Санкт-Петербургский государственный институт
точной механики и оптики (технический университет)

Федерация Интернет-образования

Д.Г.Штенников, А.Л.Борисик, А.А.Зинчик

Создание образовательных ресурсов на основе использования технологий Macromedia Flash (Flash 4 для начинающих, Flash 5 для продвинутых)

Учебно-методическое пособие

Санкт-Петербург

2002

УДК 681.3

Штенников Д.Г., Борисик А.Л., Зинчик А.А. Создание образовательных ресурсов на основе использования технологий Macromedia Flash (Flash 4 для начинающих, Flash 5 для продвинутых). Учебно-методическое пособие. – СПб: СПбГИТМО(ТУ), 2002. – 100 с.

Рецензенты: Л.С. Лисицына, к.т.н., доцент, директор РЦ ФИО, директор ЦДО СПбГИТМО(ТУ)

Д.Д. Рубашкин, к.т.н., доцент, зам. Директора РЦ ФИО, доцент кафедры КТ СПбГИТМО(ТУ)

Учебно-методическое пособие предназначено для разработчиков образовательных internet-ресурсов. Вопросы, обсуждаемые в пособии , будут интересны и полезны при создании дистанционных курсов в системе ДО СПбГИТМО (ТУ), при подготовке выпускных квалификационных работ слушателей ФИО, слушателем курсов ДО для подготовки в рамках обучения по программам городского комитета по образованию, слушателем специализированного курса по Macromedia Flash и студентам.

ISBN 5-757-0091-2

ОГЛАВЛЕНИЕ

Оглавление....3
ВВЕДЕНИЕ....6
ИНСТРУМЕНТЫ РИСОВАНИЯ (drawing tools)....6
 Инструмент СТРЕЛКА....7
 Инструмент ЛАССО....8
 Инструмент ЛИНИЯ....8
 Инструмент ТЕКСТ....9
 ,  Инструменты КРУГ и КВАДРАТ....9
 ,  Инструменты КАРАНДАШ и КИСТОЧКА....9
 Инструмент ЧЕРНИЛЬНИЦА....10
 Инструмент ВЕДЕРКО ЗАЛИВКИ....10
 Инструмент ПИПЕТКА....10
 Инструмент ЛАСТИК....11
 Водопроводный кран....12
Изменение размеров и формы ластика....12
 Инструмент РУКА....12
 Инструмент ЛУПА....12
УРОВНИ....12
ВИДЫ АНИМАЦИИ....13
Shape....14
Motion....16
Motion guide....16
МЕНЮ....17
Создание меню с помощью переключений между ключевыми кадрами....17
Создание меню с помощью Tell Target....18
Создание меню с помощью Load Movie....18
Примечания и полезные советы:....19
Flash 5....20
Введение во Flash 5....20
Анимация во Flash....25
Кадры, слои, символы, временная шкала....26
Временная шкала....26
Слои....27
Кадры....27
Символы....28
Анимация....29
Покадровая анимация....29
Shape tweening....30
Motion Tweening....33
Направляющие слои....34

Цветовые эффекты ... 35
Основы ActionScript во Flash ... 36
Термины ... 36
Панель действий (Actions Panel) ... 37
Кнопки ... 38
Основные действия с Movie Clips ... 39
Имена ... 40
Пути ... 40
Отладка в ActionScript ... 42
Скорость исполнения фильмов ... 43
Размеры и "количество" анимации ... 43
Качество ... 43
Поточное воспроизведение и предварительная загрузка ... 44
Поточный звук ... 44
Размер .swf файлов ... 44
Повторное использование, символы ... 45
Графика, созданная вручную ... 45
Текст, шрифты ... 45
Звук ... 46
Изображения ... 46
Отчет ... 46
OOP Flash ... 47
Сразу опишем методы и работу с ними ... 52
childs() ... 52
allChilds() ... 52
pause(); ... 53
Описание механизма работы методов ... 54
Справка. "AS_3W_MC_Library.as" ... 58
Как пользоваться ... 58
Методы ... 59
movieTo(x,y); ... 59
movieBy(xOffset,yOffset); ... 60
size(W,H) ... 60
scale(persent); ... 60
scaleTo(xscale,yscale); ... 61
scaleBy(ratio); ... 61
rotateBy(angle); ... 61
hide(); ... 61
show(); ... 61
visInvert(); ... 62
flipV(); ... 62
flipH(); ... 62
playOffset(Offset) ... 62
"Редизайн" встроенных методов ... 62
Упражнение. SetTimeOut() ... 63
Упражнение. Тасование колоды во Flash5 ... 65

Примеры....65
Создание меню при помощи MC.play();....65
Создание меню через MC.nextFrame();....68
GetProperty....71
SetProperty....73
Путь к объектам....75
DuplicateMovieClip();....77
Пример № 1....77
Пример № 2....78
Пример № 3....79
Пример № 4....80
attachMovieClip....80
Создание своего курсора....83
Пример № 1....83
Пример № 2....83
Перетаскивание объектов (startDrag)....86
Получение координат мыши....86
Создание формы....87
Движение маски за мышью....88
Password....89

ВВЕДЕНИЕ

Итак, мы приступаем к изучению такого интересного и популярного пакета под названием Macromedia Flash на данный момент вышло уже шестое поколение этого программного продукта под названием Flash MX, но в целях упрощенного обучения я разбил свой курс на две части. Первая из них предназначена для людей, ранее не знакомых с этим пакетом, в ней разбираются интерфейс и базовые приемы работы с Flash 4, поскольку он является наиболее простым пакетом для освоения базовых принципов анимации. Часть, описывающая, работу с более поздней версией Flash предназначена для более опытных пользователей и содержит описание объектно-ориентированного программирования. Для начала я, как и обещал, рассмотрю базовые приемы работы с пакетом Flash 4

Программный пакет Macromedia Flash (в дальнейшем просто Flash) – мощный инструмент, облегчающий создание анимированных WEB-страничек. Модель графики в Flash представляет собой комбинацию растровой и векторной графики, соединяя в себе положительные стороны обоих графических представлений. Для начала давайте познакомимся с внешним видом главного окна и расположением панелей инструментов. Сразу под заголовком окна расположено меню. Ниже меню находится панель кнопок, которые дублируют наиболее часто используемые операции. Ниже панели кнопок находится окно управления слоями (Layers) и временная линейка (timeline). В центре расположено рабочее поле, то место, на котором будут создаваться рисунки и анимация. Слева находится панель инструментов рисования. Давайте рассмотрим эти инструменты подробнее.

ИНСТРУМЕНТЫ РИСОВАНИЯ (drawing tools)

Рисовать во Flash достаточно легко, если вы поймете основные принипы, такие, как использование инструментов, выделение, связывание и разбиение.

Выберите различные инструменты из панели инструментов рисования. В зависимости от выбранного инструмента внешний вид панели может измениться, могут появиться выпадающее меню в нижней части панели. Эти изменения показывают, что может делать выбранный инструмент.

 Инструмент СТРЕЛКА

Выделяет, меняет форму кривых, и передвигает элементы на рабочем поле. Кроме того, может использоваться для вырезания прямоугольных областей рисунка. Когда стрелка выбрана, вам доступны следующие опции.

 **Snap** Включает и выключает "намагничивание". Инструмент "намагничивание" поможет вам рисовать, автоматически выравнивая элементы относительно друг друга и разметки (grid) сцены, т.е. позволяет позиционировать объект по линиям сетки, точно соединять концы линий, позиционировать центр одного объекта с концом другого и т.д. *Он работает только со сгруппированными предметами или с символами*

 **Smooth** Сглаживает выделенные кривые

 **Straighten** Выпрямляет выделенную линию (отрезок линии) и уменьшает количество изгибов.

 **Rotate** Вращает и наклоняет выделенный объект в любом направлении в плоскости рисунка.

 **Scale** Увеличивает (уменьшает) выделенный объект

СТРЕЛКА позволяет изменять линию, отгибать края, изменять форму фигуры, перемещать объекты, выделять элементы несгруппированных объектов, разделять не сгруппированные объекты на составляющие элементы. Для перемещения элементов их надо выделить. Если вы попытаетесь передвинуть невыделенную линию, вы измените ее форму, но не сможете

передвинуть ее. Если выделено несколько предметов, а вам нужно передвинуть только один, снимите выделение и выделите только тот объект, который вам нужно передвинуть

Если вы передвигаете выделенную заливку поверх другой заливки и затем снимите выделение, то та часть рисунка, которая находится под верхним, вырежется. Это схоже с традиционными программами рисования.

 Инструмент ЛАССО

Инструмент ЛАССО выделяет произвольную часть рисунка. Поэтому при помощи ЛАССО можно выделять более точно, чем при помощи любого другого инструмента. Чтобы выделить область, ведите лассо вокруг этой области. Закончите круг приблизительно в том же месте где начали. Flash автоматически закроет петлю прямой.

Обычно текущее выделение снимается, как только вы делаете другое выделение, но если вы будете держать нажатой клавишу Shift когда используете ЛАССО, вы сможете добавлять другие области к уже выделенной. Используйте волшебную палочку для ЛАССО чтобы выделить какой либо цвет у растрового изображения, импортированного во флэш. (предварительно обязательно сделайте Modify →Break apart).

Два вышеперечисленных инструмента также часто используются для операций вырезания, копирования и вставки фрагментов изображений. Соответствующие команды Edit→Cut, Edit→Copy и Edit→Paste.

 Инструмент ЛИНИЯ

Позволяет рисовать прямые линии.

Дополнительные опции:

- Изменение цвета линии
- Изменение толщины линии
- Изменение стиля линии (одно из стандартных или Custom для создания собственного стиля)

 Инструмент ТЕКСТ

Позволяет вводить текст, аналогично любому текстовому редактору.

Дополнительные опции позволяют изменять шрифт и его размер, выделять жирным или курсивом, выравнивать по центру, справа, слева и по ширине страницы. Создает поле ввода, куда можно вводить текст.

 Инструменты КРУГ и КВАДРАТ

Позволяет рисовать овалы и прямоугольники. При нажатой клавише Shift соответственно круги и квадраты. Инструмент аналогичен инструменту ЛИНИЯ, в дополнительных опциях можно изменять цвет, толщину и стиль контура фигуры, а также цвет заливки.

 Инструменты КАРАНДАШ и КИСТОЧКА

Используйте КАРАНДАШ и КИСТОЧКУ, чтобы создать произвольную линию или штрих. Линия является одинарным элементом. Штрих кисточки - это заполненная область с контуром.

Когда вы рисуете линии, Flash может сглаживать кривые, превращать их в изломанные линии или оставить их в точности такими, как вы нарисовали. Вы выбираете эти опции, используя выпадающее меню. Помимо этого, вы можете указать толщину, стиль и цвет линии.

Straighten(выпрямление) распознает линии и кривые и "выпрямляет" их. Т.е., если вы выберете эту опцию и нарисуете волнистую линию, как только вы отпустите мышь, она превратится в зигзагообразную.

Smooth (сглаживание) не выпрямляет линии и не распознает кривые. Т.е., если вы выберете эту опцию и нарисуете зигзагообразную линию, она будет волнообразной (т.е. сглаженной).

Ink (чернила) оставляет линии такими, как вы их нарисовали, не сглаживая и не выпрямляя.

Oval рисует овалы или круги. Ведите мышью по диагонали из начальной точки к конечной. Если нажат "магнит", то, как только у вас получается круг, кружочек около курсора становится темнее и больше.

Rectangle рисует квадраты и прямоугольники. При нажатом "магните", как только у вас получается квадрат, кружок около курсора становится больше и темнее.

Line рисует прямую линию из начальной точки к конечной. При нажатом магните вы можете получить горизонтальную или вертикальную линии, если отпустите мышь, как только около курсора появится более темный и большой кружок.

 Инструмент ЧЕРНИЛЬНИЦА

ЧЕРНИЛЬНИЦА меняет цвет, толщину и стиль нарисованной линии. Нарисуйте линию. Щелкните на ЧЕРНИЛЬНИЦУ и выберите другой цвет, стиль и т.д. Щелкните на нарисованную линию. Она поменяет свои атрибуты. Если вы выделите несколько линий и щелкните чернильницей на них, то все выделенные линии поменяют атрибуты

Большинство опций ЧЕРНИЛЬНИЦЫ такие же, как и опции КАРАНДАША.

ЧЕРНИЛЬНИЦА не изменяет линий в сгруппированном символе.

 Инструмент ВЕДЕРКО ЗАЛИВКИ

Заливает контур выбранным цветом

 Инструмент ПИПЕТКА

ПИПЕТКА берет информацию о цвете и стиле готового изображения. Когда вы щелкаете на область инструментом ПИПЕТКА, она "копирует" цвет и стиль. Если вы щелкните на область заливки, около ПИПЕТКИ появляется значок КИСТОЧКИ, если щелкните на линию - появляется КАРАНДАШ. Инструмент ПИПЕТКА очень полезен для изменения атрибутов линии. Значения инструментов КАРАНДАШ и ВЕДЕРКО ЗАЛИВКИ при щелчке пипетки меняются на тот цвет и стиль, которые "скопировала" пипетка.

Если вы нажмете Shift и щелкните ПИПЕТКОЙ, цвет, который она "скопирует" одновременно появится и в инструменте КАРАНДАШ, и в инструменте " ВЕДЕРКО ЗАЛИВКИ, и в инструменте ТЕКСТ.

Примечание: ПИПЕТКА не может скопировать информацию из содержимого сгруппированных символов, пока вы не сделаете modify ->break apart.

Инструмент ЛАСТИК

Ластик стирает линии и заливку. Вы можете задать опции ластику так, что он будет стирать:

- только линии,
- только заливку,
- только выделенную заливку,
- только ту заливку, которую вы начали стирать.

Используйте опции ластику чтобы выбрать один из пяти размеров или изменить форму ластика.

Примечание: Ластик может стирать линии и заливки только в сцене, он не стирает группы объектов, символы или текст. Если вам нужно стереть часть группы выделите эту группу и выберите Edit > Edit Selected или Modify > Break Apart.

Опции в выпадающем меню определяют как ластик будет стирать. Опции очень похожи на те, которые вы видели в инструменте КИСТОЧКА.

Erase Fills стирает только заливку. Если вы попадаете на линии - они остаются.

Erase Lines стирает только линии. При попадании на заливку она остается нетронутой.

Erase Selected Fills стирает только выделенную заливку и не затрагивает линии вне зависимости, выделены они или нет.

Erase Inside стирает только ту область заливки, с которой вы начали стирать.(если вы начали стирать с пустого места, ластик не сотрет ничего) Линии не затрагиваются в этой опции.

 Водопроводный кран

Убивает отрезок линии или область заливки. Щелкните на ластик, а потом на кран.Если вы щелкните затем на линию или заливку, все отрезки линии и заливки будут стерты.

Изменение размеров и формы ластика

Изменяет размер и форму ластика. Вы можете выбрать любой из 5 размеров и форму круга или квадрата.

 Инструмент РУКА

Двигает все рабочее поле (заменяет полосы прокрутки)

 Инструмент ЛУПА

Изменение масштаба изображения.

Примечание: Увеличение и уменьшение не совпадает по величине (ошибка программы).

УРОВНИ

Уровни (Layers) используются для облегчения процесса редактирования. Фактически уровни представляют собой набор стекол, на которых нарисованы фрагменты картины. Наложенные друг на друга они образуют всю картину. Кроме того, разные типы анимации рекомендуется

располагать на разных уровнях. Желательно также использовать отдельный уровень для кнопок и команд.

На панели управления уровней доступны следующие установки режимов:

Глаз –Видно \ не видно изображение на этом уровне

Замок – Запрет \ разрешение редактирования на этом уровне

Контур – Представление объектов в виде контуров. Очень помогает при работе с невидимыми объектами (прозрачные объекты)

Режимы включаются \ выключаются нажатием левой кнопки мыши на соответствующую иконку.

ВИДЫ АНИМАЦИИ

Основной элемент, с помощью которого создается анимация – временная линейка (Time line). В отличие от мультипликации, при которой приходится рисовать все кадры, отражающие процесс изменения или движения, Flash позволяет рисовать только ключевые кадры (Keyframe), а изображение в кадрах, находящихся между ключевыми, выполняется программно, что, естественно, облегчает работу.

Анимация во Flash представлена двумя видами: анимация типа Shape и анимация типа Motion. Принципиальное отличие между этими типами анимации состоит в том, что первый тип работает с графикой как с набором графических примитивов (линий, дуг, окружностей, прямоугольников, заливок и т.д.), а второй тип работает только со сгруппированными объектами, «символами» (Symbols) в терминах Flash. Символы, как уже было сказано выше, представляют собой набор графических примитивов, объединенных в единый объект. В пакете Flash символы могут быть трех типов:

Graphics простое объединение примитивов.

Button объект, выполняющий функции кнопки, т.е. позволяющий выполнить определенные действия, при щелчке курсором мыши на нем.

Movie Clip анимационный ролик.

Символы могут быть помещены в библиотеку и затем использоваться многократно, по мере надобности.

Символ может быть создан двумя способами:

1) С помощью меню **Insert->New symbol…** создается новый (пустой) символ, в котором производится рисование. При этом в диалоговом окне надо выбрать тип создаваемого символа и его имя (имя по умолчанию **Symbol № …**). После окончания рисования символ переносится в локальную библиотеку символов.

2) Выделить «стрелкой» область рисунка из которого вы хотите символ создать. Выберите пункт меню **Insert->Convert to Symbol…** и нажмите кнопку OK (не забудьте указать тип символа). Созданный символ будет помещен в библиотеку, доступную с помощью команд **Window->Library,** и останется на рабочем поле.

Обратная операция, то есть разбивающая символ на составляющие его примитивы, выполняется выбором пункта меню **Modify->Break Apart**.

Shape

(преобразование формы, морфинг) – изменяет цвет и форму.

В самом общем виде алгоритм создания этой анимации может быть представлен следующим образом.

Шаг 1. Нарисуйте прямо в сцене, что вашей душе угодно, для морфинга сгодятся следующие приемы:

- Все, что вы нарисуете инструментами Flash.
- Импортированные bmp, gif или jpeg (если вы будете делать именно так, не забудьте сделать разбиение, выбрав из меню - **Modify ->Break Apart** или нажав ctrl +B) Импортированные символы Flash.
- Ну и конечно шрифт из Flash.

Шаг 2. Создайте конечный кадр (переведите курсор, к примеру, на 10(20, 30, 100): щелкните на кадре левой кнопкой мыши > выберите **Insert > Keyframe** (F6).

