c = a * b;
c = a / b;
```

Переменная с получит значение 2 в первой строке, 35 - во второй строке и 1,4 - в третьей строке.
Вы также можете выполнять более сложные математические операции при помощи специальных функций, встроенных во Flash. Все они содержатся в объекте Math, за которым следует точка и имя функции. Например, при помощи функции Math.sqrt можно вычислить квадратный корень числа:

```
a = 9;
b = Math.sqrt(a);
```

Значением b является 3 - квадратный корень из 9.
Один из важнейших элементов компьютерного языка, позволяющий программистам создавать игры, - генератор случайных чисел. Без случайных чисел игры были бы полностью предсказуемыми и скучными.
Flash генерирует случайные числа при помощи функции Math. random (). При этом генерируются случайные десятичные значения между 0.0 и 1.0. Нужные числа получают, умножая данные значения на целое число, а затем переводя их в целое число при помощи функции int. Например, следующая строка служит для генерации случайного числа от 0 до 9:

```
myRandomNumber = int(Math.random()*10) ;
```

Другие математические функции будут рассмотрены по мере их появления в примерах.
Функции могут применяться и к строкам. Например, для объединения двух строк может использоваться символ +:

```
a = "Hello";
Ь = "World";
c = a + b;
```

В итоге переменная с превратится в строку "HelloWorld".

Условные выражения

Из урока 5 "Создание кнопок для пользователя" вы узнали, как пользователь может использовать кнопки для перехода от одного кадра к другому. Пользователь решает, какую часть ролика он хочет увидеть следующей, щелкает по кнопке, и сценарий, приписанный данной кнопке, отправляет ролик в соответствующий кадр.
ActionScript также может принимать решения. Эти решения основываются на сравнении значений, которое осуществляется при помощи оператора if. Например, могут сравниваться два значения. Если они равны, программа отреагирует на это определенным образом.
Приведем пример программы, которая сравнивает переменную со значением. Если значение удовлетворяет условию, код внутри фигурных скобок выполняется. В противном случае программа пропускает его.

```
if (a == 7) {
GotoAndPlay("special frame");
}
```

Символ == используется для установления тождественности двух значений. Если значения равны, условие верно. Если нет, условие ложно. Задача оператора if состоит в проверке верности условия. Если это условие верно, код в фигурных скобках выполняется.

Вы можете продлить выражение if и задать, что при невыполнении данного условия должно совершаться определенное действие. Используя оператор else после оператора if, вы можете включить еше один набор фигурных скобок, в которых будет задано какое-либо действие при невыполнении условия:

```
if (a =- 7) {
gocoAndPlay("special frame");
} else {
gotoAndPlay("another frame");
}
```

Выражение if может быть длинным и включать несколько проверок условий. Выполняется код, следующий за первым верным условием. Если ни одно условие не является верным, выполняется код, следующий за оператором else.

```
if (a == 7) {
gotoAndPlay ("special frame");
} else if (a == 12) {
gotoAndPlay("very special frame");
} else if (a == 15) {
gotoAndPlay(" extremely special frame");
} else {
gotoAndPlay("a not so special frame");
}
```

Символ == может также использоваться для сравнения двух строк. Например, для того чтобы определить, содержит ли переменная username строку "Gary", вы можете использовать следующий код:

```
if (username == “Dima") {
```

Кроме символа ==, определить, будет ли одно число соответственно меньше или больше другого, помогут символы < и >, а символы <= и >= указывают, будет ли число меньше или равно или больше или равно другому числу. Данные символы могут использоваться и при сравнении строк, в этом случае они производят сравнение исходя из алфавитного порядка.
У вас есть возможность одновременно выполнять несколько сравнений. Например, вы можете проверить, равно ли значение a определенному числу, a username определенной строке:

```
if ((a == 7) and (username == “Dima")) {
```

Предыдущая строка верна только при выполнении обоих условий. Выполнение одного из условий можно проверить при помощи оператора or:

```
if ((a == 7) or (username == “Dima")) {
```

Выражения if являются основными элементами всех программ. Они могут использоваться для изменения переменных, перехода к другим кадрам, а также для того, чтобы определить выигрыш или проигрыш игрока. Другим важным элементом программы являются Циклы, которые рассматриваются в следующем разделе.

Циклы

Условные выражения являются необходимым элементом программирования; не менее важный элемент - циклы. Компьютеры превосходно выполняют повторяющиеся задания. Во Flash этими заданиями являются циклы. Наиболее распространенная их разновидность - цикл for. Он позволяет выполнять задание определенное количество раз. Приведем пример цикла for:

```
for(i=0;i<10;i++) trace(i);
```

Традиционным для программирования является использование переменной i в качестве счетчика циклов программы. В данном случае i обозначает increment (приращение). Это неплохая идея - всегда давать переменной имя, обозначающее выполняемую ей функцию. В циклах for, как правило, используется именно переменная i.
На первый взгляд цикл for кажется довольно сложным. Он состоит из трех частей, разделенных точкой с запятой. Первая часть позволяет задавать определенное значение переменной. В данном случае переменной i присвоено значение 0.
Вторая часть синтаксической структуры for проверяет выполнение условия, аналогично тому, как это делалось в конструкции if. В нашем случае условие ложно, если i меньше 10.
Третья часть синтаксиса for показывает, как должна изменяться переменная в каждом новом цикле. В данном случае i каждый раз увеличивается на 1. Результатом является следующий цикл: в начале переменная i имеет значение 0, в каждом цикле это значение увеличивается на 1, цикл заканчивается, когда переменная достигает значения 10. Таким образом, значение переменной i изменяется с 0 до 9 за 10 шагов. Если запустить вышеописанный код, вы увидите в окне Output цифры от 0 до 9.

Циклы for применяются для обработки данных внутри программ, а не для анимации объектов на экране. Пока выполняется цикл, на экране ничего не происходит. Короткие циклы, наподобие предыдущего примера, выполняются настолько быстро, что не оказывают никакого влияния на ролик. Однако длинные циклы могут задерживать воспроизведение ролика на несколько секунд и больше.
Приведем более распространенный пример цикла for. Предположим, у вас есть 10 клипов с именами "fox1" - "fox10", и вы хотите передвинуть их на 1 пиксель влево. Это действие будет выполнено при помощи следующего цикла:

```
for(i=0;i<10;i++)
_root["fox"+i]._x++;
}
```

Цикл меняет значение переменной i от 1 до 10. В нем применяется объект _root, с которым вы познакомились в уроке 7 "Возможность контроля клипов пользователем". Вы можете обратиться к любому клипу главной временной шкалы при помощи объекта _root [ ] с именем клипа внутри скобок. В данном случае код представляет собой строку, состоящую из слова "fox" и числа i.
Есть еще два способа создания циклов в ActionScript. Первый - это цикл while. В отличие от счетчика приращения (например, i), цикл while продолжает выполняться до тех пор, пока условие в начале выражения while верно. В цикле do, наоборот, условие находится в конце цикла. Разница в том, что если применяется вариант while, цикл может ни разу не выполниться, а в случае do он всегда выполняется один раз. Начинающие программисты на ActionScript нечасто используют подобные циклы, однако их необходимо знать.

Текст и строки

Использовать числа в ActionScript несложно. Вы можете прибавлять, вычитать и выполнять другие операции с числами. Со строками также можно выполнить множество операций. Операции со строками являются важным шагом в изучении программирования.
Одна из простейших операций над строками - их объединение. Мы рассмотрели ее в уроке 9 "Выполнение операций". В уроке 10 "Условные выражения" вы сравнивали строки для того, чтобы определить их тождественность и порядок следования.
Теперь давайте рассмотрим еще несколько строковых функций. Предположим, вы хотите узнать, включает ли одна строка другую. Для этого существует команда indexOf, определяющая положение первого символа второй строки внутри первой. Если вторая строка не найдена, результатом выполнения команды будет значение -1. Например, следующий код находит строку "World" в строке "Hello World" на шестой позиции, то есть там, где находится буква "W" (считая от 0). Если же вы зададите поиск "Earth", вы получите значение -1, так как этого слова в сочетании "Hello World" нет.

```
myString = "Hello World.";
myPosition - myString.indexOf("World");
```

При ссылке на положение символа ActionScript начинает отсчет с 0. Строка "Hello" будет находиться в положении 0 строки "Hello World".

При помощи команды length вы можете узнать количество символов в строке:

```
myString - "Hello World.";
myStringLength = myString.length;
```

Иногда необходимо получить определенную часть строки. Для этого существуют две команды. Команда substring задает часть строки от одного положения символа до другого. В данном примере результатом выполнения команды является текст "lo Wo":

```
myString = "Hello World,";
mySubstring = myString.substring(3,8);
```

Эту же задачу выполняет функция substr. В отличие от предыдущей команды она задает начальную позицию отрезка и количество содержащихся в нем символов. Следующий код задает строку, которая начинается с третьего символа и состоит из пяти символов:

```
myString = "Hello World.";
myPosition - myString.indexOf("World");
```

У нас есть уже две функции, выполняющие практически одну и ту же задачу. Почему бы не ввести и третью? Функция slice полностью аналогична substring.
Созданный текст вам нужно будет показать пользователю. Это очень легко. В главе 1 "Элементы Flash для создания игр" вы познакомились с динамическим текстовым полем. Когда вы преобразуете текст в динамический в панели Properties, вы можете в поле Var присвоить ему имя.

На этом рисунке изображена панель Properties, в которой динамическому текстовому полю присвоено имя mySubstring. Пока переменная и текстовое окно находятся на одном уровне Flash, панель будет отображать содержимое переменной mysubstring. Если же эти элементы находятся на разных уровнях, например один из них является частью клипа, они не будут связаны.

Панель Properties может использоваться для связи текстового окна с переменной ActionScript

Переменная может также быть связана с редактируемым текстом. В данном случае при изменении переменной соответственно изменяется содержимое текстового окна и наоборот.

В следующем уроке мы рассмотрим функции, а также пример динамического текстового окна.

Создание функций

Функция - это часть кода, которая может быть использована многократно. Вы можете передавать в нее несколько значений и получать из нее новые. Примером является функция, выдающая сумму двух чисел. Вот как она будет выглядеть в ActionScript:

```
function sum(a, b) {
c = a + b;
return c;
}
```

Обычно функции помешаются в сценарии кадра главной временной шкалы. Функция задается при помощи ключевого (зарезервированного) слова function. Затем указывается имя функции, за которым следуют круглые скобки и открытая фигурная скобка {. Круглые скобки могут быть пустыми или содержать список переменных, задаваемых при вызове функции. Эти переменные называются параметрами функции (или аргументами). Например, предыдущая функция может быть вызвана следующим образом:

```
trace (sum (7, 12));
```

Результатом обращения к функции будет число 19, помещаемое командой trace в окно Output. При обращении к функции sum значение 7 передается в функцию как переменная a, а значение 12 - как переменная b. После выполнения функции команда return посылает значение , с обратно к sum. Таким образом, для выполнения своей функции команда trace использует результат sum (7, 12) - число 19.
Функции используются по двум причинам. Первая - это возможность разделить программу на более мелкие части. Программу, состоящую из 30 строк и выполняющую три различные задачи, можно заменить на три функции, состоящие из 10 строк, каждая из которых выполняет свою задачу. Это значительно облегчает работу с кодом.
Другой причиной является возможность их повторного и неограниченного использования. Таким образом, если определенная часть кода используется несколько раз, вы можете поместить ее в функцию, которую затем будете постоянно вызывать, изменяя ее входные параметры.

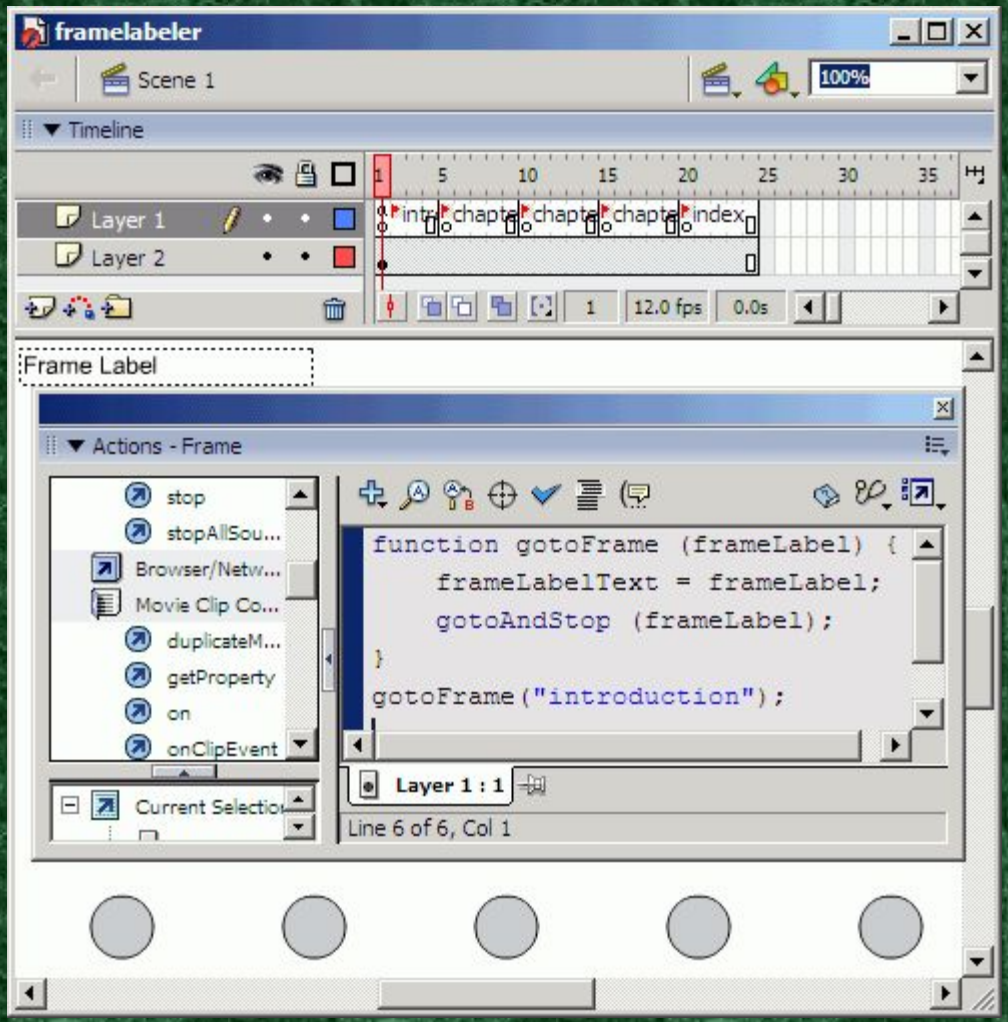
Приведем пример функции, использующей в качестве входного параметра имя метки кадра. Она выполняет две задачи. Во-первых, помещает имя в текстовую переменную textFrameLabel. Это текстовое окно видно пользователю при воспроизведении ролика, оно информирует пользователя о том, в каком кадре он находится в данный момент. Второй задачей функции является выполнение команды gotoAndstop - переход к определенному кадру и остановка воспроизведения.

```
function gotoFrame(frameLabel) {
franeLabelText - frameLabel;
gotoAndStop (frameLabel) ;
}
```

Функция помешается в первый кадр главной временной шкалы вместе с командой, задающей переход ролика к начальному кадру. Так как ролик уже находится в первом кадре, команда gotoAndstop просто останавливает ролик.

К каждому из пяти кадров пользователь может перейти при помощи кнопок, расположенных в нижней части экрана. Вместо команды gotoAndStop все кнопки обращаются к команде gotoFrame. На рис. 2.12 показан пример ролика. В верхнем левом углу рабочего поля находится область динамического текста. Временно он содержит слова "frame label". При первом обращении к функции gotoFrame они будут заменены на "introduction".

Ролик состоит из пяти кадров и пяти кнопок для перехода к каждому кадру. Текстовое окно отображает имя текущей метки кадра



Преимущество использования одной функции для контроля всей навигации заключается не только в меньшем количестве нажатий на клавиши при программировании. Предположим, что к моменту завершения программы вы 100 раз обратились к функции gotoFrame. Затем вы решили удалить окно textFrameLabel или немного изменить его. Если бы вы не использовали функцию, то вам пришлось бы удалять или изменять код в 100 местах. Но так как все ваши навигационные кнопки используют функцию gotoFrame, то для изменения их поведения достаточно изменить только ее.

Массивы

Никакое введение в язык программирования не было бы полным без рассмотрения массивов. Хотя множество простых операций может быть выполнено без применения массивов, человек, не умеющий работать с массивами, не может считаться настоящим программистом.
Массив является разновидностью переменной. Переменные, которые вы использовали до сих пор, могут содержать одно значение: число или строку. Массив может содержать ни одного, одно и больше значений. Вот как выглядит массив:

```
myArray - ["Apple", "Orange", "Peach", "Plum"];
```

Для обращения к элементу массива используется специальный синтаксис:

```
myItem. = myArray [1];
```

Переменная myItem будет иметь значение "Orange", так как отсчет положений в массивах, так же как и в строках, начинается с 0.
Несложно догадаться, что массивы применяются для хранения множества однотипных данных. Для выполнения операций над массивами предназначено несколько функций. Прежде всего, давайте рассмотрим, как создаются массивы. Это можно сделать несколькими способами. До сих пор мы рассматривали только способ, при котором все элементы массива задаются сразу.

Вы также можете создать пустой массив и добавлять в него элементы:

```
myArray = new Array();
myArray.push("Apple");
myArray.push{"Orange");
myArray.push("Peach");
myArray.push("Plum");
```

Первая строка данного кода создает пустой массив. Каждая из следующих четырех строк добавляет в массив один элемент при помоши команды push. По сравнению со способом "все сразу" данный способ кажется более сложным, однако он незаменим при построении массивов, все элементы которого не известны заранее. Например, вы можете позволить пользователю ввести данные, а затем добавить их в массив.
Длину массива можно узнать так же, как и длину строки:

```
myLength = myArray.length;
```

Вы уже знаете, как получить доступ к отдельному элементу массива при помоши квадратных скобок. Используя команду pop, вы сможете взять из массива значение последнего размешенного там элемента. При этом из массива он удаляется. В нашем примере мы передаем его в окно Output

Следующий сегмент кода добавляет в массив четыре элемента, а затем, используя цикл while, удаляет конечные элементы массива и перемешает их в окно Output:

```
myArray = new Array();
myArray.push("Apple");
myArray.push("Orange");
myArray.push("Peach");
myArray.push("Plum"); while (myArray.length>0) ( trace
(myArray.pop());
```

Первой строкой в окне Output является "Plum", последней- "Apple", так как команда pop берет элементы в порядке, обратном помещению элементов массива {стековая схема: "последний вошел, первый вышел" -First In Last Out).

Массивы можно упорядочить. В случае, если элементами массива являются числа, Flash упорядочит их в цифровой последовательности. Если это строки, они будут упорядочены по алфавиту:

```
myArray = new Array ();
myArray.push("Peach"};
myArray.push("Orange");
myArray.push{"Apple");
myArray.push("Plum"};
myArray.sort{);
trace(myArray.toString());
```

Обратите внимание на последнюю строку кода, которая содержит функцию tostring для приведения массива к виду, пригодному для отображения в окне Output, то есть преобразует все элементы массива в строку. При помоши данной функции вы можете убедиться в том, что массив соответствует вашим требованиям. Наконец, функция splice позволяет удалять один или несколько элементов из массива:

```
myArray = [ "Apple", "Orange", "Peach" "Plurr." ] ;
myArray.splice(2,1);
```

Команда splice может выполнять несколько функций. В данном примере она содержит два параметра. Первый задает позицию, с которой начинается удаление элементов (считая от 0!), второй - количество удаляемых элементов. В нашем случае будет удален элемент "Peach", находящийся в позиции 2. Изменив второй параметр на 2, вы удалите и " Peach" и "Plum". Убрав второй параметр, вы удалите все элементы массива.
Функция splice используется также для добавления элементов. Добавляемый элемент задается третьим параметром:

```
myArray = [ "Apple", "Orange", "Peach" "Plurr." ] ;
myArray.splice(2,1, "Pear");
```

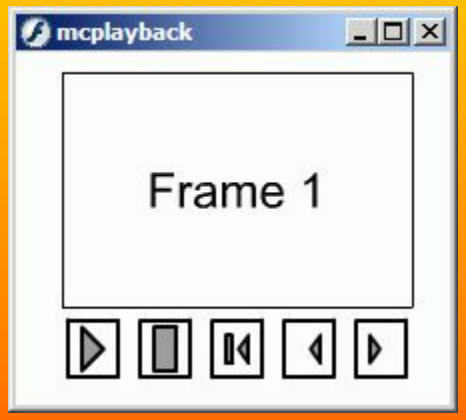
В данном примере удаляется элемент м Peach", а его место занимает элемент " Fear". Четвертый и последующие параметры используются для добавления второго и более элементов. В случае, если значение второго параметра окажется равно 0. никакие элементы удалены не будут. Теперь, когда у вас есть представление об основах ActionScript, самое время применить эти знания для создания вполне профессиональных фрагментов Flash-роликов.

Контроль воспроизведения ролика

Клипы представляют собой как бы небольшие ролики внутри основного Flash-ролика. Они включают полноценную временную шкалу со слоями, а также большинство элементов основного ролика. И основной Flash-ролик, и клипы можно контролировать при помощи команд play() и stop(). В библиотеке файла под названием Mcplayback.fla, находится клип "myMovieClip". Этот клип состоит из 10 кадров, промаркированных для того, чтобы вы знали, какой кадр клипа воспроизводится в данный момент.
Перетащив клип из библиотеки на рабочее поле, вы создадите его копию, которой можно и даже нужно присвоить имя. Для этого используйте панель Properties (Свойства)

Вы наверняка заметили, что копия клипа имеет такое же имя, как и клип в библиотеке. Обычно это делается при создании одной копии клипа. Имя копии легче запомнить, если оно совпадает с именем объекта библиотеки.

Библиотека ролика, рассматриваемого в примере, также содержит пять кнопок. Они помешены на рабочее поле под копией клипа. Присваивать имена копиям кнопок необязательно, впрочем, вы и не сможете этого сделать. В любом случае код не может ссылаться на кнопку, поэтому в имени нет необходимости.
После добавления клипа и пяти кнопок рабочее поле будет иметь при мерно такой вид, как показано на этом рисунке. Кнопки не только выглядят, но и используются как кнопки управления видеомагнитофоном.



Пять кнопок позволяют полностью контролировать клип. Кнопка Play (Воспроизведение) запускает воспроизведение ролика. При достижении конца клипа воспроизведение начинается снова с первого кадра. Кнопка Stop (Стоп) останавливает воспроизведение ролика в текущем кадре. Кнопка Rewind (Перемотать) возвращает клип к первому кадру. Кнопки Previous (Предыдущий) и Next (Следующий) продвигают клип на один шаг соответственно назад или вперед.
Прежде чем передать контроль над клипом пользователю, необходимо остановить клип. Обычно клипы запускаются сразу же после своего появления на рабочем поле. Чтобы предотвратить это, в первый кадр главной временной шкалы был помещен следующий сценарий, останавливающий анимацию клипа, а также сам ролик:

```
myMovieClip.stop();
stop () ;
```

Точка в первой строке сценария показывает, что функция stop () будет выполнена для копии клипа "myMovieClip". Вторая строка не содержит имени копии, поэтому команда выполняется в месте расположения сценария - в данном случае в главной временной шкале.
В сценариях для кнопок команды клипу посылаются также при помоши точки. Вот сценарий кнопки Play, задающий воспроизведение клипа:

```
on (press) {
myMovieClip.play() ;}
```

Кнопка Stop посылает клипу команду stop () :