Шаг 3. Затем (курсор находится на конечном фрейме) создайте в нем любой другой рисунок или импортируйте любой другой gif, jpeg или bmp (в последнем случае не забудьте снова сделать (trace bitmap). Вполне можете комбинировать - в первом кадре, например, стрейсенная картинка, а в последнем текст. Главное помнить одно- картинка что в первом, что в последнем фрейме ни в коем случае не должна быть сгруппирована, а если это сгруппированный символ (например, из библиотеки), обязательно сделайте ему Break apart.

Шаг 4. Проверьте еще раз, чтобы ваш символ был разгруппирован. Проверить это можно следующим образом. Щелкните на ваш рисунок - если вокруг него появляется квадрат, а сам символ никак не выделяется - срочно жмите ctrl+b. Если же ваши рисунки и в первом, и в последнем кадре "затемняется", значит все нормально.

Шаг 5. Щелкните 2 раза мышью на 1 кадре (или правой кнопкой - properties).

Шаг 6. Выберите закладку **Tweening** из выпадающего меню выберите "**Shape tweening**".

Шаг 7. Опция Distributive лучше подходит для круглых рисунков. Angular - лучше применять для рисунков с множеством углов, изгибов и тд.

Выберите пункт меню **Control->Play**.

Motion

Этот вид анимации создается точно таким же образом, как и предыдущий. Единственное отличие заключается в том, что после создания рисунка, его необходимо преобразовать в объект (symbol в терминах Flash). Дальнейший процесс создания анимации ничем не отличается от создания Shape. На последнем шаге надо установить не **Shape Twinning**, а **Motion Tweening**. Существует более короткий способ создания этой анимации: надо щелкнуть правой кнопкой мыши между ключевыми кадрами, и выбрать пункт меню **Create Motion Tween**.

Motion guide

Это расширение анимации типа Motion. Позволяет задать движение вашего объекта по траектории любой формы.

Шаг 1. В этом виде анимации объект рисуется в одном слое, а траектория – в другом. (Guide: layer 2). Объект надо сделать графическим (выделить > F8 > graphic)

Шаг 2. В начальном кадре (начальные и конечные кадры в обоих слоях) объект «насаживается» на начало траектории, а в последнем – на конец траектории. Для упрощения соединения центра объекта и траектории рекомендуется включить режим «намагничивания» **Snap**.

Шаг 3. В layer 1 Properties > Motion Tweening. Если Вы хотите, чтобы объект просто двигался по траектории, то в окошке «orient to path direction» флажок нужно убрать, а если хотите, чтобы объект изменял свою ориентацию в пространстве в зависимости от направления траектории, то в окошке «orient to path direction» флажок нужно поставить.

МЕНЮ

Основной элемент меню – кнопка. Кнопка, как уже говорилось выше, является объектом (символом) и создается стандартным образом. Любой фрагмент рисунка может быть объявлен кнопкой. Кнопка имеет три состояния:

Up – исходное состояние

Over – курсор мышки находится над кнопкой

Down – нажатое состояние

и одно свойство

Hit – область нажатия, то есть та область сцены, нажатие в которой будет ассоциироваться с нажатием на Вашу кнопку.

По умолчанию кнопка имеет одинаковый вид для всех трех состояний и область нажатия, ограниченную размерами самой кнопки. Для редактирования состояний кнопки щелкните на ней правой кнопкой мышки и выберите один из режимов редактирования Edit in Place или Edit in new Window. В режиме редактирования нарисуйте каждое состояние. При необходимости создания анимационной кнопки, выделите ее, преобразуйте в символ типа Movie Clip и создайте анимацию, как было описано выше.

Создание меню с помощью переключений между ключевыми кадрами

Шаг 1. Нарисовать элемент > превратить его в кнопку (F8 > Button) или взять готовую кнопку библиотеки (Libraries > Buttons или Button –Advanced)

Шаг 2. Создайте конечный кадр (переведите курсор, к примеру, на 10(20, 30, 100): щелкнуть на кадре левой кнопкой мыши > выбрать Insert > Keyframe (F6).

Шаг 3. Затем (курсор находится на конечном фрейме) создайте в нем кнопки меню

Шаг 4. В первом кадре щелкните правой кнопкой мыши, выберите Properties > закладку Actions > Stop.

Шаг 5. Щелкните на начальной кнопке правой кнопкой мыши, выберите Properties > закладку Actions >Go To. В Frame Number укажите номер конечного кадра (или кадра с кнопками меню).

Шаг 6. То же самое можно сделать на любой из кнопок меню, чтобы вернуться к первой кнопке (только в Frame Number указать номер кадра, где находится начальная кнопка).

Создание меню с помощью Tell Target

Tell Target – очень мощный инструмент Flash, позволяет управлять ранее созданными или взятыми из библиотеки клипами.

Шаг 1. Создание клипа Insert > New Symbol > Movie Clip

Шаг 2. Создаем клип (рисунок выпадающих кнопок): первый кадр - один элемент > последний кадр – меню > между кадрами делаем анимацию Shape.

Шаг 3. В первом и последнем кадре щелкните правой кнопкой мыши, выберите Properties > закладку Actions > Stop.

На этом этапе у нас есть клип, помещенный в библиотеку.

Шаг 4. В Scene 1 из библиотеки Windows > Library извлечь Symbol 1 и присвоить ему имя: Modify > Instance >имя клипа

Шаг 5. Создать стартовую кнопку.

Шаг 6. На этой кнопке щелкните правой кнопкой мыши, выберите Properties > закладку Actions > Tell Target > имя клипа > «+» > play (On или Press или Release).

? Усложненный вариант:

Определяется переменная. Если она 0, то переход на метку 1 (Label 1)

Создание меню с помощью Load Movie

Использование операции Load Movie позволяет подгружать в текущую сцену Ваш ранее созданный ролик. Использование этой команды позволяет разбить Ваш файл на несколько составляющих и подгружать их по мере необходимости, что позволяет получить большой выигрыш в производительности при работе в условиях WWW. Те части ролика, которые

становятся не нужными после исполнения, могут быть выгружены командой Unload Movie.

Команда Load Movie использует 2 параметра. Первый: полный путь к файлу, с указанием префикса *http://*

Шаг 1. Создать клип и сохранить его в файле.

Шаг 2. Сделать из него выполняемый файл с расширением swf: file > publish

Шаг 3. В новом файле создаете кнопки пуска и остановки работы клипа или берете готовые из библиотеки.

Шаг 4. На кнопке пуска щелкаете правой кнопкой мыши > Properties > закладку Actions > Load Move > в окне URL полный путь и название файла с клипом.

Шаг 5. На кнопке остановки щелкаете правой кнопкой мыши > Properties > закладку Actions > Unload Movie.

Шаг 6. Сохранить файл и сделать из него выполняемый файл (swf): file > publish.

Примечания и полезные советы:

1. HTML и HTM расширения разные. При обращении из Flash к внешним файлам необходимо это учитывать.

2. Любую выполненную операцию можно отменить с помощью пункта меню **Edit->Undo**. Степень вложенности составляет несколько десятков операций.

3. Команды **View->Grid** и **View->Ruler** позволяют отобразить сетку на рабочем поле и линейку с делениями. Оба этих инструмента позволяют позиционировать элементы Вашего рисунка с точностью до одного пикселя.

4. «Горячие» клавиши для часто используемых команд:

Команда	«Горячие» клавиши:
Edit->Undo	Ctrl+Z

Insert->Keyframe	F6
Insert->Frame	F5
Insert->Blank Keyframe	F7
Insert->Delete Keyframe	Shift+F5
Insert->Delete Frame	Shift+F6
Insert->Convert to Symbol	F8
Insert->New Symbol	Ctrl+F8
Modify->Movie	Ctrl+M
Modify->Break Apart	Ctrl+B
Control->Play	Enter
Control->Test Movie	Ctrl+Enter
File->Preview in Browser	F12

Для начинающих все ☺. Если у Вас есть силы продолжать движемся дальше. И читаем курс для профессионалов.

Flash 5

Введение во Flash 5

Рассмотрим некоторые преимущества и недостатки популярной версии Flash 5 из которых вытекают наиболее логичные применения этого пакета. Как вы помните отличительной особенностью Flash является его возможность создавать нестандартные интерфейсы. «Нестандартные» интерфейсы имеют ряд отличий от «стандартных» (под «стандартными» подразумеваются привычные интерфейсы HTML):

1. Специальные управляющие объекты (кнопки, панели, блоки). Для примера, рулетка в Microsoft Word – нестандартный объект. Данный объект можно реализовать в HTML, но только в виде картинки, но невозможно повторить интерактивную функциональность объекта.
2. Независимое размещение объектов, другими словами, не размещение объектов относительно друг друга, а расположение по координатам и уровням. В DHTML такая возможность существует, но в DHTML надежно реализовать можно только совсем простые вещи.
3. Прозрачное взаимодействие с любым объектом. Т.е. все объекты равны. Не складывается ситуация, когда часть принадлежит системе, часть вашему коду, и т.д., и при этом набор обрабатываемых событий один для всех.

В результате подобной «нестандартности» появляется полная свобода в создании интерактивного интерфейса, более удобного, более наглядного, более функционального. Это реально повышает уровень предоставляемого сервиса. А значит, достигается «customer satisfaction» (удовлетворение

посетителя), и, в конечном итоге, система становится более конкурентоспособной.

И именно Flash 5 дает возможность делать такие интерфейсы принципиально проще, чем любой другой инструмент, который можно всерьез рассматривать как сколько-нибудь значимую рыночную технологию.

Разработчики, попробовавшие программировать в среде Flash 5, подтвердят мои слова: Flash - уже не просто технология для создания анимационных роликов. Другими словами, Flash стал применим для создания интерактивных приложений.

Резонно задаться вопросом: а зачем это надо? Есть ли смысл использовать Flash 5 там, где он никогда не использовался? Ведь «несть числа» всевозможным языкам программирования, описывающим клиентскую часть. А Flash, к тому же, - один из самых медленных.

Здесь важно понять, что существует два принципиальных условия применения Flash:

1. Надо аккуратно выбирать область применения Flash за пределами анимации.
2. Этим инструментом надо уметь грамотно пользоваться.

С первым условием достаточно просто: Flash нужен там, где нестандартный интерфейс дает много новых возможностей, где нужна интерактивность, где не подходит «спартанская» внешность. При совпадении всех этих требований имеет смысл задуматься об использовании Flash 5 в качестве инструмента для построения системы.

Чем определяется «грамотность» применения Flash 5? Необходимо определиться, каковы преимущества использования именно Flash в конкретном проекте, и с какими «подводными камнями» придется столкнуться.

Основные плюсы программирования в среде Flash 5 - в процессе разработки:

1. Почти каждая аккуратно запрограммированная функция сразу очевидно полезна во многих местах.
2. Возможно построение универсального сервера.
3. Легко переносится часть логики с серверной на клиентскую часть.
4. Свобода в верстке и в наборе управляющих элементов (кнопок, меню, списков, таблиц).

Есть, однако, и недостатки - что-то работает не лучшим образом, а потому, если в системе важны определенные компоненты, Flash использовать пока нецелесообразно. Собственно, список тех компонентов, которые на данный момент «не дружат» с Flash:

1. Сложные математические операции на клиентской части.
2. Работа с очень сложными структурами данных на клиентской части.
3. Мелкие тексты, написанные по-русски, из-за проблемы с кодировкой.
4. Сайты со сверхсложной бизнес-логикой, требующие мгновенной загрузки.

5. Механизмы, требующие работы с файловой системой клиента или его устройствами (например, web-камерой или микрофоном).

Вот, пожалуй, и все трудности.

1. Отладка Performance and Memory-Use на XML, Math/Data Functions.
2. Окончательное разделение программирования и дизайна.

Как вы думаете, возможно ли уместить страничку, содержащую приличное количество анимации, звука и удивительных способов интерактивности в файл порядка 100kb? Сделать так, чтобы эта страница работала одинаково как в Netscape Navigator (NN), так и в Internet Explorer (IE)? Компания Macromedia решила большинство проблем совместимости и производительности, выпустив Flash, который к сегодняшнему дню весьма эволюционировал и является полноценной частью инструментов / техник web-дизайна.

Коротко и ясно о том, что это такое. Существуют plug-ins (программы-надстройки), которые встраиваются в браузер (web browser), и служат для просмотра Flash страниц. Называются они Flash Player. Причем в последних версиях IE и NN эти программы уже встроены (если нет, то их можно бесплатно скачать с сайта Macromedia). И существует программа Flash, с помощью которой эти страницы создаются.

В пользу Flash приведу его основные достоинства и статистку Macromedia.

- **Маленький размер получающихся файлов и, соответственно, более быстрая загрузка из сети.** Flash использует векторный формат изображений и сжимает растровые и звуковые файлы, (которые также могут использоваться в страницах Flash), что очень положительно влияет на уменьшение размера страницы и время ее скачивания.
- **Устранение проблем совместимости между браузерами.** В отличие от HTML, Flash одинаково работает как в IE, так и в NN. Имеется даже специальный вариант программы-проигрывателя для браузеров, поддерживающих Java (Flash Java Player).
- **Мощный событийно-управляемый язык.** В Macromedia Flash используется специальный язык, при помощи которого можно создавать "интеллект" для своей страницы. Причем если в Flash 4 это был, скорее, некий скрипт (script), имеющий всего несколько основных функций, то в Flash 5 (несмотря на название "ActionScript") - это почти полноценный язык программирования, с поддержкой условий, циклов, массивов, функций и классов, которые можно наследовать.
- **Красота.** Да, да, именно! Flash имеет автоматическую поддержку anti-aliasing (антиалайсинг) - сглаживание контуров с помощью смешения соседних цветов). В результате даже простая линия или кружочек, нарисованные во Flash, выглядят приятно для глаз. Что уж говорить о рисунках, нарисованных профессионалами.
- **Удобство.** Создавать страницы во Flash под силу даже ребенку, и, признаюсь, это весьма приятное занятие. А если обладать элементарными

навыками дизайна и рисования, открывается весь простор для Вашей фантазии, предоставляемый Flash.

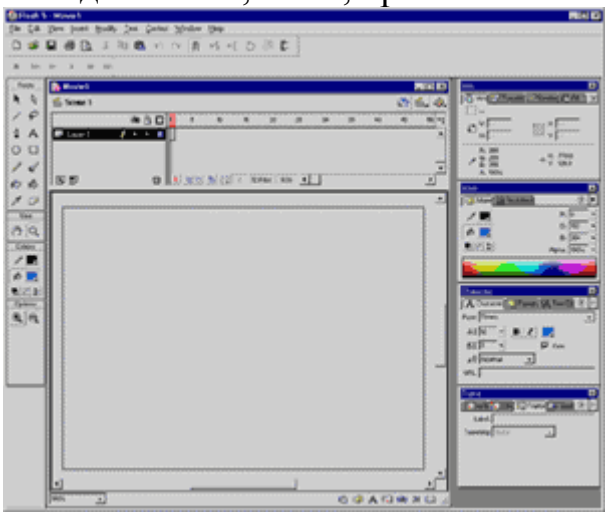
- **Распространенность.** Flash потихоньку становится стандартом де-факто (см. статистику ниже). В случаях, где необходима широкая интерактивность, графика, звук, и маленький размер, Flash незаменим.

Статистика такова:
На сегодняшний день Flash Player используют 222 миллиона человек, и каждый день его скачивает еще 1.4 миллиона. По данным Macromedia это позволяет 90% пользователей Сети просматривать страницы с Flash содержимым.

Plug-ins распространяются бесплатно, в то время как за программу создания Flash файлов приходится платить. Последняя, 5-я версия продукта стоит $399. Пользователям старых версий это удовольствие достанется за $149.

Итак, приступим.

Будем считать, что у вас уже есть Flash 5. Программа работает под Windows 95/98/NT/2000. Установить ее очень легко - с этим справится любой, кто установил хотя бы парочку программ в Windows. После установки можете смело заходить в нее, и вот, приблизительно, то, что вы увидите:



Интерфейс Flash 5 очень похож на интерфейс программ Adobe. Он весьма удобен и легок. После некоторого времени работы с программой понимаешь, как хорошо все продуманно и сколько труда разработчики Macromedia вложили в свое детище.

Слева находятся панели инструментов. С помощью них можно выбирать инструменты, а также управлять рабочей областью, модифицировать объекты и выбирать простые цвета. Справа находятся диалоги настройки инструментов, цвета, текста, свойства кадров и объектов. Посередине - рабочая область, где мы будем творить, а над ней шкала времени (Timeline).

Во Flash очень интересно рисовать. Этот векторный редактор не похож ни на один из существующих. К сожалению, подробное описание инструментов рисования выходит за рамки данной статьи. Впоследствии, мы можем рассмотреть эти инструменты подробнее. Самый хороший способ чему-то научиться - практика при некотором количестве теории. В качестве

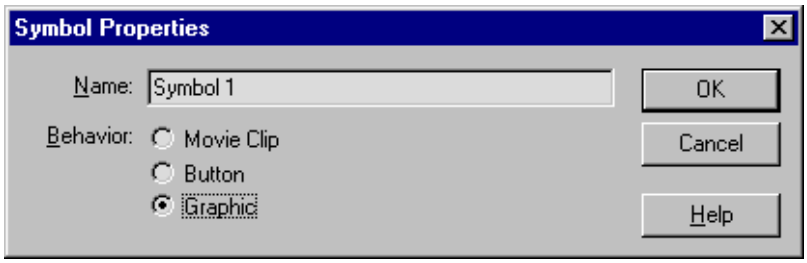
практики просто поэкспериментируйте с различными инструментами, а в качестве теории можете прочитать про это в книгах или в Сети.

Странички (файлы, анимации - как хотите) Flash принято называть фильмами (Movie), хотя мне больше нравится слово "мультик". Наверное, это оттого, что во Flash имеется шкала времени и, хоть вы и обладаете безграничными возможностями по использованию этой шкалы, ваше творение все равно будет упорядоченно выполнятся во времени. На самом деле, вы можете останавливать, вновь запускать мультик, прыгать с одного кадра на другой, загружать другие мультики и много чего еще.

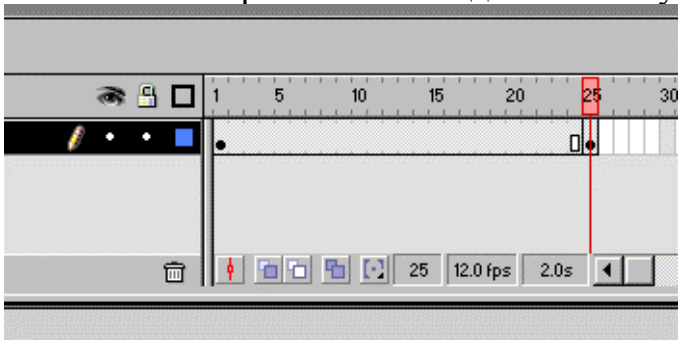
Процесс создания состоит в следующем. Вы создаете так называемый "авторский файл", который имеет расширение .fla, а затем он транслируется в результирующий .swf файл, который уже может быть просмотрен в браузере, обрамлен с помощью HTML, и т.д. Кстати, Flash может записать ваше творение в отдельный исполняемый .exe файл, сгенерировать java-код. И даже сохранить в виде статического GIF изображения.

Давайте попробуем нарисовать первый простой мультик. Нам понадобятся инструменты - "овал"  и "выделение"  . Выполните следующую последовательность действий:

1. Выберите инструмент "овал" и нарисуйте с помощью него овал или круг в левой части сцены.
2. Далее выберите инструмент "выделение" и выделите весь овал вместе с кромкой. Для этого либо охватите вашу фигуру прямоугольной рамкой, держа кнопку мыши нажатой, либо два раза быстро щелкните на ней. У вас выделится весь овал.
3. Теперь войдите в меню Insert и выберите Convert to symbol (или нажмите F8). В появившемся диалоге выберите селектор Graphic и нажмите OK:



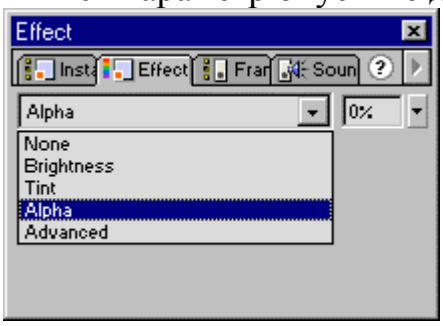
4. Теперь выберите на шкале времени вверху 25-й кадр (просто щелкните по нему мышкой), и выберите меню Insert -> Keyframe (или нажмите F6). Этим вы создадите так называемый "ключевой кадр" на 25-м кадре вашей шкалы времени. У вас должно получиться нечто похожее:



5. Выделите кружок (теперь вокруг него возникнет голубая рамка) и переместите его в правую часть рабочей области.

6. Вернитесь на шкале времени в 1-й кадр. (В доказательство Вы должны увидеть свою фигуру опять в левой части экрана). Из меню Insert (или из контекстного меню при нажатии правой клавиши мыши на первом кадре) выберите Create motion tween.
Поздравляю! Вы только что сделали очень простой, но уже мультик, во Flash. Выберите Control > Play (или просто нажмите Enter, или Ctrl+Enter), чтобы просмотреть результат.

7. Давайте добавим кое-что в наш мультик. Выберите опять 25-й кадр и выделите вашу фигуру. Войдите в диалог Effect. Если сложно отыскать нужную закладку в диалогах справа, выберите Window > Panels > Effect. В выпадающем списке в этом диалоге выберите Alpha (прозрачность), а появившийся параметр опустите до 0%.



Попробуйте еще раз проиграть ваш фильм. Во Flash можно всего лишь задавать "ключевые точки" анимации, программа сама будет просчитывать промежуточные кадры. Хотя вполне возможна покадровая анимация.

8. Нужно оттранслировать файл в .swf и сгенерировать HTML файл, который загружал бы мультик в браузер. Это можно сделать, выбрав File > Publish (или нажав Shift-F12).

Теперь можно открыть созданный вами HTML файл в браузере. Это можно сделать даже из Flash, выбрав File > Publish Preview > HTML (или нажав F12). При этом Flash воспользуется браузером, установленным по умолчанию.

Вот, вкратце, цикл создания простой сценки во Flash.

Анимация во Flash

Итак, анимация. В нашем случае это слово означает последовательность сменяющихся изображений (кадров), в результате чего возникает иллюзия движения. В Macromedia Flash существует два принципиально разных способа анимировать что-либо. Первый - прорисовывать каждый кадр самому, используя Flash только в качестве средства, позволяющего быстро пролистывать ваши изображения, и второй - заставить Flash автоматически просчитывать промежуточные кадры.

Для людей незнакомых с базовыми приемами компьютерной анимации (или просто - чтобы было понятнее), поясню на примере. Скажем, у вас есть зеленый квадрат, который нужно переместить из левой части экрана в правую. И сделать это надо в течение 25 кадров. В случае первого "покадрового" способа анимации вам придется нарисовать все двадцать пять кадров, и в

каждом последующем кадре чуть-чуть сдвигать ваш квадрат, чтобы он оказался справа на 25-м кадре. А если вам нужно будет сделать так, чтобы квадрат потихоньку исчезал, двигаясь вправо? А если вдруг понадобится, чтобы он исчезал (увеличивался атрибут прозрачности) экспоненциально? Что, придется все это считать вручную и присваивать нужное значение прозрачности на каждом кадре?

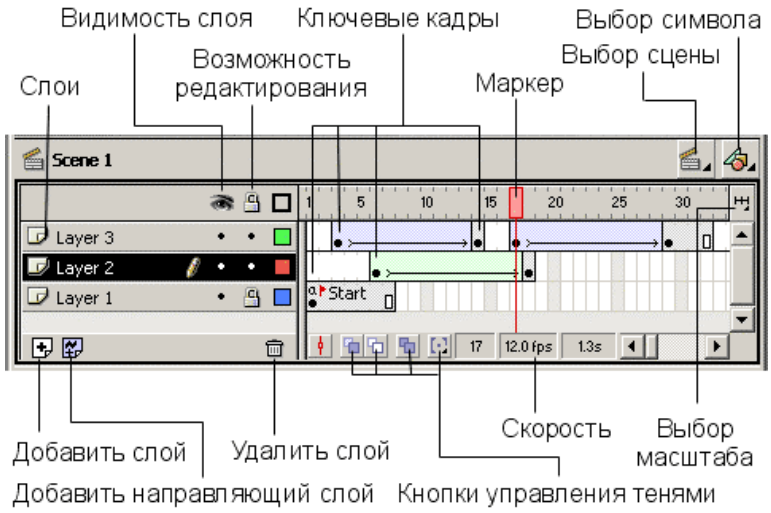
Вовсе нет. Для этого существует второй способ анимации - с помощью промежуточных отображений (tweening animation). В этом случае вы только задаете ключевые кадры (keyframes), а промежуточные Flash просчитывает автоматически. Вам понадобится задать только 2 кадра: начальный и конечный. По умолчанию Flash рассчитает промежуточные кадры по линейному закону, но можно задать возрастающую или затухающую экспоненту. Это нужно, чтобы отразить какие-нибудь процессы, происходящие в реальном мире. Например, движения мяча. Мы еще вернемся к этой теме.

Кадры, слои, символы, временная шкала

Мы досконально разберем все способы создания анимации, но сначала определимся с некоторыми базовыми понятиями. Этими понятиями являются **кадры** (frames), **символы** (symbols), **слои** (layers) и **временная шкала** (timeline).

Временная шкала

Временная шкала - основной инструмент при работе с анимацией во Flash. На ней отображается информация о слоях, о том какие кадры являются ключевыми, а какие генерирует Flash. С помощью временной шкалы можно понять, какие кадры содержат действия или метки. Она позволяет перемещать ключевые кадры и целые куски анимации. Вы очень быстро освоитесь с этим инструментом, благодаря хорошо продуманному и удобному интерфейсу. Временную шкалу очень легко найти, даже если вы впервые работаете во Flash:



Временная линия

Подробное рассмотрение всех элементов шкалы займет очень много времени и места, поэтому я только перечислю ее основные возможности:

- **Маркер** - указывает на текущий кадр, отображаемый в окне. При клике на какой-либо кадр, маркер автоматически перемещается на него.

- **Слои** - слева находится перечень слоев. Под ним существуют кнопки, позволяющие добавлять и удалять слои. Каждый слой можно сделать невидимым и запретить его для редактирования.

- **Шкала кадров** - поле, где вы можете добавлять и удалять простые и ключевые кадры. Если вызвать контекстное меню (нажать на правую клавишу мыши) на каком-либо кадре, вы увидите перечень действий, которые можно совершить. На шкале отображается информация о кадрах, которые являются ключевыми (такие кадры помечаются черными кружочками), содержат действия (буковка "a" над кружочком) или метку (красный флажок, после которого идет название метки). Цвет тоже говорит о типе кадров. Серый цвет - это кадры, которые в точности повторяют ключевой кадр (keyframe). Синеватая или зеленоватая подсветка говорит о том, что кадры сгенерированы Flash (о различиях будет рассказанно ниже). И, наконец, белое или "пустое" полосатое пространство говорят о том, что на этих кадрах ничего нет.

- **Кнопки управления тенями** - это кнопки, позволяющие отображать соседние кадры как бы через кальку, чтобы видеть разницу между предыдущими и последующими кадрами. Можно задавать глубину такого отображения по обе стороны от маркера.

Я думаю, вы очень быстро разберетесь со всеми функциями шкалы времени, если немножко поэкспериментируете.

Слои

В компьютерной графике этот инструмент используется очень часто. Представьте, что вы рисуете на прозрачных листах, а потом накладываете их друг на друга. То, что находится на верхних слоях, закрывает содержимое нижних слоев. Слои можно делать невидимыми и/или недоступными, чтобы облегчить редактирование сцены в целом.

Во Flash есть пара особенных типов слоев: слои, содержащие траектории движения и слои - маски. Про первые мы поговорим ниже, обсуждение же вторых выходит за рамки данной статьи.

Есть достаточно большое количество приемов, в которых используются слои, но во Flash без них просто нельзя обойтись по одной важной причине: **в один момент времени для каждого объекта анимации нужен отдельный слой**. Объектом анимации считается фигура (shape) или символ (symbol).

Кадры

Наша анимация состоит из последовательности кадров. Кадр может быть как составленным вручную, так и сгенерированным Flash. Это относится к кадрам одного слоя. Так как сцены Flash состоят обычно из нескольких слоев, то итоговые "многослойные" кадры, могут содержать, как сгенерированные, так и "самодельные" слои.

В компьютерной анимации существует понятие - **ключевые кадры** (keyframes). Их название говорит само за себя. Это кадры, которые Flash не вправе изменять в процессе создания анимации. Вы задаете эти ключевые кадры, а промежуточные кадры между ними выстраивает Flash. Существует два типа промежуточных кадров - кадры, построенные на основе изменения

геометрии (shape tweening) или кадры, построенные на изменении символов (motion tweening). И, конечно же, кадры могут быть пустыми, т.е. ничего не содержать.

Элементарные операции с кадрами:

Вставить пустой ключевой кадр - Insert->Blank keyframe, F7

Ключевой кадр, повторяющий содержание предыдущего - Insert->Keyframe, F6

Очистить ключевой кадр - Insert->Clear keyframe, Shift-F6

Вставить обычный кадр - Insert->Frame, F5

Удалить кадр - Insert->Remove Frames, Shift-F5

Символы

Символы - одно из ключевых понятий во Flash. Символом может быть, как простейшая геометрическая фигура или объединение этих фигур, так и целая анимация (movie). Это позволяет использовать символы как мощный механизм создания абстракций во Flash.

Например, можно сделать символы - колесо, корпус, стекла, антенны. Потом все это объединить в символ - автомобиль. А затем создать сцену, на которой этот автомобиль будет ехать. Другой пример. Допустим, вам нужно нарисовать падающий снег. Вы создаете символ снежинки, создаете символ, содержащий несколько анимирующихся снежинок, далее создаете символ в виде столбика падающих снежинок, затем размножаете этот столбик - и получаете падающий анимирующийся снег на всю сцену.

Я думаю, вы уловили смысл символов. Символы добавляют гибкости вашей сцене. В случае с автомобилем вы можете сделать колесо анимированным символом, так, чтобы ощущалось вращение. Можете сделать дверь кнопкой, чтобы при клике мышкой она открывалась. Смысл в том, что в любой момент вы можете изменить содержание и вид символа, что существенно сокращает затраты на модификацию Flash сцен.

Существует три вида символов: **анимация** (movie clip), **кнопка** (button) и **изображение** (graphic):

- **Изображение** (graphic), представляет собой символ, состоящий из единственного кадра. Отсюда следует его статичное название. Если символ действительно представляет собой статичный (не анимирующийся) объект, лучше сделать его изображением (graphic).

- **Кнопка** (button). Во Flash есть специально приспособленный под функции кнопки вид символа. В нем имеется 4 кадра: Up, Over, Down, Hit, которые содержат следующие состояния кнопок:

- Up - обычное состояние кнопки.
- Over - когда курсор мышки находится над кнопкой.
- Down - когда курсор находится над кнопкой и нажата клавиша мыши.
- Hit - обычное состояние, для кнопки, содержащей ссылку, которую пользователь уже посещал.

- **Анимация** (movie clip). Это самый "полноценный" тип символа. В нем может быть любое количество кадров. Символ этого типа может

восприниматься как объект типа Movie в ActionScript (это встроенный язык Flash).

Символы могут быть вложенными вне зависимости от типа. Это является самым главным их достоинством. Например, можно сделать кнопку, которая начнет двигаться, когда над ней будет "пролетать" курсор мыши, просто поместив в кадр Over символ - анимацию. Или (парадокс!) на изображение поместить бегущую кошку. Все остальное - дело вашей фантазии.

Символы можно создавать как "с нуля" (Insert->New Symbol, Ctrl+F8), так и используя текущее выделение, поместив его сразу в символ (Insert->Convert to Symbol, F8). Второй прием используется гораздо чаще, чем первый, т.к. в этом случае отпадает надобность его позиционировать и изменять под нужный размер.

Для управления символами используется так называемая **библиотека** (Library), описание которой, к сожалению, не входит в рамки данной статьи. Окно библиотеки находится по адресу Window->Library (или Ctrl-L).

Анимация

Теперь, когда вы уже знакомы с основными понятиями, необходимыми нам для создания анимации, можно приступать к рассмотрению непосредственно предмета данной статьи.

В начале статьи мы определили, что существует два метода анимации - покадровый и путем создания промежуточных кадров. Итак:

Покадровая анимация

Это анимация, полностью составленная из ключевых кадров. Т.е. вы сами определяете, как содержимое кадра, так и его "длительность" (т.е. сколько таких статических кадров будет занимать изображение).

На временной шкале покадровая анимация выглядит следующим образом:

Покадровая анимация

Достоинства:

- Покадровая анимация дает вам, в некотором смысле, больший контроль над анимацией, и если вы опытный аниматор, вы можете выгодно ею пользоваться.
- Это единственный способ организовать смену абсолютно независимых изображений – слайд-шоу (например, создавая обычный баннер средствами Flash).
- И все остальное, что вытекает из возможности прорисовывать каждый кадр вручную.

Недостатки:

- Покадровую анимацию сложно модифицировать. Особенно, если это не дискретный набор изображений, а связанная анимация. Приходится модифицировать все кадры. На деле, у опытных Flasher-ов, такая ситуация практически не встречается.
- Покадровая анимация занимает достаточно большой объем, так как приходится хранить информацию о каждом кадре.

Анимация с построением промежуточных кадров (tweened motion)

При этом способе анимации Flash автоматически строит промежуточные кадры между ключевыми кадрами, заданными вами. Это означает, что вы рисуете объект, потом на другом кадре производите изменения, о которых мы поговорим ниже, и просите Flash рассчитать те кадры, которые лежат между этими двумя ключевыми кадрами. Он выполняет эту работу, и вы получаете плавную анимацию.

Скорость и плавность анимации зависят от количества кадров, которые вы отводите под движение и скорости вашего Flash фильма (movie). Скорость фильма можно изменить здесь: Modify->Movie:, Ctrl+M - там параметр Frame Rate задает количество кадров в секунду. Для качественной анимации скорость должна быть не меньше 25-30 кадров в секунду.

Плавность и длительность задается количеством кадров, отведенных на анимацию (ее фрагмент). Например, если скорость вашего фильма - 30 кадров/сек., и вам нужно совершить перемещение, скажем, самолетика, из одного угла картинки - в другой за 2.5 секунды, то на это движение вам нужно отвести 75 кадров.

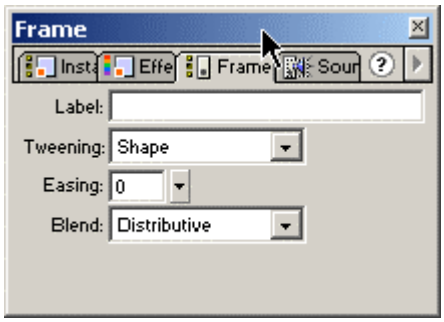
Во Flash существует два варианта построения промежуточных изображений - **motion tweening** (построение анимации на основе модификации символов) и **shape tweening** (построение анимации на основе изменения формы). Эти способы отличаются в корне. Первый используется чаще всего, т.к. с помощью него можно построить подавляющее большинство анимаций. Второй применяется в случаях, когда нужно плавное изменение формы. Поговорим, сначала о нем.

Shape tweening

Скажем, вам нужно, чтобы квадрат плавно превратился в круг, или силуэт кролика плавно перетек в силуэт волка. В этих случаях используется **shape tweening**.

Как обычно, вы задаете два ключевых кадра на некотором расстоянии друг от друга. В этом варианте анимации есть жесткое ограничение: ваша анимация должна занимать отдельный слой и быть единой нарисованной фигурой (не должно быть групп или символов).

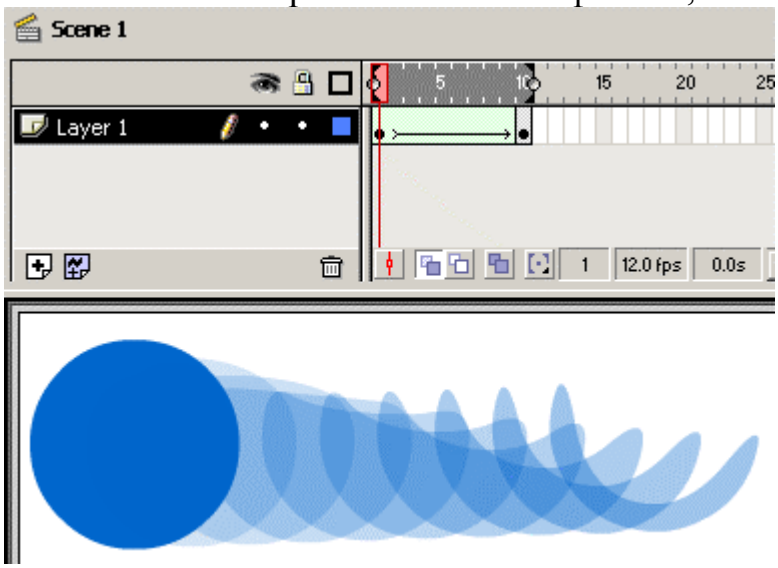
После того как у вас есть два ключевых кадра, вы делаете активным первый из них (просто переходите на него), и выбираете на панели Frame (Windows->Panels->Frame, Ctrl+F) в списке **Tweening** строку **Shape**:



Shape tweening

Кадры на временной шкале должны окраситься в зеленоватый цвет и от первого кадра ко второму должна протянуться стрелочка.