```
on (press) {
myMovieClip.stop() ;}
```

Кнопка Rewind задает переход клипа к первому кадру и его остановку. Это действие выполняет команда gotoAndstop() с параметром 1.

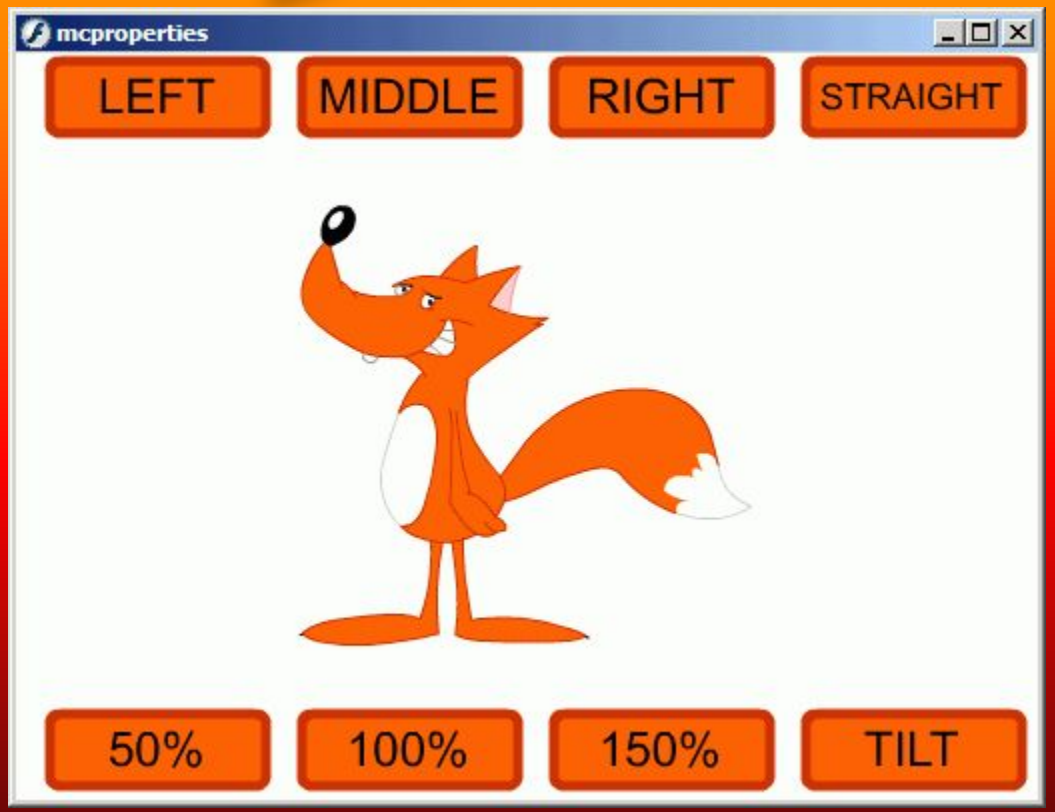
```
on (press) {
myMovieClip.gotoAndStop(); }
```

Кнопки Previous и Next перемешают клип на один кадр вперед или назад. Эти действия выполняются при помоши команд prevFrame () И nextFrame():

```
on (press) {
myMovieClip.prevFrame () ;}
on (press) {myMovieClip.nextFrame ();}
```

Управление свойствами клипа

Вы можете не только контролировать содержание клипа, но и задавать его внешние свойства. Например, управлять его положением, вращением и масштабом. Вокруг изображения лисы расположено восемь кнопок. Каждая из них задает свойство клипа. Внешний вид экрана показан на рисунке. Первые три кнопки имеют названия Left (Левая), Middle (Средняя) и Right(Правая).



При помоши кнопок, расположенных вокруг клипа "fox", можно изменить его положение, масштаб и повернуть его

В уроке 6 "Анимация при помоши ActionScript" мы узнали, что переменная _x используется для описания горизонтального положения клипа на рабочем поле. Эта переменная и будет использоваться кнопками Left, Middle и Right. Они могут использовать и переменную _y для описания вертикального положения клипа, однако в данном примере это не требуется. Приведем код кнопки Left. Он задает горизонтальное положение рисунка равным 200, то есть размещает его немного слева от центра при условии, что ширина ролика составляет 550 пикселов. Кнопки Middle и Right отличаются только значением переменной _x.

```
on (press) {
fox._x = 200;
} [code]
```

Для изменения масштаба клипа используются переменные _xscale и _yscale. В случае, если переменные имеют одинаковые значения, масштаб клипа будет изменяться пропорционально. Значение 100 соответствует масштабу 100%.

Значения переменных _xscale и _yscale можно задавать отдельно друг от друга. Они могут быть разными, при этом клип станет более тонким или более плоским.
Ниже приведен сценарий кнопки "50%". Две другие кнопки масштабирования используют аналогичный код, но с другими значениями.

```
[code]on (press) {
fox.__xscale = 50;
fox._yscale = 50;
}
```

Для вращения клипа используется переменная _rotation. Значение переменной задается в градусах, при этом 0 равен 360. Вы можете задать значение больше 360 или меньше 0, однако Flash преобразует их в показания между 0 и 360.

Далее приводится сценарий кнопки Tilt (Наклон). Она поворачивает клип на 30 градусов по часовой стрелке. Кнопка Straight (Прямо) возвращает клип в положение 0 градусов.

```
on (press) {
fox._rotation = 30;
}
```

Используя эти переменные в более сложных сценариях, рассматриваемых далее, вы сможете перемешать клип по рабочему полю, что необходимо для создания игр различных типов.

Применение метода “Перетащи и положи” к клипам

Важным свойством интерфейса, будь то игра или приложение, являете возможность перетаскивать элементы по экрану. Во Flash это можно сделать несколькими способами. Мы рассмотрим три из них. Команда startDrag задает автоматическое следование клипа за курсором. Пример Dragsimple.fla демонстрирует самый простой способ применения данной команды.

```
startDrag("circle", true);
stop () ;
```

В нашем примере команда startDrag использует два параметра. Первый - это имя перетаскиваемого клипа. Второй - ключевое слово true которое в данном случае задает привязку центра объекта к курсору. Ее второй параметр имеет значение false, за основу будет взято расстояние между курсором и центром клипа на момент выполнения командь startDrag. Ключевые слова true и false называются булевыми константами.

Они могут использоваться в случае, если атрибут либо действует, либо нет, а также отображать результат сравнения, например "a = = b.

В нашем примере клип "circle" следует за курсором. Однако чаше всего применяется другой способ перетаскивания: пользователь подводит курсор к объекту, нажимает кнопку мыши, перетаскивает объект в нужное место и отпускает кнопку.

Подобную манипуляцию можно осуществить с клипом, внутри которого содержится кнопка.

```
on (press) {
startDrag("",false) ;
on (release) {
stopDrag();
}
```

Пустые кавычки внутри команды startDrag сообщают Flash, что перетаскиваться будет текущий клип. Кнопка находится внутри клипа "circle", поэтому перемещаться будет именно он. false в качестве второго параметра означает, что клип не будет привязан к центру курсора, а сохранит первоначальное расстояние между курсором и центром клипа. При этом будет создаваться впечатление, что курсор схватил круг в некоторой точке и перетаскивает его за нее.

Команда stopDrag не требует никаких параметров. Одновременно может перетаскиваться только один клип, поэтому команде необходимо лишь остановить текущее действие перемещения, благодаря чему клип вернется в неподвижное состояние.

Одним из недостатков команды startDrag является то, что одновременно может быть перемешен только один клип. Кроме того, клип перемешается автоматически, поэтому за его движением трудно проследить. По этой причине вам не помешает знать еше один способ перетаскивания клипа.

Согласно данному сценарию, переменная drag получает значение true, когда пользователь нажимает на кнопку, и значение false, когда отпускает.