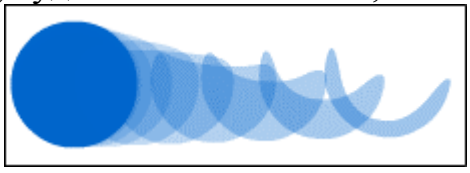
В результате вы получите ряд промежуточных кадров, которые будут отражать переход от первой фигуры ко второй. Я специально включил отображение теней на протяжении всего фильма, чтобы отобразить эти кадры:



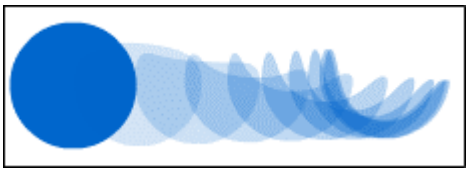
Анимация на основе Shape tweening

В этой маленькой анимации круг переходит в некое подобие полумесяца. На первом ключевом кадре я нарисовал круг, а на втором ключевом кадре (это 10-й кадр сцены) превратил его в полумесяц. Немного о параметрах shape tweening. Вы, наверное, заметили, что появилась пара других параметров, когда вы выбрали shape tweening в панели Frame - **Easing** и **Blend**. Поле **Label** содержит метку кадра. О метках мы поговорим в статье, посвященной анимации с помощью ActionScript.

Easing задает обратное экспоненциальное ускорение. Величина этого параметра может изменяться от - 100 до + 100. Это означает, что если вы зададите отрицательный easing, движение будет происходить с положительным ускорением, скорость будет увеличиваться. И наоборот, если easing будет положительным, анимация будет замедляться.



Easing: -100



Easing: +100

Параметр **Blend**, определяет алгоритм перехода: Distributive (распределяющий, общий) и Angular (угловатый). Первый старается максимально смягчить, сгладить переход от одной фигуры к другой. Второй же пытается сохранить пропорции углов. Если переход вас не удовлетворяет, можно поэкспериментировать с этим параметром.

И, наконец, последний инструмент в анимации shape tweening - **контрольные точки** (shape hints, дословно - подсказки для форм). Это точки, с помощью которых вы помогаете Flash правильно осуществить переход. Без них не обойтись в случае сложных форм. Пользоваться ими очень легко:

На первом ключевом кадре (с которого начинается анимация) вы добавляете контрольную точку (Modify->Transform->Add shape hint, Ctrl+Shift+H). На сцене появится маленькая красная точка, обозначенная буквой латинского алфавита. Вы прикрепляете ее к той части изображения, которая двигается не так, как вы хотели. Затем вы переходите на второй ключевой кадр, и прикрепляете эту же точку к части, в которую должна была перейти часть на начальном кадре. Точка будет уже зеленого цвета, а на начальном кадре она станет желтой. Так вы можете отличать начальные и конечные ключевые точки, так как на одном кадре могут присутствовать и те и другие.

Удалить все точки можно с помощью Modify->Transform->Remove All Hints. Удалить же единственную точку можно, нажав на ней правую кнопку мыши, и в контекстном меню выбрав Remove Hint.

Так как контрольные точки обозначаются буквами латинского алфавита, то их может быть максимум 27.

На рисунках вы можете заметить разницу между кадрами, созданными без использования контрольных точек, и с использованием таковых.

Shape tweening без использования контрольных точек

Shape tweening с использованием контрольных точек

При использовании анимации на основе изменения формы (shape tweening) могут модифицироваться следующие параметры фигуры:

- форма

- расположение
- размер (любые пропорции)
- цвет
- угол поворота

Если вам нужно отключить shape tweening, в панели Frame выберите Tweening: None.

Motion Tweening

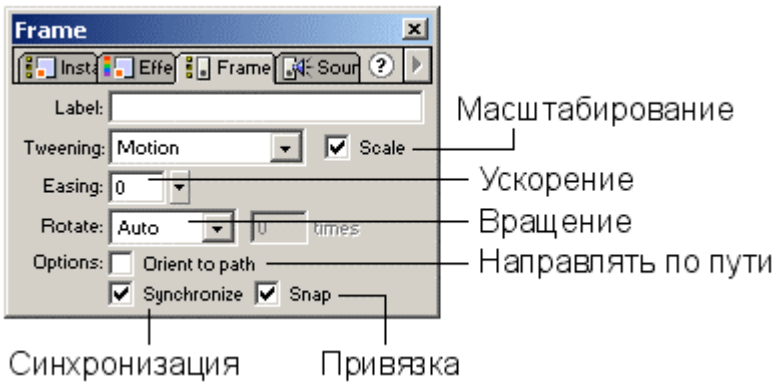
И, наконец, наиболее часто используемая техника анимации во Flash - Motion Tweening. В этом случае анимация строится на основе модификации символов, т.е. объектом анимации является символ.

Как и в анимации shape tweening, на каждый объект в один момент времени, нам нужен один слой. На этом слое должен находиться один символ, с которым и будут происходить все изменения.

Вот какие параметры символа могут модифицироваться при использовании Motion Tweening:

- размер (как пропорционально, так и непропорционально - отдельно высоту и ширину)
- наклон
- расположение
- угол поворота
- цветовые эффекты (см. ниже)
- можно использовать направляющие слои для задания траектории движения объекта

Включить motion tweening можно несколькими способами (а отключить, к сожалению, только одним). Для того, чтобы включить motion tweening, нужно сделать активным начальный кадр вашего перехода, затем, нажав правую клавишу мыши, в контекстном меню выбрать Create motion tween (это же можно сделать, выбрав Insert->Create motion tween). Универсальный способ включения/выключения motion tweening - с помощью панели Frame, выбрав Motion в поле Tweening. Там же можно контролировать параметры анимации:



Motion tweening

Easing - обратное экспоненциальное ускорение, работает абсолютно так же, как и в shape tweening.

Rotate позволяет управлять вращением. **Auto** - Flash автоматически пытается определить количество витков. **CW** (Clockwise, по часовой стрелке) и **CCW** (Counter Clockwise - против часовой стрелки). При этом рядом в поле

справа появляется возможность ввести количество оборотов. Можно использовать только целые значения. Можно отключить вращение, выбрав **None**.

Orient to path - поворачивает символ в соответствии с направляющей линией. **Snap** привязывает символ к этой направляющей. (см. ниже)

В случаях, когда количество кадров основной сцены не бывает кратным количеству кадров символа, флажок **Synchronize** позволяет синхронизировать эти две анимации.

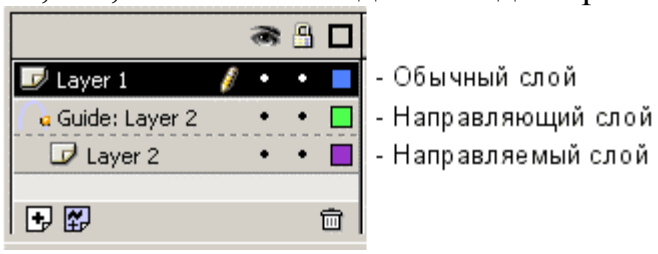
Направляющие слои

Выше было упомянуто о слоях, содержащих траекторию движения, так называемых **направляющих слоях** (guide layers). Это слои, которые содержат кривую, по которой должен двигаться объект.

Скажем, вам нужно анимировать самолетик, который выписывает виражи по небу. У вас уйдет уйма времени и сил, на создание этого движения. При этом анимация будет состоять из маленьких отрезков motion tweening и отдельных кадров. Вместо этого можно нарисовать траекторию на специальном слое и привязать символ самолетика к ней.

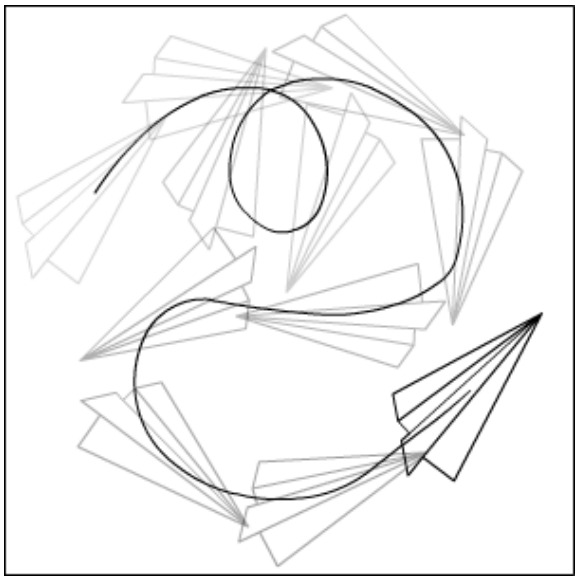
Итак, если вы используете траекторию, то вам нужен дополнительный слой для нее. Кстати, с одной траекторией можно использовать несколько символов.

Для того, чтобы добавить направляющий слой, вам нужно выбрать слой, на котором находится ваш символ, и, нажав правую клавишу мыши, в контекстном меню выбрать Add Guide. При этом исходный слой становится направляемым (guided layer). Это далеко не единственный способ создать направляющий слой (guide layer). Любой слой можно сделать направляющим, указав это в его свойствах, или направляемым, перетащив нужный слой мышкой, так, чтобы он находился под направляющим.



Направляющий слой

Далее, вам нужно нарисовать траекторию движения. Траекторией может быть любая кривая, не являющаяся областью заливки. Управляющий слой готов. Вы можете его запретить для редактирования, чтобы было удобней работать, а в дальнейшем и вовсе сделать его невидимым.



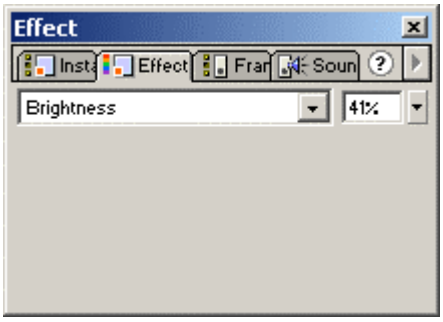
Анимация, с использованием траектории

Теперь, чтобы использовать этот слой, вам нужно взять ваш символ за центральную точку (это такой маленький кружочек) и перетащить ее на траекторию. Вы почувствуете, когда символ "зацепится" за нее, и увидите, как он будет по ней скользить. Далее все по знакомому сценарию - ключевые кадры, нужно включить motion tweening: Если нужно, чтобы объект поворачивался согласно траектории, а не просто двигался по, то на панели Frame нужно включить флажок Orient to path.

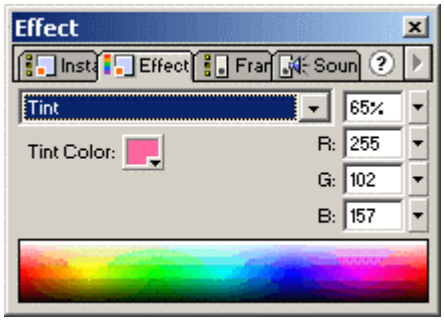
Цветовые эффекты

Motion tweening позволяет использовать различные цветовые эффекты применительно ко всему символу. Эта возможность отсутствует в shape tweening.

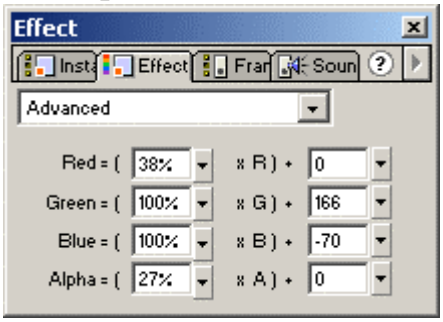
Для того, чтобы применить эффект к символу, нужно выделить этот символ, и на панели эффектов (Windows->Panels->Effects), выбрать нужный эффект



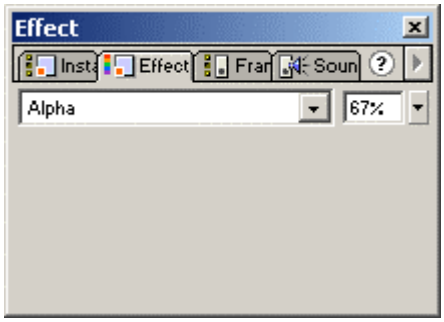
Установка яркости



Цветовое смещение, оттенок



Точная установка всех атрибутов



Установка прозрачности

Основы ActionScript во Flash

ActionScript: событийно-управляемый язык, встроенный во Flash. Последняя версия ActionScript, которая присутствует во Flash 5, существенно отличается от предыдущей. Если в прошлой версии это был ограниченный набор команд, позволяющий осуществлять лишь основные действия и вводимый с помощью не очень удобного интерфейса, то новый ActionScript - это мощный язык с увеличенным набором команд, поддержкой классов, наследования (!) и гораздо более удобным интерфейсом.

ActionScript делает ваши страницы интерактивными. Вы можете реагировать на события с мышки или с клавиатуры, можете выполнить какие-либо действия при проигрывании определенного кадра.

Для того чтобы овладеть ActionScript в полной мере, желательно уже иметь опыт программирования (предпочтительно на C++, JavaScript, etc.). Однако одним из достоинств языка Flash является то, что вам не нужно быть профессионалом во Flash, или полностью знать ActionScript, чтобы писать на нем качественный код. Вы можете использовать лишь те возможности языка, которые сочтете необходимыми для своей работы.

Так как эта статья посвящена основам языка, в ней мы рассмотрим:

- Панель действий (Actions panel), на которой происходит практически все общение с ActionScript.
- Кнопки - как их заставлять работать так, как нам требуется.
- Пути - как обращаться к нужным объектам?
- Основные действия с Flash-мультиками (movie clips) - мы будем управлять процессом проигрывания фильма, как нам угодно.
- Отладку в ActionScript - окошки Output и Debugger.

Цель этой статьи - дать вам возможность почувствовать ActionScript и показать, что этот язык может служить как для создания весьма внушительных программ, так и для выполнения элементарных действий, которые сделают вашу страницу гораздо привлекательней.

Термины

Прежде чем мы перейдем к конкретным действиям, несколько терминов из области ActionScript:

- **Действия (Actions)** - это инструкции, которые говорят Flash-мультику, что делать. От них произошло название языка - ActionScript (дословно - сценарий действий). Давайте договоримся, что в рамках этой статьи мы будем использовать термин "инструкция", дабы не путать их с настоящими действиями, которые будем производить.
- **События (Events)** - это действия, которые происходят, когда проигрывается мультик. События, например, могут происходить, когда заканчивается загрузка какого-то кадра, когда мы достигаем какого-то кадра, когда пользователь нажимает клавишу на клавиатуре или курсор мышки оказывается над нашим объектом.

- **Выражения (Expressions)** - это любая часть инструкции, которая порождает значение. Следующие примеры являются выражениями: `2 + 2`, `2 * 2`, `a + b`, `2*pi*r`, `(15 + k) * random(10)`.
- **Функции (Functions)** - это блоки кода, которые можно многократно использовать. Функциям можно передавать значения и получать от них результат. Например, `number = get_color(15, 24)`. 15 и 24 являются аргументами (или параметрами) функции get_color, возвращаемое значение которой записывается в переменную number.
- **Классы (Classes)** - это типы объектов. Например, класс дерева - растение. Во Flash есть некоторое количество предопределенных классов (очень похожих на классы JavaScript). Вы можете создавать свои классы или модифицировать существующие.
- **Экземпляры (Instances)** - это буквально экземпляры определенных классов. Экземпляр - это уже конкретный реальный объект. Если класс - это определение объекта (экземпляра), то экземпляр - это уже конкретное воплощение, это класс в действии. Каждому экземпляру можно присвоить имя, чтобы через него обращаться к функциям или переменным объекта.
- **Обработчики (Handlers)** - это специальные инструкции, которые обрабатывают события. Например, `onClipEvent` - обработчик действий, связанных с конкретным символом.
- **Операторы (Operators)** - это элементы языка, которые вычисляют значения, исходя из одного или более аргументов. Например, оператор сложения (+) возвращает сумму двух значений, стоящих слева и справа от него.
- **Переменные (Variables)** - это идентификаторы, которые могут хранить значения. Например, a = 5; или name = "Michael".

Панель действий (Actions Panel)

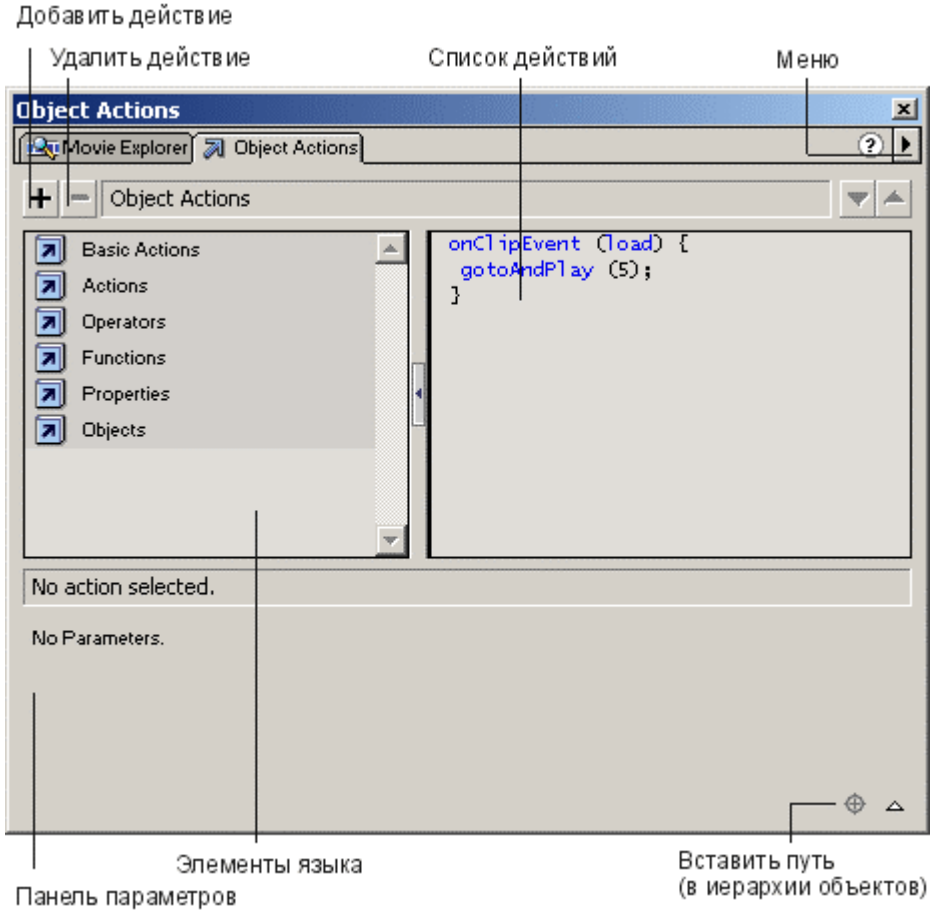
Панель действий служит для отображения и ввода ActionScript-программ. Существует два режима работы с панелью - нормальный (для "чайников") и экспертный. В экспертном режиме список команд - это простое поле для ввода текста. В нормальном же режиме мы не можем напрямую редактировать команды. Для этого используется панель параметров.

Добавить инструкцию можно, нажав на кнопку "+" или выбрав соответствующую инструкцию в списке элементов языка. Кроме того, для всех действий во Flash имеются последовательности клавиш, с помощью которых это можно сделать гораздо быстрее. Они приведены справа от каждого действия в меню кнопки "+". Например, чтобы добавить функцию `stop()`, нужно нажать Esc+st (последовательно: Esc, затем "s", затем "t").

Удалить инструкцию можно, выбрав ее и нажав кнопку "-" (или просто клавишу Delete).

Рекомендуется не начинать сразу же пользоваться экспертным режимом, если у вас нет опыта программирования на C-подобных языках (C++, Java, JavaScript, C#). У нормального режима есть большое достоинство, делающее его незаменимым для новичков - в этом случае гораздо меньше шансов

ошибиться с синтаксисом языка. Новичкам это поможет быстрее понять тонкости ActionScript.



Кнопки

Первое, что хочется, когда начинаешь изучать интерактивность Flash - сделать что-нибудь, что бы откликалось на действия пользователя, "оживить" ваше творение, добавить обратную связь. Самый простой способ сделать это - кнопки. Поэтому с них-то мы и начнем.

Как вы знаете, во Flash существует специальный тип символа для создания кнопок - Button. Будем считать, что вы уже научились создавать кнопки, теперь научимся отслеживать нажатия на эти кнопки.

Кнопки в Macromedia Flash обладают обширным списком событий, на которые мы можем реагировать:

- `press` - клавиша мышки нажата, когда курсор находится в пределах кнопки;
- `release` - клавиша мышки отжата, когда курсор находится в пределах кнопки;
- `releaseOutside` - клавиша мышки отжата, когда курсор находится вне пределов кнопки;
- `rollOver` - курсор мыши входит в пределы кнопки;
- `rollOut` - курсор выходит за пределы кнопки;
- `dragOver` - курсор входит в пределы кнопки, при этом была нажата кнопка, и нажата клавиша мыши;

- `dragOut` - курсор выходит за пределы кнопки, при этом была нажата кнопка, и нажата клавиша мыши;
- `keyPress` ("клавиша") - была нажата "клавиша". Список клавиш можно посмотреть в справке по Flash (объект Key), или использовать панель параметров для ввода нужной клавиши.

К сожалению, Flash "понимает" только левую клавишу мыши. Правая используется для вызова контекстного меню (щелкните правой клавишей на каком-нибудь Flash мультике). Способов отлавливать во Flash среднюю клавишу или "колесико" (mouse wheel) я пока не встречал; думаю, что их не существует.

Перехватываются эти события с помощью директивы `on()`. Синтаксис ее таков:

```
on (событие)
  {
  ... // Наши действия
}
```

Или переход на страничку.

```
on (release)
  {
  getURL("http://rubs.boom.ru");
  }
```

Чтобы проверить этот сценарий, выделите вашу кнопку, нажмите Ctrl+Alt+A и введите программу.

Вот так просто можно перехватить все события, связанные с кнопкой. Ну а как их использовать - это дело исключительно вашего воображения.

Основные действия с Movie Clips

Огромная доля творчества во Flash приходится на манипуляцию символами. Практически все базовые приемы, все трюки и эффекты невыполнимы без этих действий.

С помощью сценариев на ActionScript вы можете выполнять практически любые действия над символами. Надо только помнить, что выполнить эти действия можно только либо в ответ на действие пользователя, либо при наступлении какого-то кадра на временной шкале.

Итак, что же у нас есть? (Я перечисляются только основные. Остальное вы найдете в списке элементов языка или в помощи).

Функции клипов (movie clip), которые можно вызывать:

- **`play()`** - начинает или возобновляет воспроизведение клипа;
- **`stop()`** - останавливает воспроизведение клипа;
- **`gotoAndPlay()`** - переходит на определенный кадр (сцену) и продолжает воспроизведение;
- **`gotoAndStop()`** - переходит на определенный кадр (сцену) и останавливает воспроизведение.

Свойства (параметры) клипов, которые можно считывать/изменять:

- **`_x, _y`** - координаты клипа (в пикселях);

- **_xscale, _yscale** - масштаб клипа (в процентах), соответственно по горизонтали и по вертикали;
- **_width, _height** - ширина и высота клипа (в пикселях);
- **_rotation** - угол поворота клипа (в градусах);
- **_alpha** - прозрачность клипа (в процентах);
- **_visible** - видимость.

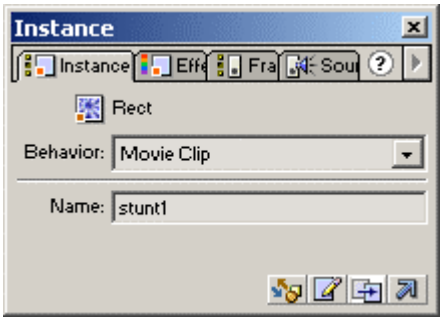
Это далеко не все, что можно делать с клипами. Используйте другие параметры, экспериментируйте, творите!

Имена

Для того, чтобы обращаться к клипам, нам потребуется разобраться с понятием имени объекта (instance name) и пути до объекта (target path). Договоримся, что клип (movie clip) и объект для нас - одинаковые вещи.

Имя объекта - это имя конкретного экземпляра символа. Скажем, у нас может быть символ - машинка, а экземпляры этого символа будут называться "Машинка1", "Машинка2", "Pickup", "Запорожец"…

Для того чтобы дать имя объекту, нужно выделить объект и в панели Instance (Window->Panels->Instance, Ctrl+I) в графе Name ввести имя объекта. Имена могут состоять только из букв, цифр и символа подчеркивания ("_"), причем имя не может начинаться с цифры.



Пути

Путь до объекта - это запись имени объекта с учетом иерархии.

Вы знаете, что во Flash объекты можно "вкладывать" друг в друга, составляя таким образом иерархию. Так вот, эта вложенность обеспечивает не только удобство в обращении с объектами, она еще и ограничивает видимость имен объектов. Видимость ограничивается своим уровнем. Объект может напрямую (по имени) обращаться только к объектам, входящим в него, стоящим на 1 уровень ниже в иерархии.

Для того чтобы обратиться к объекту другого уровня, нужно знать путь до него. Причем путь может указываться как абсолютно (с самого верхнего уровня иерархии), так и относительно (с текущего уровня).

Путь включает в себя объекты, через которые нужно "пройти" по дереву иерархии, чтобы добраться до нужного нам объекта. Имена объектов перечисляются через точку. Кроме того, существует несколько указателей (можно их назвать "виртуальными объектами"), которые часто очень полезны:

this - указатель на "самого себя" (т.е. на текущий объект). Бывает нужен, например, для передачи в функцию указателя на объект, из которого эта функция вызывается.

_parent - указатель на "родителя". Указывает на объект, стоящий в иерархии одним уровнем выше.

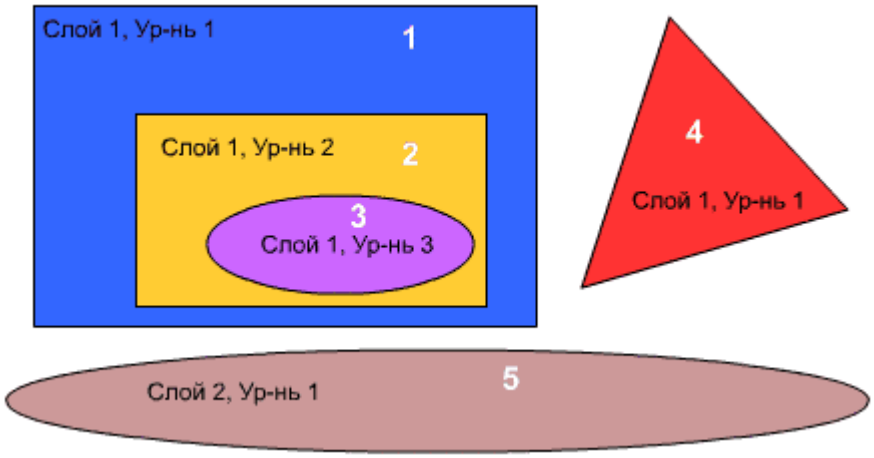
_root - "корень". Это начало иерархии. Без него не обойтись при указании абсолютного пути.

Путь выглядит так:

`leaf.play();` - у подобъекта `leaf` (лист) вызывается функция `play();`

`_parent.tree.leaf.stop();` - подразумевается, что на одном уровне имеется объект `tree`, у которого есть объект `leaf`, у которого и вызывается функция `stop();`

`_root.banner._visible = false;` - сделать клип `banner`, находящийся на 1-м уровне, невидимым.



Для иллюстрации возьмем иерархию из 5-ти объектов. Объекты 1-4 находятся на 1-м слое, объект 5 - на 2-м слое. Объект 2 вложен в объект 1, а объект 3 вложен в объект 2. Объекты на рисунке визуально вложены друг в друга, но это ни в коем случае не означает, что так должно быть и "в жизни". Здесь они так сгруппированы для наглядности. Так как имя объекта не может начинаться с цифры, пусть объекты у нас называются `obj1-obj5`.

Теперь займемся путями. Для начала посмотрим, какие объекты могут обращаться друг к другу по имени. `obj1` может обращаться к `obj2`, а `obj2` - к `obj3`, но при этом `obj1` не может обратиться к `obj3` напрямую, т.к. тот содержится не в `obj1`, а в `obj2`.

Например, первому объекту нужно, чтобы объект 3 начал заново воспроизводиться с 1-го кадра. Вот как это делается:

```
obj2.obj3.gotoAndPlay(1);
```

Чтобы 4-му объекту сделать 1-й объект (заметьте - со всеми подобъектами!) полупрозрачным, ему нужно в своем сценарии написать следующее:

```
_parent.obj1._alpha = 50;
```

или

```
_root.obj1._alpha = 50;
```

Так как `obj4` у нас находится на первом уровне иерархии, то для него `_root` и `_parent` - одно и то же.

Теперь для объекта 3 напишем скрипт, который сделает объект 5 невидимым при нажатии клавиши мыши. В сценарии для объекта 3 пишем:

```
onClipEvent (mouseDown)
  {
 _root.obj5._visible = false;
  }
```

В этом фрагменте мы использовали абсолютный путь. Если бы мы использовали относительный, это выглядело бы как:

```
_parent._parent._parent.obj5._visible = false;
```

Приведенные выше примеры показали не только как выглядят пути, но и как вызываются функции и присваиваются значения свойствам.

Вы можете попробовать использовать обработчик `onClipEvent`, задавая различные условия и выполняя различные действия с объектами при этом.

Одними из самых важных являются функции управления ходом воспроизведения клипа (`play()`, `stop()`, `gotoAndPlay()`, `gotoAndStop()`). Функции `play()` и `stop()` вызываются без параметров, в то время как в goto нужно указывать кадр, и, возможно, сцену.

Отладка в ActionScript

Последнее, что мы рассмотрим в этой статье - окна Output (вывод) и Debugger (отладчик). Это инструменты, служащие для отладки сценариев ActionScript.

Окошко Output пришло из Flash 4, где оно было единственным инструментом для отладки. Существует директива `trace()`, которая выводит сообщения в это окошко. Туда же выводятся сообщения об ошибках.

Использовать trace очень просто:

```
trace ("280-й кадр");
```

или, например,

```
trace (xpos + k);
```

В 5-м Flash появился специальный инструмент - окошечко Debugger. Чтобы им пользоваться, нужно проверять свои фильмы не как обычно (Test movie, Ctrl+Enter), а с помощью Debug movie (Ctrl+Shift+Enter). Окошко Debugger (рис. 4) можно скрыть/показать с помощью Window->Debugger.

За последний год технология Flash завоевала много умов и сердец, в том числе и в России. Для некоторых Flash стал профессией, некоторые использовали его для украшения своих страниц, а некоторые просто цокали языками в восхищении, увидев его интерактивные возможности. Появились книги по Flash, материалы в сети. Можно однозначно сказать, что "дело Flash" в России интенсивно продвигается. Итак…

До сих пор мы задавались вопросом - как сделать? Но просто сделать, это еще не признак мастерства. Настоящий профессионал будет всегда задавать вопрос - как сделать хорошо? Чтобы сделать хорошо, надо досконально знать свой инструмент, обращая внимание на мельчайшие детали, и правильно разрешать компромиссы, касающиеся этих деталей. Речь пойдет об оптимизации во Flash.

Статья поделена на два раздела: один посвящен оптимизации скорости и качества исполнения, другой - оптимизации размера .swf файлов.

Скорость исполнения фильмов

Общеизвестно, что плавность анимации достигается большим количеством проигрываемых кадров в секунду (fps - frames per second). Следовательно, мы стремимся указывать большую скорость в свойствах Flash-фильма. (По умолчанию, Flash использует значение 12 fps. Для качественной анимации требуется минимум 25-30 fps).

Однако, увеличение количества кадров в секунду требует большей производительности компьютера, на котором исполняется анимация, и если ее не хватает, Flash сокращает частоту кадров. Поэтому, даже если мы установим fps равным 100, Flash будет исходить из возможностей, имеющихся для воспроизведения.

Размеры и "количество" анимации

Имеются способы повысить fps. Первый, наверное, не самый выгодный - уменьшить размеры воспроизводимого клипа: на слабой машине клип с размерами 300x200 будет воспроизводиться гораздо лучше, чем, скажем, клип с размерами 600x400.

Второй заключается в нескольких простых правилах составления анимации:

1) Самое простое: *чем больше у вас анимирующихся объектов в сцене, тем сложнее Flash обрабатывать их, и тем медленней будет воспроизведение*.

2) *Выгодней использовать один символ, содержащий мелкие объекты, чем много символов, отдельно для каждого объекта*. В качестве примера можно привести имитацию снега на клипе. Много символов, каждый из которых будет отвечать за отдельную снежинку, будут двигаться гораздо медленней, чем один символ, отвечающий за весь снег.

3) *Не держите символы на сцене, если вы их не используете*. К сожалению, Flash не настолько интеллектуален, чтобы не просчитывать клипы с `_alpha` или `_vizible` равными нулю. Если клип невидим, лучше его убрать со сцены, а потом, когда будет нужно, его показать.

Имейте ввиду эти моменты, когда создаете анимацию во Flash.

Качество

Вы, наверное, уже знакомы с понятием качества во Flash (опция Quality при публикации, параметры `_quality`, `_highquality`, функция `toggleHighQuality()`). Качество тоже сильно влияет на скорость воспроизведения. Тут тоже существует несколько компромиссов.

Во первых, уровень качества можно установить вручную при экспорте. При этом не забывайте, что если не отключить контекстное меню, то у пользователя остается возможность регулировать качество.

Во вторых, качество можно менять динамически во время исполнения анимации (параметр `_quality`). На время воспроизведения особенно сложных и "быстрых" фрагментов, можно понижать качество, тем самым

выигрывая в скорости, а когда "количество" анимации уменьшится, вновь возвратиться к высокому уровню качества.

При создании анимационных заставок, можно давать выбор качества пользователю, поместив в уголок пару/тройку кнопок, позволяющих регулировать качество.

Поэкспериментируйте, и вы увидите, что качество существенно влияет на скорость воспроизведения клипов.

Поточное воспроизведение и предварительная загрузка

Flash, специально приспособленный под Сеть, является поточным форматом. Это означает, что фильмы Flash могут начать воспроизводиться, не загрузившись до конца. С одной стороны, это преимущество, т.к. фильм начинает воспроизводиться достаточно рано, и пользователю нет нужды ждать конца загрузки. С другой стороны, если канал, по которому передаются данные, окажется уже, чем нужно для передачи Flash-потока, анимация будет приостановлена, и пользователю, как при просмотре "Санта-Барбары", на самом интересном месте придется ждать "следующей серии".

Это еще один компромисс, который нужно учитывать. Если для вас незначительны такие задержки, или у вас есть уверенность, в том, что каналы связи не подведут - вам не о чем беспокоиться. Но если вам хочется, чтобы ваш клип воспроизводился без задержек, тогда его надо снабдить предварительным загрузчиком (preloader) - это прием, позволяющий задержать воспроизведение до полной загрузки фильма.

Имеет смысл протестировать работу вашего фильма на нескольких скоростях с помощью функции Test Movie (Control->Test Movie, Ctrl+Enter). Скорость можно выбирать в меню Debug, которое появляется после запуска фильма.

Поточный звук

Последний аспект, который, относится к производительности Flash анимации - это поточный звук. Во Flash есть несколько способов синхронизации звука. Если вы используете достаточно продолжительный фрагмент музыки (или любого другого звукового сопровождения), и события анимации должны совпадать с событиями звука, то вам нужно использовать синхронизацию Stream (поток). В этом случае вся анимация будет синхронизирована со звуком, и два этих потока будут идти параллельно, не отставая друг от друга.