```
on (press) {
drag = true;
on (release) {
drag = false;
}
```

Переменная drag является глобальной переменной и совместно используется всем кодом клипа. В зависимости от значения drag сценарий определяет, должен ли клип следовать за курсором. Сценарий устанавливает соответствие между переменными _x, _y перетаскиваемого клипа и _xmouse, _ymouse ролика. Два последних параметра описывают положение курсора мыши относительно рабочего поля основного ролика.

```
onClipEvent (enterFrame) {
if (drag) {
this._x = _root._xmouse;
this._y = _root._ymouse;
}
}
```

Ключевое слово this обозначает ссылку на текущий объект. Сценарий назначен клипу "circle", поэтому this относится к нему. В следующем разделе мы рассмотрим другие способы обращения к клипам на различных уровнях.

Клипы и уровни

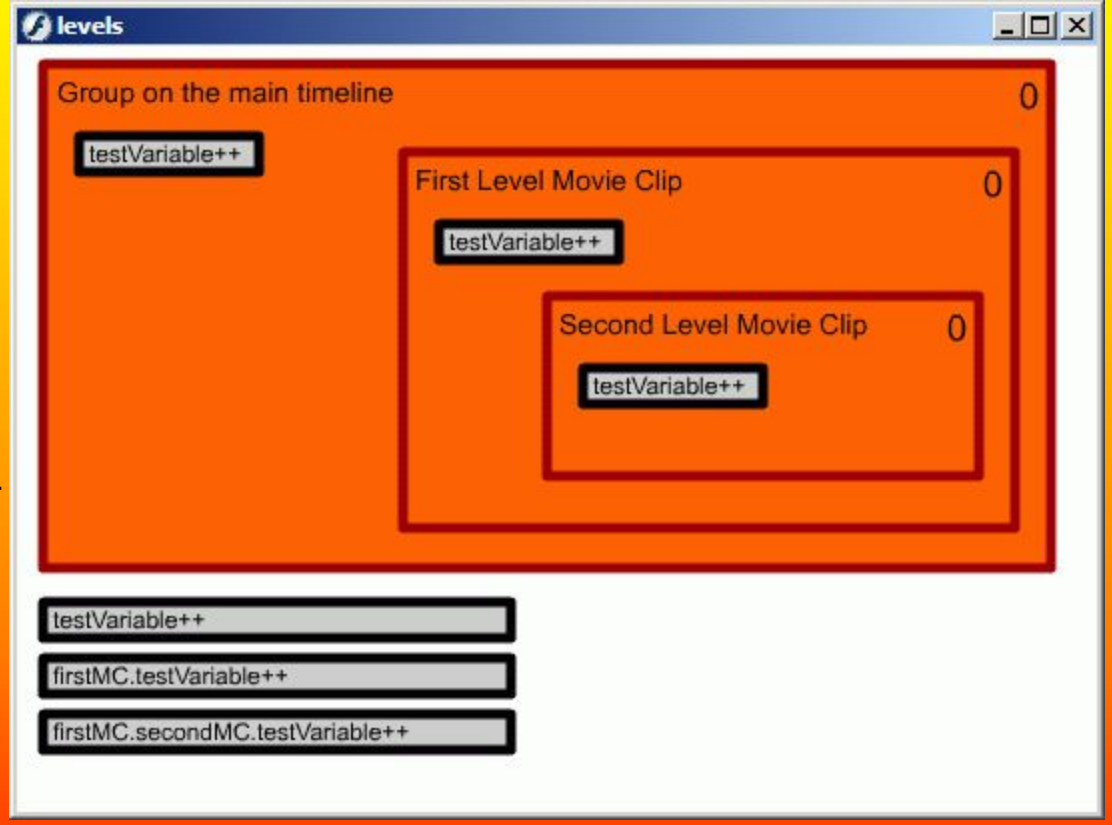
Начинающему программисту на ActionScript будет трудно понять, каким образом устроены клипы и уровни. Клип можно представить как ролик внутри основного Flash-ролика. Этот клип имеет свои переменные и атрибуты, отличные от параметров и атрибутов основного ролика.

Каждый раз, когда вы помешаете клип на рабочее поле, вы создаете новый объект. Основной Flash-ролик является объектом, а клип - объектом внутри этого объекта.

Flash-ролик можно сравнить с коробкой, полной игрушек. Если вы раскрасите коробку в синий цвет, игрушки не станут синими. Каждая из них сохранит свой первоначальный цвет. С другой стороны, если вы перенесете коробку на другое место, остальные игрушки последуют за ней, сохранив при этом свои свойства, такие как цвет и положение внутри коробки.

Предположим, коробка с игрушками закрыта, и вы просите кого-то в комнате достать машинку. Если человек не знает, что игрушка находится внутри коробки, ему будет нелегко это сделать. Недостаточно просто сказать: "Достань машинку". Необходимо сказать: "Достань машинку, которая находится в коробке". Подобным образом устроены и клипы. Если ваш клип расположен на главной временной шкале Flash-ролика, вы можете обратиться к нему по имени "toyTruck" ("игрушечный грузовик"). Однако, если ваш клип "toyTruck" расположен внутри другого клипа под названием "toyBox", к нему необходимо будет обратиться следующим образом: "игрушечный грузовик внутри коробки с игрушками", или "toyBox.toyTruck'.

В каждом из клипов и на главной временной шкале (можно сказать в клипе _root) мы определили свою переменную и каждой из них присвоили имя testvariable.
Рабочее поле, а также каждый клип содержат кнопку, увеличивающую значение testvariable. Каждой кнопке назначен следующий код:



Клип "secondMC" включен в клип "firstMC", находящийся на рабочем поле

```
on (press) { testVariable++; }
```

Кнопка изменяет значение переменной testvariable. на уровне того клипа, в котором находится данная кнопка. Кнопка, расположенная на рабочем поле, изменяет переменную testvariable на главном уровне Кнопка в клипе "firstMC" изменяет переменную testvariable в "firstMC", кнопка в "secondMC" - переменную testvariable в "secondMC" .
Следует отметить, что три переменные с именем testvariable. являются тремя разными переменными. Находясь на разных уровнях, они как не связаны друг с другом.
Нажатие одной из кнопок меньшего размера изменяет значение переменной testvariable только на том уровне, где она расположена.
Вы можете изменять значение переменных не только на уровне, где не ходится код, но и на других уровнях при помоши синтаксиса ActionScript. Примером могут служить три кнопки большего размера, расположенные в нижней части экран
Все три кнопки находятся на рабочем поле, а не внутри клипов. Первая кнопка изменяет значение testvariable, не указывая определенна клип. В результате изменяется переменная testvariable рабочего поля.

Вторая кнопка указывает переменную testvariable внутри клипа "firstMC". Код выглядит следующим образом:

```
on (press) {
firstMC.testVariable++ ;
}
```

В результате изменяется переменная в клипе "firstMC". Для изменения переменной внутри клипа "secondMC", необходимо учитывать, что "secondMC" находится внутри "firstMC" и программе нужно это указать.