Размер .swf файлов

Наверное, основным достоинством Flash, как векторного формата, является маленький размер файла: чем меньше файл, тем быстрее он загружается по сети, тем больше информации можно передать. К сожалению Flash не является высокоразвитым инструментом и не умеет оптимизировать размер файлов. Однако, если мы знаем как он работает, нам легче создавать наши Flash-творения в соответствии с теми требованиями, которые у нас есть. В этом разделе мы рассмотрим аспекты, влияющие на размер Flash файлов.

Повторное использование, символы

Символы во Flash - мощный способ экономии места. Они позволяют использовать повторно любые фрагменты вашего творчества. Если вы используете что-либо хотя бы два раза, сделайте это символом.

Например, вам нужно нарисовать множество разноцветных мячиков разного размера. Используйте один символ для всех мячиков. Вы можете изменить размер и цвет каждого экземпляра этого символа и получить нужную сцену. Это займет гораздо меньше места, чем если бы для каждого мячика выделялся бы один символ.

Смысл символов - экономия. Подумайте, какие общие свойства имеются у объектов в вашем фильме, и вынесите их в отдельный символ.

Графика, созданная вручную

Когда вы создаете графику во Flash, убедитесь в том, что у вас нигде не осталось лишних линий, пустых или прозрачных фигур, которые не несут на себе функциональной нагрузки.

Ограничьте себя в использовании специальных типов линий, таких как пунктир, нечеткие линии, точки и т.д. Сплошные линии занимают меньше места. Толстые линии, нарисованные карандашом, занимают гораздо меньше, чем линии, нарисованные кистью.

Когда вы импортируете векторную графику, убедитесь, что в ней не существует скрытых линий или объектов.

Flash позволяет сглаживать, выпрямлять и оптимизировать линии (меню Modify -> Smooth, Straighten, Optimize). Чем прямее линии, тем меньше места они занимают. И, наоборот, чем они детальнее, тем больше. Оптимизируя линии, можно задать уровень сглаживания, а так же выполнить многопроходную оптимизацию.

Текст, шрифты

По умолчанию, Flash преобразует все используемые буквы (слово "символ" не используется, чтобы не путать с известным термином Flash) в полигоны. Это означает, что для фразы "Жили-были дед да баба" Flash сохранит начертания букв "Жилбыдеа-". Каждая буква сохраняется в виде полигона, который потом размножается нужное количество раз. Причем, если мы напишем ту же фразу другим шрифтом, Flash будет вынужден сохранить начертания букв этого шрифта тоже. Представьте, сколько места займет описание целого шрифта, в случае если мы задействуем весь алфавит (плюс ко всему, Flash различает прописные и строчные буквы)!

Есть способ избежать сохранения шрифта, правда придется пожертвовать сглаженными краями букв. Для этого нужно выбрать Use Device Fonts в опциях текста. При этом Flash будет сохранять не начертания букв, а только характеристики и название шрифта (на практике это всего несколько байт). При воспроизведении будет использован указанный шрифт, либо, если такого шрифта не окажется в системе, Flash использует ближайший по характеристикам шрифт.

Отсюда выводы: большие объемы текста лучше не хранить во Flash, (а использовать, например, HTML) - Flash подходит больше для коротких надписей, лозунгов и т.п.; стараться использовать как можно меньше

различных шрифтов. Если уж очень нужно поместить большое количество текста во Flash, используйте опцию Use Device Fonts. Все это сократит размер создаваемого файла.

Звук

Очевидно, что использование звука сильно увеличивает размер Flash-файлов. Звук тоже поддается оптимизации во Flash.

В параметрах публикации (Publish Settings) можно установить тип компрессии и качество звука. В большинстве случаев имеет смысл использовать компрессию MP3, выбирая качество "по потребности". Чем шире поток (bit rate), тем больше места занимает звук.

В общем случае, на размер звуковых данных влияют частота дискретизации (sample rate) и количество каналов (стерео, моно). Понятно, что стерео звук будет занимать в два раза больше места, чем моно, и что звук оцифрованный с частотой дискретизации 44 kHz, ровно во столько же раз больше отрывка с частотой 22 kHz. Частоту дискретизации относительно исходной можно понизить, повышать ее не имеет смысла (лучше от этого звук не станет).

Еще один способ сократить затраты на звук - проигрывать один и тот же фрагмент несколько раз. Например, вам нужен фрагмент звуков джунглей, длиной 40 секунд. Вы можете взять фрагмент, длиной 10 секунд и повторить его несколько раз, при этом будут сохранены только требуемые 10 секунд записи.

Изображения

Так как Flash является векторной средой, по возможности следует использовать векторную графику. К тому же, растровая графика при масштабировании выглядит не привлекательно.

Если изображение не является фотографическим, имеет смысл преобразовать его в векторный формат. В параметрах изображения, после его импортирования, можно выбрать тип компрессии и посмотреть, как оно будет выглядеть.

Накладно и не оправдано создание во Flash анимации с помощью растровых изображений. Для этого можно использовать, скажем, формат GIF.

Отчет

Flash может генерировать отчет, в котором по байтам расписан весь фильм. (Publish Settings -> Flash -> Generate size report). Для примера, рассмотрим мультик:

В этом фильме 70 кадров. В отчете будет отражено количество байт, нужное для каждого кадра (Frame Bytes), и размер фильма к этому кадру (Total Bytes). Исходя из этих данных можно рассчитать требуемую скорость линии передачи, чтобы Flash-фильм мог отображаться без задержек.

После отчета по кадрам идет отчет по сценам - какая сцена сколько занимает, а затем - по символам. 136 байт на первой (и единственной) сцене занимает фраза "(C). Rouben Sardarian, 2001". Этот текст использует опцию Use Device Fonts, и поэтому занимает мало места.

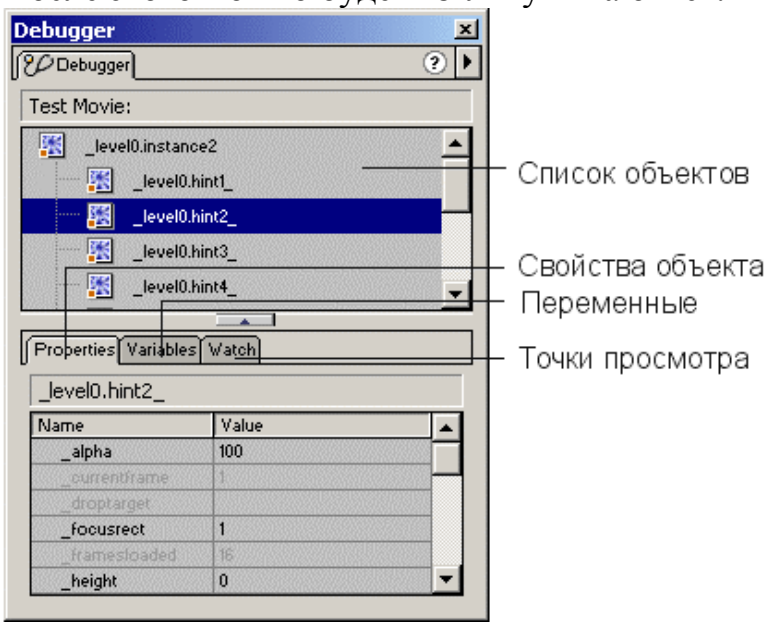
Слово "Optimized" и буквы слова "Flash" являются символами, причем на каждую букву этого слова отводится два символа - один содержит графику,

другой анимацию (движение вверх-вниз). Заметьте, что буквы, использованные в слове "Flash" не используют шрифт (см. конец отчета - там нет шрифта Arial, которым это слово было написано). Эти буквы были преобразованы в "самостоятельную" графику после ввода текста (Modify->Break Apart, Ctrl+B).

В конце отчета помещаются данные о шрифтах, звуке и изображениях. Звук и импортированные изображения отсутствуют в нашем примере, поэтому в отчете данных о них нет.

Зато мы можем увидеть, что описание шрифта для фразы "(C). Rouben Sardarian, 2001" занимает 20 байт. Прибавив к этому 136 байт, которые описывают фразу, получаем 156 байт. Сравните это с 519 байтами (470 байт - 8 букв, из которых состоит слово "Optimized" + 49 байт описание слова).

После этого можно будет взглянуть на отчет.



В одной части окна Debugger находится иерархический список объектов, используемых в фильме. Выбрав объект, можно просматривать его свойства (закладка Properties).

Под закладкой Variables находятся все переменные. Преимущество закладки Variables состоит в том, что вы можете модифицировать значения любых переменных "на лету" и тут же получать отражение этого изменения в фильме.

И, наконец, можно добавить любые переменные в список просмотра (Watch list) и наблюдать за их значениями (закладка Watch).

Debugger позволяет отслеживать практически любые параметры Flash-фильмов. Тем не менее, что свое применение есть и у окошка Output, и у Debugger-а.

OOP Flash

Суть объектной модели в Action Script (как и в JS) в том, что любой объект имеет ссылку на свой прототип, как бы «лекало», содержание которого (параметры и методы) объект наследует.

Простой пример.

```
a = {} //создаем новый пустой объект.
a.__proto__ = {a:"a"} // вместо существующего прототипа
//подставляем новосозданный объект с параметром a, равным строке «a».
b = {} // создаем второй объект.
b.__proto__ = a // в качестве прототипа подставляем первый объект.
trace(b.a) // делаем проверку на наличие параметра прототипа у второго объекта.

a.__proto__.a = "c"
// меняем значение поля у прототипа первого объекта.
trace(b.a) // тестируем изменения во втором объекте.
```

__proto__ - это ссылка на прототип объекта.

Вместо цепочки наследований можно делать деревья со сложной структурой, когда поля и методы одного объекта наследуют сразу несколько объектов.

Остановимся на механизме наследования. Важно понять, что методы и параметры объектов не копируются из прототипа. Просто при работе с объектом происходит поиск вызываемого параметра или метода. Поиск производится как в параметрах и методах самого объекта, так и в полях прототипа (а также в прототипе прототипа, и так далее). Это происходит всегда. Только чаще прототип пуст и его полей мы не видим. Поэтому, как только мы меняем объект-прототип, мы меняем объекты-наследники.

Второй момент объектной модели флэша, который нужно понять, это работа с конструкторами. Хоть в нашем предыдущем примере мы обошлись без конструктора, конструктор был задействован автоматически.

Конструктор – это функция. Любая функция во флэш может быть использована в качестве конструктора.

Используя конструкцию данного вида, мы создаем новый объект.

```
any_new_object = new constructor_name();
```

Мы, не задумываясь, используем данную конструкцию, создавая массивы, например.

Любая функция имеет ссылку на некий объект, который будет прототипом для объектов, которые будут созданы функцией. Изначально

данный объект пуст. Изменяя его или подставляя уже готовый объект, мы сможем создавать объекты с нужным прототипом.

```
    //**************************************************
**********************
    function constructor_one()
    {
                this.a = "a"
                this.b = "b"
                this.method = function()
                {
                            trace("one")
                }
    }
    //**************************************************
**********************
    one = new constructor_one();
```

Мы создали объект с пустым прототипом с двумя полями и одним методом.

Теперь изменим прототип конструктора.

```
    constructor_one.prototype.new_param = "new_param";
```

Создадим второй объект.

```
    two = new constructor_one();
    trace(two.new_param) // трейсится «new_param»
```

Второй объект имеет дополнительный параметр «new_param», наследованный от прототипа.

Проверяем первый объект на наличие нового поля и убеждаемся, что оно также присутствует.

Мы так же можем не просто модифицировать прототип конструктора, но и подменять его на готовые объекты. Но наследоваться новые прототипы будут только в новосоздаваемых объектах.

Попробуем сделать подмену прототипа у конструктора. Вернемся к моменту, когда был создан первый объект.

```
    function constructor_one()
    {
                this.a = "a"
                this.b = "b"
```

```
                this.method = function()
                {
                        trace("one")
                }
    }
    //************************************************
**********************
    one = new constructor_one();
    constructor_one.prototype = {param:”any_value”}; //
делаем подмену прототипа
```

Создадим второй объект. И проверим первый и второй объект на наличие параметра «param».

```
    two = new constructor_one();
    trace(one.param)
    trace(two.param)
```

Первый объект имеет то же количество полей, что и при создании. И поля «param» он не имеет.

Почему же, изменяя прототип конструктора, созданные до этого объекты наследуют изменения, а при подмене нет? Все просто. В первом случае мы модифицируем объект, на который имеют ссылку все созданные объекты и конструктор. При подмене же конструктор уже имеет ссылку на другой объект-прототип и ничего уже существующими объектами не наследуется. Казалось бы, элементарно, но эта тонкость создает путаницу у начинающих.

Повторюсь. Передача ссылки на объект-прототип от конструктора к создаваемому объекту происходит в момент создания объекта.

Чтобы «синхронизировать» прототипы двух созданных нами объектов, нам нужно скопировать ссылку с прототипа второго объекта на первый объект.

```
    one.__proto__ = two.__proto__;
```

Такой вопрос. Где лучше создавать методы и параметры объектов. В прототипах или в конструкторах. Ответ – в прототипах. Методы и параметры не дублируются в каждом объекте. Их легче менять и не тратится лишняя память.

Для работы с объектами и с прототипами у нас есть еще одно ключевое слово constructor. Мы можем им пользоваться, например, тогда, когда мы не знаем, какой конструктор у объекта, и мы хотим создать другой объект, используя тот же конструктор.

```
three = new two.constructor();
```

И естественно, есть ключевое слово this. Но я думаю, если вы читаете данную статью, вам значение данного слова объяснять не нужно.

А как же «менять акции???». Очень просто. Обращаться к встроенным объектам, к их прототипам и менять то, что нужно.

Простой пример, меняющий акцию «play» на «stop», и наоборот.

```
MovieClip.prototype.stop_old = MovieClip.prototype.stop;
MovieClip.prototype.play_old = MovieClip.prototype.play;
MovieClip.prototype.stop = MovieClip.prototype.play_old;
MovieClip.prototype.play = MovieClip.prototype.stop_old;
```

А вот классный пример, который убыстряет работу метода split объекта String почти в десять раз:

```
st.OldSplit = st.split;//Creat Link to old split function
//**********************************************************
st.split =  function(str)
{          //Creat new split function
                              if(str.length>1)return this.OldSplit(str);
                if (str == "" || str == null) return this.toCharArray();
                var result = new Array();
                var d = this.length;
                var n = 0;
                var tmpStr = "";
                var ch;
                while(n<d)
                {
                            //ch = this.charAt(n++);
                            ch = substring(this ,++n, 1);
                            if(ch==str)
                            {
                                        result.push(tmpStr);
                                        tmpStr="";
                            }else{
```

```
                                        tmpStr += ch;
                                }
                }
                result.push(tmpStr);
                return result;
    }
    //************************************************
**********************
    st.toCharArray = function()
    {
                var st = this;
                var arr_result = new Array();
                for (var s=0;s < st.length;s++)
                {
                                arr_result.push(substring(th
is, s+1, 1));
                }
                return arr_result;
    }
```

И так до победного конца. Удачи.

Механизм паузы. Дополнительный метод к объекту мувиклип.

Сразу опишем методы и работу с ними

Новые методы:

```
childs()
allChilds()
pause()
```

childs()

Метод возвращает массив включенных в мувиклип дочерних мувиклипов.

```
myMc.childs();
```

Если у мувиклипа есть три дочерних мувиклипа, то метод выдаст массив из трех элементов.

allChilds()

Метод возвращает массив включенных в мувиклип дочерних мувиклипов и их детей. Возвращает все дерево.

```
myMc.allChilds();
```

Например, если сделать `_root.allChilds()`, то метод возвратит все мувиклипы, которые есть в `_root`.

pause();

Метод останавливает проигрывание всех дочерних мувиклипов (все дерево) того мувиклипа, чей метод был вызван, включая его самого. Повторный вызов метода pause() включает анимацию во всех остановленных клипах.

```
myMc.pause();
```

Если вызвать `_root.pause();` то будет остановлен весь ролик. Метод создан для игр, в которых нужно реализовать временную остановку игры, а также для блокирования выполнения ресурсоемких отрезков программной анимации, когда постоянное проигрывание их будет вредно для других процессов.

Примечание. В данном методе обрабатываются только те клипы, которые не были до этого остановлены методами объекта мувиклип: stop, gotoAndStop, prevFrame, nextFrame. Или те, которые были сначала остановлены, а потом были вызваны методы: gotoAndPlay и play. То есть метод `pause` *не будет считать остановленным клип, который был остановлен через tell target или из самого клипа акциями Flash 4. Другими словами, чтобы не запустить при отмене паузы клипы, которые стояли в вашем ролике, вы не должны пользоваться tell target и, при остановке клипа из собственных фреймов, вы должны ставить перед акциями: stop, play, gotoAndStop, gotoAndPlay, prevFrame, nextFrame путь к объекту* - `this`.

```
    play();                 =>              this.play();
stop();                 =>              this.stop();
gotoAndStop();          =>          this.gotoAndStop();
gotoAndPlay();          =>          this.gotoAndPlay();
prevFrame();            =>            this.prevFrame();
nextFrame(); => this.nextFrame();
```

Суть проблемы в том, что акции stop(), gotoAndStop(), prevFrame(), nextFrame(),: gotoAndPlay() и play() работают в двух разных режимах. При их вызове без указания на объект, они уже не являются методами объекта MovieClip, а вызывают одноименные акции более ранних плэйеров (flash 2-4). Это сделано для обратной совместимости с более старыми версиями флэш-плэйеров.

Для более полного понимания данной проблемы прочитайте дальнейшее описание.

Описание механизма работы методов

Начнем с постановки задачи.

Необходимо создать метод объекта MovieClip, который остановит анимацию самого клипа и всю иерархию вложенных в него мувиклипов.

Повторный вызов метода должен включить анимацию во всех остановленных клипах.

Для того чтобы узнать все включенные в мувиклип дочерние мувиклипы, создадим метод объекта MovieClip childs().

Чтобы создать новые методы, мы должны изменить прототип объекта MovieClip.
Для этого обратимся к прототипу. `MovieClip.prototype`

Так как ссылка на прототип нам будет нужна в дальнейшем, создадим переменную, хранящую путь к изменяемому объекту. `var mc = MovieClip.prototype;`

Создаем новый метод объекта childs().

```
mc.childs = function(){
var result_childs = new Array();
for(var v in this){
if(typeof(this[v])=="movieclip"){
result_childs.push(this[v]);
}
}
return(result_childs)
}
```

Данный метод через цикл `for in` перебирает вложенные в родительский клип объекты и если это movieclip, добавляет его ссылку к массиву. По окончании выполнения метода возвращается массив, в котором хранятся ссылки на вложенные мувиклипы. Очень простая функция.

Данный метод возвращает только ссылки на мувиклипы находящиеся на одном уровне иерархии ниже главного. А нам нужно все уровни. А если включенный в главный мувиклип имеет включенные в него мувиклипы?

Поэтому создадим другой метод, который вернет все дерево. Старый оставим. Возможно, что нам иногда нужно будет узнать только один уровень вложенности.

Создадим метод allChilds().

```
mc.allChilds = function(){
if(_root.allChilds_root_movieclip == null)_root.allChilds_root_movieclip = this;
this.allChilds_tested = true;
var result_allChilds = new Array();
for(var v in this){
if(typeof(this[v])=="movieclip"){
if(this[v].allChilds_tested || this[v]._parent != this)continue;
result_allChilds.push(this[v]);
result_allChilds = result_allChilds.concat(this[v].allChilds());
}
}
if(_root.allChilds_root_movieclip == this){
for (var allChilds_i=0;allChilds_i < result_allChilds.length;allChilds_i++){
delete result_allChilds[allChilds_i].allChilds_tested;
}
}
return(result_allChilds);
}
```

Здесь нам не обойтись без рекурсии. allChilds(), как и childs(), проходит по всем мувиклипам и добавляет их в массив, но она в тоже время вызывает тот же метод (allChilds) в найденном мувиклипе. Тот в свою очередь просматривает свои мувиклипы и вызывает тот же одноименный метод у каждого. В конечном итоге все уровни вложений мувиклипов собираются в одном массиве. Красиво и удобно.

Но… Рекурсия очень опасна. Она может зациклиться. То есть, если мувиклип хранит ссылку на другой мувиклип, то в мувиклипе, куда ведет ссылка, есть ссылка на предыдущий мувиклип. Поиск будет идти по кругу. Бесконечная рекурсия.

Сделаем защитные механизмы. Во-первых, создадим флаг, который показывает, что у мувиклипа уже был произведен поиск. `this.allChilds_tested = true;`

По окончании выполнения поиска флаги удалим. Для этого нам нужно узнать, закончен ли поиск. Поиск будет закончен, если закончится выполнение метода клипа, который главный в иерархии.

Ставим в начале:

```
if(_root.allChilds_root_movieclip == null)_root.allChilds_root_movieclip = this;
```

В конце проверяем, является ли мувиклип, в котором выполняется метод, главным и если да, то проходим по массиву мувиклипов и удаляем флаги.

```
if(_root.allChilds_root_movieclip == this){
for (var allChilds_i=0;allChilds_i < result_allChilds.length;allChilds_i++){
delete result_allChilds[allChilds_i].allChilds_tested;
}
delete _root.allChilds_root_movieclip;
delete allChilds_tested;
}
```

Первая проблема решена.

Вторая проблема. Нам не нужны ссылки на клипы, которые не являются детьми мувиклипов, в которых производится поиск. Иначе мы выйдем за пределы нужной нам иерархии.

Объединим две проверки в одну.

```
if(this[v].allChilds_tested || this[v]._parent != this)continue;
```

Цикл возвращается к проверке условия, и опасные действия не производятся.

Итак, с поиском дочерних клипов разобрались. Пока все идет неплохо :).

Вернемся к паузе. Мы должны остановить анимацию только в тех клипах, в которых есть анимация. Это понятно. Когда мы будем отменять паузу, нам не хотелось бы запустить анимацию в мувиклипах, которые до паузы не проигрывались.

Для того чтобы знать проигрывается клип или нет, мы должны ввести флаг в объект MovieClip, который бы показывал, есть ли в нем анимация. Мувиклип по умолчанию проигрывается, поэтому значение флага будет **true**.

```
mc.played = true;
```

Теперь нам нужно отслеживать, остановилась ли анимация. Для этого мы должны переработать методы, которые останавливают анимацию. Это stop(), gotoAndStop(), prevFrame(), nextFrame(). При вызове этих методов мы поменяем значение параметра played на **false**.

```
mc.old_gotoAndPlay = mc.gotoAndPlay;
mc.gotoAndPlay = function(){
this.gotoAndPlayed = true;
this.old_gotoAndPlay(arguments[0],arguments[1]);
}
mc.old_stop = mc.stop;
```

```
mc.stop = function(){
this.played = false;
this.old_stop();
}
mc.old_gotoAndStop = mc.gotoAndStop;
mc.gotoAndStop = function(){
this.played = false;
this.old_gotoAndStop(arguments[0],arguments[1]);
}
mc.old_prevFrame = mc.prevFrame;
mc.prevFrame = function(){
this.played = false;
this.old_prevFrame();
}
mc.old_nextFrame = mc.nextFrame;
mc.nextFrame = function(){
this.played = false;
this.old_nextFrame();
}
```

В методах, которые запускают анимацию (play() и gotoAndPlay()) поменяем значение параметра played на **true**.

Теперь создадим сам метод.

```
mc.pause = function(){
if(this.paused_mc_array != null){
for (var pause_i=0;pause_i <
this.paused_mc_array.length;pause_i++){
this.paused_mc_array[pause_i].play();}
delete this.paused_mc_array;
}else{
var pause_array = this.allChilds();
pause_array.push(this);
this.paused_mc_array = new Array()
for (var pause_i=0;pause_i <
pause_array.length;pause_i++){
if(pause_array[pause_i].played &&
pause_array[pause_i]._totalframes != 1){
pause_array[pause_i].stop();
this.paused_mc_array.push(pause_array[pause_i]);
}
}
}
}
```

Все достаточно банально. Узнаем все дочерние мувиклипы(allChilds()). Добавляем главный клип в массив мувиклипов. Проходим по массиву циклом.

Проверяем мувиклипы. Нас интересуют те, которые проигрываются (played) и у которых больше, чем один кадр (`_totalframes != 1`).

Если клип нам подходит, помещаем его в массив останавливаемых мувиклипов. И останавливаем в нем анимацию (`stop()`).

При повторном вызове (`this.paused_mc_array != null`) метода проходим по массиву остановленных мувиклипов и включаем анимацию (`play()`). Удаляем массив.

Проще не бывает :).

Применений данному методу масса. Можно «замораживать» анимацию на одной ветке иерархии, а на другую включать… Или, например, выполняется сложный код, хотелось бы подождать его выполнение, обработать и опять пустить анимацию. Или вот у меня была проблема, окно в фоне тормозило процессы в других окнах. Теперь все просто. Привязываем onBlur в JS к функции, которая вызывает остановку анимации во флэше. А когда onFocus, запускаем анимацию. И, естественно, данный метод полезен в играх. Ну, это можно не рассказывать.

Удачи.

Справка. "AS_3W_MC_Library.as"

Как пользоваться

Для того чтобы разместить данную библиотеку в ваш проект, вам нужно воспользоваться акцией #include. Поместите файл библиотеки в папку с вашим рабочим Flash файлом и в первом (или другом) кадре Flash файла и вставьте данную акцию:
`#include "AS_3W_MC_Library.as"`

Важно, чтобы этот кадр не проигрывался в дальнейшем. В противном случае постоянное выполнение кода библиотеки может замедлять проигрывание вашего SWF.

Теперь Action Script нашей библиотеки будет встраиваться при экспорте вашего Fla в Swf. При размещении вашего ролика в Интернет вам файл библиотеки вам не понадобится, скрипт уже в Swf.

Скрипт в библиотеке добавляет новые методы для работы с мувиклипами. Он также улучшает работу с некоторыми методами.

После ее добавления в ваш файл каждый клип (включая _root) имеют эти методы, и вы можете их вызывать как обычные методы.

myMc.method([args]);

Методы

position----------

```
movieTo(x,y);
movieBy(xOffset,yOffset);
```

size ----------

```
size(W,H);
```

scale ----------

```
scale(scale);
scaleTo(xscale,yscale);
scaleBy(ratio);
```

rotate ----------

```
rotateBy(angle);
```

visible ----------

```
hide();
show();
visInvert();
```

mirrors ----------

```
flipV();
flipH();
```

animation----------

```
playOffset(Offset)
```

Я делал методы для действий, которые больше всего встречаются и упрочение работы с которыми значительно ускорит работу с клипами.

Итак, у вас есть некий мувиклип под названием "myMc" и вы хотите с ним работать.

Для работы с координатами клипа я создал два метода - **movieTo()** и **movieBy().**

movieTo(x,y);

Предположим, что вам нужно разместить ваш мувик в нужную точку.

Для этого вам было бы нужно присвоить двум его координатам нужное значение.

Предположим, что вам нужно поместить клип в 200 по оси икс и 300 по оси игрик.

```
myMc._x = 200;
myMc._y = 300;
```

В принципе просто, но теперь еще проще.

```
myMc.movieTo (200,300);
```

Если вы вызываете данный метод из изменяемого клипа, то достаточно просто:

```
movieTo (200,300);
```

Если вы делали данную процедуру уже миллион раз, вы оцените.

Не нужно заводить функцию, которая обрабатывала данное действие, не нужно передавать ей путь к клипу. Каждый клип (включая _root) уже имеет нужный метод и может его вызывать.

movieBy(xOffset,yOffset);

В работе нам часто нужно переместить клип на несколько пикселей относительно его текущего положения.

Например, я хочу переместить клип на 3 пикселя вверх и на двадцать вправо.

```
myMc.movieBy(20,-3);
```

size(W,H)

Метод используется для задания ширины и высоты клипа.

Например, я хочу установить ширину клипа в 200 пикселей, а высоту в 100

```
myMc.size(200,100);
```

Если вы захотите поменять только один параметр, вам нужно будет в место неизменяемого значения поставить 0 или null.

scale(persent);

Очень часто нам нужно поменять значения scale, при этом значение по иксу и по игрику одинаковое. Для этого вам нужно присвоить одно и то же значение для параметров _yscale и _xscale. С методом scale все все проще.

Например, я хочу присвоить клипу myMc значение scale, равное 50 процентов.

```
myMc.scale(50);
```

scaleTo(xscale,yscale);

Если вам нужно присвоить разные значения. то для этого вам нужно воспользоваться методом scaleTo.

Присваиваем клипу значения scale по горизонтали 50 и по вертикали 30.
```
myMc.scaleTo(50,30);
```

Если вас интересует только один параметр, параметр неизменяемый должен быть равен 0 или null.

scaleBy(ratio);

Часто бывает ситуация, когда вы хотите поменять значения горизонтальной и вертикальной scale, умножив его на какой-то числовой коэффициент.

Например, я хочу уменьшить клип на 90 процентов:
```
scaleBy(0.9);
```

rotateBy(angle);

Изменение значения вращения клипа - в принципе, простая процедура, поэтому я создал метод только для случая, когда нужно изменить вращение клипа на нужный шаг.

Поменяем значение вращения клипа на 12 градусов,
```
myMc.rotateBy(12);
```

если нужно повернуть клип против часовой стрелки, то приращение должно быть отрицательным.
```
myMc.rotateBy(-12);
```

hide();

Очень простое, но нужное действие, которое делает клип "невидимым" (_visible = false)
```
myMc.hide();
```

show();

Противоположное действие, но такое же частое и необходимое. "Сделать видимым"
```
myMc.show();
```

visInvert();

Очень полезное действие, которое позволит вам "невидимые" сделать "видимыми", а "видимые" наоборот. Инвертирует значение видимости. `myMc.visInvert();`

flipV();

Если вы часто делаете "зеркальное отражение" клипа относительно вертикали, то вы оцените полезность данного метода. `myMc.flipV();`

flipH();

То же самое, но по горизонтали. `myMc.flipH();`

playOffset(Offset)

Очень часто нам нужно отправить анимацию клипа относительно текущей в какой-нибудь кадр. Чтобы сделать как можно менее обременительной данную процедуру, написан этот метод.

Отправим анимацию на два кадра назад. `myMc.playOffset(-2)`

Пока это все методы

Пока это все методы. Немного, но все крайне полезные.

Если вы регулярно будете заходить на наш сайт, то вы сможете получить свежую версию данной библиотеки и новые библиотеки, но уже для других объектов и действий.

Описанные выше методы могут применяться для любого клипа, включая _root.

"Редизайн" встроенных методов

Встраиваемые во Flash библиотеки позволяют не только создавать новые методы для встроенных объектов. Немаловажно, что можно их настраивать для собственных задач. Это отдельная и большая тема. В данной библиотеке решается одна проблема, которая меня лично сильно раздражает.

При создании клипа методом attachMovie() функция не возвращает линк на созданный клип. Приходится писать вторую строку, которая вычисляет ссылку.

Меня лично это расстраивает крайне. Поэтому я добавил к методу attachMovie return, который возвращает ссылку на клип.

И вместо двух строк:

```
myMc.attachMovie("IdClip","newName",depht);
newClip = myMc["newName"];
```

Достаточно одной:

```
newClip = myMc.attachMovie("IdClip","newName",depht);
```

Упражнение. SetTimeOut()

```
function setTimeOut(funcString,timeOut,inOut){
var McTimeOut = _root.MySTM;

if(inOut){
var tmpArr = new Array();
tmpArr = funcString.split("(")
var funcLink = eval(tmpArr[0]);
var funcArgs = tmpArr[1];
funcArgs = funcArgs.substr(0,funcArgs.length - 1);
var tA = new Array();
tA = funcArgs.split(",");
funcLink(tA[0],tA[1],tA[2],tA[3],tA[4],tA[5],tA[6],tA[7]
,tA[8],tA[9]);

}else{

McTimeOut.n++;
McTimeOut.STM.duplicateMovieClip("STM"+McTimeOut.n,McTim
eOut.n);

var newTarg = McTimeOut["STM"+McTimeOut.n];

if(typeof(timeOut)=="string"){
timeOut = Number(timeOut);
newTarg.frame = true;
}else{
newTarg.time = getTimer();
}

newTarg.funcString = funcString;
newTarg.timeOut=timeOut;
}
```

```
}
```

Универсальным мувиклипом-триггером выступает клип лежащий в корневом каталоге с "инстансом" "MySTM"

```
var McTimeOut = _root.MySTM;
```

Триггер включает в себя клип "STM" имеющий два кадра с кодом:

```
if(_name == "STM"){
stop();
}else if(frame){
nTime++;
if(nTime>=timeOut){
_root.setTimeOut(funcString,timeOut,true);
removeMovieClip (this);
}
}else if(getTimer()-time>=timeOut){
_root.setTimeOut(funcString,timeOut,true);
removeMovieClip (this);
}
```

Вот и все.

Работает это примерно так:

```
SetTimeOut("myFunction(12,5)",1000)
```

То есть вызывается функция "myFunction" через одну секунду.

Если нужно вызвать функцию через какое то число проигранных кадров, то пишем вот так:

```
SetTimeOut("myFunction(12,5)","3")
```

То есть вызывается функция "myFunction" через три кадра.

Отличие в том, что для микросекунд второй аргумент - номер, а для кадров - стринг.

При использовании функции не пользуйтесь кавычками и пробелами. Например, в таком виде будет сбой:

```
SetTimeOut("myFunction (12, 5)","5")
```

Я сначала сделал удаление кавычек и пробелов, но это лишний раз затормаживает работу плэйера. Для себя же пишем.

Упражнение. Тасование колоды во Flash5

```
    function                                    randomDeck(){
var                                                      r;
for                                        (i=NCards;i>0;i--){
r             =              Math.floor(Math.random()*i);
CardDeck.push(CardDeck[r]);
CardDeck.splice(r,1);
}
}
```

Есть еще более простой вариант, но мне кажется, что первый все- таки более интересен. Хотя конечно, второй проникает в душу своей простотой :)

```
function                                    randomDeck(){
CardDeck.sort(ranSort);
}
function                     ranSort                    (){
return                     Math.floor(Math.random()*3)-1;
}
```

Примеры

Создание меню при помощи MC.play();

Выберите инструмент «квадратик» и нарисуйте прямоугольник, после этого выберите инструмент «черн. стрелочка» и выделите прямоугольник, при этом прямоугольник выделяется «сеточкой». После этого выберите пункт меню Insert->Convert to Symbol (или нажмите горячую клавишу F8). В появившемся диалоговом окне выберите тип символа (Behavior) Button. Полученная кнопка выделяется голубой рамкой. Скопируйте эту кнопку в буфер обмена (Edit->Copy или Ctrl+C). Два или три экземпляра этой кнопки вставьте на сцену (Edit->Paste или Ctrl+V) и расположите по сцене выровняв их друг относительно друга. Эти две-три кнопки и будут выполнять роль подпунктов меню. При помощи курсора мыши выделите вставленные кнопки.

Преобразуйте выделенные кнопки в новый символ - MovieClip (Insert->Convert to Symbol или F8). Три кнопки объединятся в одну рамочку и будут представлять собой единый символ, обладающий набором свойств как объект в программировании. В дальнейшем для обращения к этому муви-клипу (MC) необходимо задать ему имя (Instance name). При этом имя MC не должно начинаться с цифр. Instance name можно указать выбрав Window->Panels->Instance. Для примера назовем MC «menu».

Нажмите правой клавишей мыши на одной из кнорочек, входящих в состав MC и выберите пункт контекстного меню Edit in place и вы перейдете в

режим редактирования МС. Как вы помните, каждая отдельная анимация осуществляется в отдельном слое. Поэтому необходимо создать число слоев на один меньше чем количество кнопок в МС для того чтобы задать корректно анимацию для каждой из них. Если кнопок у нас три, то соответственно требуется создать еще два слоя.

Переместим каждую кнопку в свой слой. Для этого выделим одну из трех кнопок и вырежем ее в буфер обмена (Edit->Cut или Ctrl+X). Нажатием левой клавиши мышки перейдем в первый ключевой кадр второго слоя и вставим кнопку на то же место на сцене, откуда мы ее скопировали (Edit->Paste in Place или Ctrl+Shift+V). Вторую кнопу аналогичным образом переместим в третий слой.

Создадим ключевые кадры в следующем порядке: в первом слое - на 10 кадре (нажмите левой клавишей мышки на 10 кадре первого слоя и выберите пункт меню Insert->Keyframe или нажмите горячую клавишу F6, или нажмите правой клавиши мыши и в контекстном меню выберите Insert Keyframe), во втором - на 15, и в третьем - на 20 кадре. При этом вы будете видеть всего одну кнопу в МС. Это связано с тем, что время существования первого и второго слоев меньше, чем у третьего слоя. Следовательно, продлим время существования первого и второго слоев до 20 кадра (нажмите левой клавишей мышки на 20 кадре и выберите пункт меню Insert->Frame или нажмите горячую клавишу F5, или нажмите правой клавиши мыши и в контекстном меню выберите Insert Frame) – на экране появятся все три кнопки.