```
on (press) {
firstMC.secondMC.testVariable++;
}
```

Во всех приводимых до сих пор примерах имена копий клипов указывались прямо в коде. Существует и другой способ обращения к клипам - свойство _root:

```
_root["firstMC"].testVariable++;
```

Это удобно, если имя копии клипа содержит пробел, в таком случае данный метод является единственно возможным. Свойство _root используется также при создании более сложного кода, где надо представить имя клипа как строку для неявного указания этого имени. Применение данного метода будет рассмотрено в следующем разделе.

Копирование клипов

Важным аспектом создания игры является умение манипулировать клипами. Но сначала нужно научиться создавать клипы. Хотя во Flash это сделать несложно, вы, возможно, захотите, чтобы ваш код создавал клипы во время воспроизведения ролика.

Представьте себе игру, в которой космические корабли врага атакуют игрока. Приближается один корабль, за ним следующий и т. д. Вместо того чтобы заранее создавать сотни клипов, ваш код может создавать их по мере необходимости.

Существует два способа создания клипов: копирование существующего клипа и создание клипа из эталона, находящегося в библиотеке, но не используемого изначально на рабочем поле.

В первом случае используется функция duplicateMovieClip, при помоши которой создается дубликат существующего клипа. Приведем пример:

```
firstclip.duplicateMovieClip("newclip",0);
```

Функция duplicateMovieClip запускается копируемым клипом. Поэтому при ее использовании упоминается имя этого клипа, в данном случае "firstclip". Функция также содержит два параметра: имя новой копии клипа и уровень нового клипа.

Это может немного вас запутать. Термин "уровень" здесь (применительно к команде duplicateMovieClip) означает порядок расположения клипов. В предыдущем же разделе данный термин использовался для описания включения одного клипа в другой.

В случае, если клип создан на уровне 0, как в предыдущем примере, он располагается под клипом уровня 1. Клип уровня 1 находится под клипом уровня 2 и т. д.

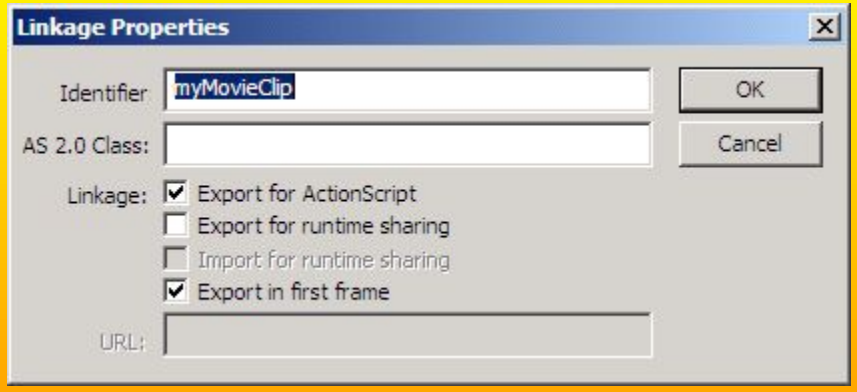
Не беспокойтесь, что два клипа окажутся на одном уровне, Flash не позволит это сделать. Достаточно в каждой команде duplicateMovieClip указывать другой номер уровня.

```
on (press) {
// Дублируем первый клип.
firstclip.duplicateMovieClip("newelip"+level,level);
// Помещаем его в случайную позицию.
_root["newclip"+level]._x = int(Math.random()*550);
_root["newclip"+level]._y = int(Math.random()*400);
// Увеличиваем счетчик.
level++;
}
```

Обратите внимание, что в коде ActionScript я в первый раз использовал комментарий. Строки комментария начинаются с двух косых черт. Содержание строки, следующей за косыми чертами, полностью игнорируется Flash и предназначено для того, чтобы вы понимали, какое действие выполняет код. Чем длиннее блоки вашего кода, тем больше они нуждаются во вспомогательных комментариях. Комментарии используются для последующего редактирования кода, а также для того, чтобы вашему коллеге было легче его понять.

В данной программе обработки переменная level используется по-разному. Во-первых, для присвоения имени новому клипу (сначала клипу присваивается имя newclip. Переменная также отображает уровень клипа В конце программы значение переменной level увеличивается на 1. Таким образом, следующий создаваемый клип будет называться level1 и располагаться на уровне 1. С помощью функции Math.random () мы задаем будущему клипу случайные координаты в пределах рабочего поля. Обратите внимание, ссылка на клип осуществляется при помощи синтаксической структуры _root []. Другой способ создания новых клипов - использование функции attachMovie. Для выполнения данной функции не требуется, чтобы клип находился на рабочем поле. Он должен просто быть в библиотеке. Однако, если клип находится в библиотеке и не используется на рабочем поле, Flash автоматически не включит его в конечный swf-файл. Для того чтобы клип был включен в конечный файл, его необходимо выбрать в библиотеке и настроить его параметры в диалоговом окне Symbol Linkage Properties, которое вызывается из меню Options

В окне следует установить флажок Export for ActionScript (Экспортировать для ActionScript). Затем вы должны придумать имя эталона, на которое будет ссылаться код. Я обычно использую имя эталона из библиотеки.



Диалоговое окно Symbol Linkage Properties позволяет включать клип в swf-файл, даже если он не используется на рабочем поле

Первым параметром функции attachMovie является имя из диалогового окна Symbol Linkage Properties. Второй параметр - это имя копии клипа на рабочем поле, третий - уровень клипа. Код данной кнопки, содержащийся в файле AttachMovie.fla, за исключением одной строки, полностью совпадает с кодом примера DuplicateMovieClip.fla.

```
on (press) {
// Дублируем первый клип.
attachMovie("myMovieClip","newclip"+level,level);
// Кладем в случайное место.
_root["newclip" + level]._x = int(Math.random!)*550);
_root["newclip"+level]._y = int(Math.random()*400);
// Увеличиваем счетчик, level++;
}
```

При помоши функции RemoveMovieClip вы можете удалять клипы с рабочего поля. Например, следующий код из файла RemoveMovieClip.fla перед созданием нового клипа удаляет предыдущий:

```
on (press) {
// Удаляем предыдущий клип.
_root["newclip"+(level-1)].removeMovieClip();
// Дублируем первый клип.
attachMovie("myMovieClip","newelip"+level,level);
// Кладем в случайное место.
_root["newelip"+level],_x=int(Math.random()*550);
_root["newclip"+level]._y=int(Math.randomf)*400);
// Увеличиваем счетчик.
level++;
}
```

Используя данные приемы, вы можете сделать так, чтобы ваши игры и приложения создавали свои собственные клипы, добавляя и удаляя их с рабочего поля по мере необходимости.

Управление несколькими клипами

Теперь, когда вы научились создавать клипы при помоши ActionScript, рассмотрим способы управления ими. Вы знаете, что можете управлять одним клипом с приписанным ему кодом, а если необходимо, управлять несколькими клипами? Что, если эти клипы ведут себя аналогичным образом?
Поместив клипы на рабочее поле, вы можете копировать код одного клипа и вставлять его в другие. Этот метод имеет несколько недостатков. Во-первых, необходимость копировать и вставлять. Затем, если вы захотите изменить код, вам придется сделать это во всех копиях клипа.

Создание клипа сценария
Единственным способом управлять несколькими клипами будет поместить код в точку, контролирующую их все. Например, если у вас есть 10 клипов, вы можете поместить код в первый клип, который будет контролировать все остальные. Почему бы вместо того, чтобы возлагать контроль на один из клипов, нам не создать клип, специально для этого предназначенный? Данный распространенный прием позволяет легко запомнить, куда вы поместили ваш код. Я назвал такой клип "actions movie clip" (клип сценария).

такой клип удобно называть контроллером.

Начните с создания при помоши инструмента Text небольшого текстового окна на рабочем поле и напишите в нем слово "actions" для того, чтобы его можно было легче идентифицировать.

Перед окончательной пубикацией ролика лучше удалить надпись "actions" из клипа сценария, так как итоговый swf-файл должен занимать как можно меньше места
Затем, выделив текстовое окно, выберите команду Insert -> Convert to Symbol для преобразования его в клип под названием "actions". Переместите его на серую область за пределами рабочего поля, так чтобы пользователь не видел слово « actions". Данный клип используется для назначения сценария, который будет управлять роликом. Предположим, вам необходим ролик, создающий 10 копий эталона из библиотеки и немного поворачивающий их с каждым новым циклом.

Прежде всего создайте эталон. В диалоговом окне Symbol Linkage Properties установите флажок Export This Symbol и укажите имя "sample" (см. файл actionsMCfla).
Код, написанный ниже, будет включен в клип "actions". Он будет находиться внутри программы обработки onClipEvent и состоять из двух частей. Первый обработчик будет отзываться на событие load. Событие load происходит при первом появлении клипа. Код выполняется один раз. В данном случае следует воспользоваться возможностью и создать 10 новых клипов:

```
OnClipEvent (load) {
// Создаем 10 клипов,
for(i=0;i<10;i++) {
_root.attachMovie("sample","sample"+i,i);
// Устанавливаем координаты
_root["sample"+i]._x = i*50 + 50;
_root["sample"+i]._y = 100; } }
```

Координаты клипов задаются одновременно с их созданием. Значение по вертикали равно 100, координата по горизонтали может иметь различные значения от 50 до 500. Результат показан на рисунке

Эти 10 клипов были созданы из эталона при помощи ActionScrip

Вторая часть кода находится в программе обработки onClipEvent (enterFrame). Код выполняется в каждом цикле клипа "actions". Если клип воспроизводится со скоростью 12 кадров в секунду, код будет выполняться также 12 раз в секунду.

```
onClipEvent (enterFrame) {
// Поворачиваем каждый клип в цикле,
for(i=0;i<10;i++) {
_root["sample"+i]._rotation += 5;
}
}
```

Данный код циклически выполняется во всех клипах и поворачивает каждый их них на 5 градусов. В результате на рабочем поле будет 10 вращающихся клипов.

Копирование клипов

В играх часто происходят столкновения между различными объектами, причем иногда с весьма разрушительными последствиями. Давайте научимся создавать код, позволяющий отслеживать такие события, как пересечение двух объектов или, например, прохождение курсора над определенным объектом.

Основной способ обнаружения ситуации, в которой два объекта пересекаются или объект закрывает определенную точку экрана, - использовать функцию hitTest. Аргументом функции hitTest могут быть координаты какой-либо точки или ссылка на обьект, например клип, кнопку или текстовое поле.

Давайте начнем с проверки пересечения клипа с точкой. Предположим, вы хотите определить, находится ли курсор над определенным клипом на рабочем поле. Клипу следует назначить следующий код:

```
onClipEvent (enterFrame) {
if (this.hitTest(_root._xmouse,_root._ymouse,true)) {
this._x = int(Math.random()*550);
this._y = int(Math.random()*400);
}
}
```

Строка this.hitTest () обозначает, что функция hitTest применяется к текущему клипу. Она включает три параметра: горизонтальное и вертикальное положение мыши, а также булевый параметр (значения true/false). Последний параметр определяет, использует ли Flash для контроля пересечения прямоугольную область, в которой содержится данный клип, или точную форму изображения в клипе. Во втором случае параметр должен быть равен true.

Для того чтобы определить, пересекаются ли два клипа, вы можете использовать один параметр - указатель на второй клип. В ролике Collision.fla на рабочем поле находятся два клипа. Клип большего размера называется "target", меньшего - "bullet". За пределами рабочего поля расположен клип "actions" со следующим кодом:

```
onClipEvent (enterFrame) {
// Выясняем, попала ли пуля в мишень.
if (_root["target"].hitTest(_root["bullet"])) {
// Попадание, увеличиваем цель.
_root["target"]._xscale += 5;
_root["target"]._yscale += 5;
// Убираем пулю.
_root["bullet"]._x = 350;
} else {
// Нет попадания, пуля летит дальше.
_root [ "bullet" ] ._x -= 5;
}
}
```

Код задает перемещение снаряда на 5 пикселов. Если сталкиваются два клипа, мишень немного увеличивается за счет увеличения масштаба на 5%.
Горизонтальная координата снаряда принимает первоначальное значение, и он может опять начинать свое перемещение.
Вы научились изменять масштаб клипа и его положение.