Нажатием левой клавиши мыши по первому ключевому кадру в первом слое перейдем в него, при этом у вас выделится первая (самая верхняя) кнопочка.

Откройте панель эффектов Window->Panels->Effect. Выберите пункт Alpha: 100% - объект непрозрачен, 0% - объект прозрачен полностью. Выставите полную прозрачность объекта, при этом, вместо кнопки вы увидите прозрачную рамку. Аналогичные операции повторите для второй и третей кнопок. При этом все три кнопки у вас исчезнут, и в первом кадре вы будете видеть крестик, символизирующий собой центр редактируемого вами МС.

Зададим анимацию. Для этого нажмем левой клавишей мыши на первом кадре в первом слое и выбирем панель Frame (Window->Panels->Frame). Выбирем анимацию Motion (Tweening->Motion) и поставим одно вращение по часовой стрелке (Rotate->CW, 1). Во втором слое все тоже самое, но два вращения, в третьем слое – три вращения.

Как вы помните, Flash имеет особенность, состоящую в том, что проигрывание анимации происходит автоматически сразу после загрузки ролика, и продолжается до бесконечности. Следовательно, возникает

необходимость принудительно заставить открываться меню по нашему желанию. Это можно осуществить следующим образом. Нажмите правой клавишей мыши на 1 ключевом кадре третьего слоя и выберите пункт Actions. Нажмите на знак «+», выберите Basic Actions->Stop, при этом в поле для скрипта у вас появится следующая запись:

```
stop();
```

Аналогичную операцию повторить для 2 ключевого кадра в третьем слое.

Вернемся на сцену 1 (нажатием на надпись Scene 1 в верхнем левом углу над Timeline). При этом вы увидите, что три пункта подменю не видны, видна только рамка голубого цвета с крестиком посередине.

Теперь необходимо на кнопку, являющуюся пунктом меню, написать скрипт, управляющий МС. Для этого нажмем правой клавишей мыши на кнопке и выбирем пункт Actions. . Нажмите на знак «+», выберите Objects->MovieClip->play. В окне скрипта появится следующая надпись:

```
on (release) {
.play();
}
```

При этом .play(); выделится красным цветом. Это говорит о том, что данная строчка скрипта содержит ошибку. Ошибка заключается в том, что не указан объект, к котором применяется метод play(). Необходимо указать имя объекта. Для этого нажмите клавишу Home на клавиатуре или аккуратно нажмите левой клавишей мыши перед точкой. В правом нижнем углу есть кнопка Insert Target Path, выполненная в виде окружности с перекрестием. По нажатию на нее появится список объектов, из которых мы выбираем «menu». Обратите внимание, что возможны две моды обращения к МС: абсолютная адресация (Absolute) и относительная (Relative). В нашем случае выбирем абсолютную адресацию и нажмем ОК.

```
on (release) {
_root.menu.play();
}
```

Все. Теперь просмотрите готовое меню при помощи Ctrl+Enter.

Создание меню через MC.nextFrame();

Выберите инструмент «квадратик» и нарисуйте прямоугольник, после этого выберите инструмент «черн. стрелока» и выделите прямоугольник, при этом прямоугольник выделяется «сеточкой». После этого выберите пункт меню Insert->Convert to Symbol (или нажмите горячую клавишу F8). В появившемся диалоговом окне выберите тип символа (Behavior) Button. Полученная кнопка выделяется голубой рамкой. Скопируйте эту кнопку в буфер обмена (Edit->Copy или Ctrl+C). Два или три экземпляра этой кнопки вставьте на сцену (Edit->Paste или Ctrl+V) и расположите по сцене, выровняв их друг относительно друга. Эти две-три кнопки и будут выполнять роль подпунктов меню. При помощи курсора мыши выделите вствленные кнопки.

Преобразуйте выделенные кнопки в новый символ - MovieClip (Insert->Convert to Symbol или F8). Три кнорки объединятся в одну рамочку и будут представлять собой единый символ, обладающий набором свойств как объект в программировании. В дальнейшем для обращения к этому мувиклипу (MC) необходимо задать ему имя (Instance name). При этом имя MC не должно начинаться с цифр. Instance name можно указать выбрав Window->Panels->Instance. Для примера назовем MC «menu2».

Нажмите правой клавишей мыши на одной из кнорочек, входящих в состав MC, и выберите пункт контекстного меню Edit in place и вы перейдете в режим редактирования MC.

Создайе 2 ключевой кадр нажатием клавиши F6. У вас получатся два рядом стоящих ключевых кадра.

Нажатием левой клавишей мыши в 1 ключевой кадр MC перейдите на него и нажмите клавишу Delete, удалив содержимое 1 ключевого кадра (подпункты меню).

Нажмите правой клавишей мыши в 1 ключевом кадре и выберите Actions. Далее нажмите на «+», выберите Basic Actions->Stop, при этом в поле для скрипта у вас появится следующая запись:

```
stop();
```

Вернитесь в режим редактирования сцены. Вместо MC вы увидите крестик.

Напишем управляющий скрипт на кнопке. Нажмем правой клавишей мыши на единственно видимой кнопке (если кнопок больше, чем одна, значит вы сделали что-то неправильно) и выберем пункт Actions. Нажмите на знак «+», выберите Objects->MovieClip->nextFrame();. В окне скрипта появится следующая надпись:

```
on (release) {
.nextFrame();
}
```

При этом .nextFrame(); выделится красным цветом. Необходимо указать имя объекта. Для этого нажмите клавишу Home на клавиатуре или аккуратно нажмите левой клавишей мыши перед точкой. В правом нижнем углу есть кнопка Insert Target Path. По нажатию на нее появится список объектов, из которых мы выбираем «menu2». Выбирем абсолютную адресацию и нажимаем ОК.

```
on (release) {
_root.menu.nextFrame();
}
```

Проверим работоспособность этого скрипта (Ctrl+Enter). По нажатию на пункт меню появится подменю. По повторному нажатию на него же ничего не произойдет.

Требуется доработать скрипт так, чтобы при повторном нажатии подпункты меню исчезали. Для этого перейдите в режим редактирования скрипта на управляющей кнопке. Нажмите на знак «+», выберите Actions->if и в условии для if (Condition) напишите !a. После этого снова нажмите на «+», выберите Actions->else и при помощи стрелочек «вверх» («вниз») переместите строчку _root.menu.nextFrame(); таким образом, чтобы она находилась в фигурных скобочках сразу после if.

Нажмите левой клавишей мыши на else, после этого нажмите на «+», выберите Objects->MovieClip->prevFrame и поставьте перед этим методом имя МС, к которому будете обращаться.

Теперь необходимо изменить значение переменной a, которую мы обрабатываем в условии if. Для этого нажмем левой клавишей мыши на предпоследнюю фигурную скобку, нажмем на «+», выберем Actions->set variable. В поле Variable указываем переменную a, в поле Value указываем 1-a и напротив этого поля ставим галочку в пункте Expression.

Особенно хотелось отметить тот факт, что переменные во Flash изначально равны нулю, чем мы в данном случае и воспользовались. Также хочется отметить, что одна и та же переменная может выступать в роли логической, целочисленной и строковой переменной одновременно без какого-либо переопределения.

В результате проведенной доработки скрипта было получено следующее:

```
on (release) {
  if (!a) {
    _root.menu2.nextFrame();
  } else {
    _root.menu2.prevFrame();
  }
a=1-a;
}
```

У вас получилась работающее меню.

Доработаем получившееся меню так чтобы по нажатию на подпункт меню запускался созданный вами мувиклип.

Создадим требуемый мувиклип – например, движущийся шарик. Для этого выберем инструмент «кружок» и нарисуем на сцене шарик. Преобразуем его в мувиклип - Insert->Convert to Symbol (или F8) и дадим ему имя «klip» Перейдем внутрь мувиклипа (контекстное меню->Edit In Place) и зададим для кружка Shape-анимацию.

Перейдите внутрь мувиклипа «menu2» на второй ключевой кадр (в котором содержатся подпункты меню). Выберите одну из кнопок и вызовите панель акций (контекстное меню -> Actions). Нажмите на «+» выберите Actions->on. В поле Event уберите галочку на Release и поставьте на Roll Over. Нажмите на знак «+», выберите Objects->MovieClip->play. В окне скрипта появится следующая надпись:

```
on (rollover) {
.play();
}
```

Поставьте курсор перед .play(); и выберите путь к мувиклипу «klip» при помощи перечеркнутого кружка в правой нижней части панели Actions.

```
on (rollover) {
        _root.klip.play();
}
```

Запустите ваш мувиклип. Теперь у вас при нажатии на подпункт меню выполняется мувиклип с именем «klip», но при этому подпункты меня остаются на экране.

Добавим скрипт, скрывающий ваши подпункты меню. Для этого перейдем на сцену, выделим там кружек, обозначающий мувиклип «menu2», перейдем в панель Actions и выберем «+»->Actions->onClipEvent, поставим галочку Mouse Up. Далее зададим для мувиклипа «menu2» переход в 1 кадр, где пункты меню отсутствуют. Выберем Objects->MovieClip->prevFrame, а далее путь к этому мувиклипу. Далее Action->set variable. В поле Variable напишем _root.a, в поле Value – 0 обязательно поставив галку перед Expression. При этом в поле для скрипта у вас появится следующая запись:

```
onClipEvent (mouseUp) {
        _root.menu2.prevFrame();
        _root.a = 0;
}
```

Последняя строчка скрипта `_root.a = 0;` означает, что мы обнуляем переменную a, отвечающую за переход между кадрами в мувиклипе «klip» содержащим и не содержащим подпункты меню. Если этого не сделать, то при повторном нажатии на управляющую кнопку меню – подпункты не появятся.

Теперь меню готова.

GetProperty

Научимся получать свойства МС с использованием как функции GetProperty, которая осталась для обратной совместимости с предыдущими версиями пакетов, так и при помощи явной передачи параметров объекта.

Для начала вытащим на сцену мышку из общей библиотеки (Window-> Common Libraries->Graphics->Mouse). Для того чтобы перетащить мышку на сцену достаточно захватить название (Mouse) или картинку перетаскиваемого объекта левой клавишей мыши и потащить на сцену, а затем отпустить.

После этого необходимо преобразовать мышку в кнопку для этого необходимо выделить эту мышку при помощи инструмента «черная стрелочка» и нажать горячую клавишу F8 и в варианте выбора типа объекта поставить Button – т.е. кнопка.

После того как проделана эта операция в скрипте для кнопки необходимо записать следующее:

```
on (release) {
   _root.xpos = getProperty ( _target, _x );
   _root.ypos = getProperty ( _target, _y );
```

```
    _root.xscale = getProperty ( _target, _xscale );
    _root.yscale = getProperty ( _target, _yscale );
    _root.alpha = getProperty ( _target, _alpha );
    _root.rotation = getProperty ( _target, _rotation );
}
```

этим скриптом мы получаем позицию текущего МС на экране по осям X и У (для адресации используется ссылка _target), его масштабирование, поворот, прозрачность. Эти свойства передаются в сцену при помощи переменных.

Теперь создадим МС, свойства которого мы будем передавать. Для этого выделите полученную кнопку при помощи инструмента «черная стрелочка» и нажмите горячую клавишу F8, а в варианте выбора типа объекта поставьте Movie Clip.

Теперь на сцене у вас находится МС, внутри которого находится кнопка
Скопируйте МС и разместите его 3-5 экземпляров на сцене и измените им свойства как то масштабирование по осям X и У, прозрачность.

Создайте на сцене 6 динамических текстовых полей по количеству предаваемых в _root переменных (Window->Panels->Text Options) и задайте полям значения переменных xpos, ypos, xscale, yscale, alpha, rotation (поле в Text Options Variable), т.е. точно такие же, как и было указанно в скрипте, написанном выше.

После этого, запустите режим TestMovie (Ctrl+Enter) и проверьте работоспособность своего скрипта.

Проделайте тоже самое, но в скрипте используйте явное задание переменных, используя свойства объекта и указатель на текущий объект this

```
on (release) {
    _root.xpos = this._x;
    _root.ypos = this._y
    _root.xscale = this._xscale;
    _root.yscale = this._yscale;
    _root.alpha = this._alpha;
    _root.rotation = this._rotation;
}
```

Все, пример выполнен.

SetProperty

Проделаем аналогичный пример с мышами, но теперь свойства мышей мы будем задавать самостоятельно.

Вытащим на сцену мышку из общей библиотеки (Window-> Common Libraries->Graphics->Mouse). Для того чтобы перетащить мышку на сцену достаточно захватить название (Mouse) или картинку перетаскиваемого объекта левой клавишей мыши и потащить на сцену, а затем отпустить. Для выполнения данного примера потребуется всего одна мышка.

Назовем эту мышку сразу Movie Clip-ом и зададим ей **Instance Name** в панели Instance – mouse.

Теперь необходимо создать кнопки, по нажатию на которые можно менять свойства мышки и текстовые поля (Input text), в которые будут задаваться новые свойства.

поле xpos, рядом с ним кнопка, в скрипте которой написано:

```
on (release) {
    setProperty ("_level0.mouse", _x, xpos);
}
```

поле ypos и кнопка со скриптом:

```
on (release) {
    setProperty ("_level0.mouse", _y, ypos);
}
```

поле xscale и кнопка со скриптом:

```
on (release) {
setProperty ("_level0.mouse", _xscale, xscale);
}
```

поле yscale и кнопка со скриптом:

```
on (release) {
```

```
setProperty ("_level0.mouse", _yscale, yscale);
}
```

поле alpha и кнопка со скриптом:

```
on (release) {
setProperty ("_level0.mouse", _alpha, alpha);
}
```

поле rotation и кнопка со скриптом:

```
on (release) {
setProperty ("_level0.mouse", _rotation, rotation);
}
```

поле visibility и кнопка со скриптом:

```
on (release) {
setProperty ("_level0.mouse", _visibility, visibility);
}
```

Теперь запустите тестовый режим и проверьте как у вас все здорово и замечательно работает☺.

Научитесь использовать адресацию МС и задание свойств явным образом без использования функций. Для этого перепишите скрипты следующим образом:

Во всех случаях вместо, напимер:

```
setProperty ("_level0.mouse", _visibility, visibility);
```

укажите либо:

```
_root.mouse._visibility = visibility;
```

либо:

```
_level0.mouse._visibility = visibility;
```

Если скрипт продолжает работать, значит вы все указали верно.

Путь к объектам

Рассмотрим на примере более подробно обращение к объектам (MC) и переменным, а также пути к ним.

Во Flash MC можно «вкладывать» друг в друга, составляя, таким образом, некое подобие иерархической структуры. Эта вложенность обеспечивает удобство в обращении к объектам и управлении их свойствами, она еще и ограничивает видимость имен объектов. Видимость ограничивается своим уровнем. Объект может напрямую (по имени) обращаться только к объектам, входящим в него, стоящим на 1 уровень ниже в иерархии. Для того чтобы обратиться к объекту другого уровня, нужно знать путь до него. Причем путь может указываться как абсолютно (с самого верхнего уровня иерархии), так и относительно (с текущего уровня). Путь включает в себя объекты, через которые нужно «пройти» по иерархическому дереву, чтобы добраться до нужного нам объекта, перечисленные через точку (как вы помните, во Flash 4 подобное обращение осуществлялось через слеш). Кроме того, существует несколько указателей, которые часто применяются для более эффективного обращения к MC:

this - указатель на «самого себя» т.е. на текущий объект. Бывает, нужен, например, когда требуется передать в функцию указатель на объект, из которого эта функция вызывается. Мы им уже пользовались, когда смотрели свойства мышек.

_parent - указатель на «родителя». Указывает на объект, стоящий уровнем выше в иерархии.

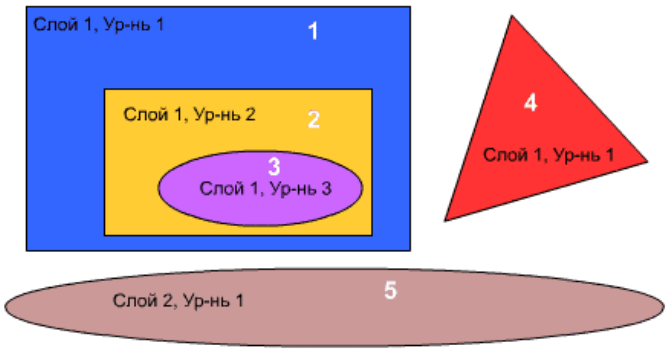
_root – «корень». Это начало иерархической структуры. Без него практически не обойтись при указании абсолютного пути. Однако еще в Flash 4 использовалось обращение через указание уровней.(_level0).

Путь выглядит, например, так:

leaf.play(); - у подобъекта leaf (лист) вызывается функция play();

_parent.tree.leaf.stop(); - подразумевается, что на одном уровне имеется объект tree, у которого есть объект leaf, у которого и вызывается функция stop();

_root.banner._visible = false; - сделать клип banner, находящийся на 1-м уровне, невидимым.



Попробуйте создать несколько MC и через кнопки обратиться к ним описанным выше образом.

Для иллюстрации возьмем иерархию из 5-ти объектов. Объекты 1-4 находятся на 1-м слое, объект 5 - на 2-м слое. Объект 2 вложен в объект 1, а объект 3 вложен в объект 2. Объекты на рисунке визуально вложены друг в друга, но это ни в коем случае не означает, что так должно быть и «в жизни». Здесь они так сгруппированы для наглядности. Так как имя объекта не может начинаться с цифры, пусть объекты у нас называются obj1-obj5.

Теперь займемся путями. Для начала посмотрите, какие объекты могут обращаться друг к другу по имени. obj1 может обращаться к obj2, а obj2 - к obj3, но при этом obj1 не может обратиться к obj3 напрямую, т.к. тот содержится не в obj1, а в obj2.

Скажем первому объекту нужно, чтобы объект 3 начал заново воспроизводиться с 1-го кадра. Вот как это делается:

```
obj2.obj3.gotoAndPlay(1);
```

Чтобы 4-му объекту сделать 1-й объект (заметьте со всеми подобъектами) полупрозрачным, ему нужно в своем сценарии написать следующее:

```
_parent.obj1._alpha = 50;
```

или

```
_root.obj1._alpha = 50;
```

Т.к. obj4 у нас находится на первом уровне иерархии, то для него _root и _parent - одно и тоже. Теперь для объекта 3 напишите скрипт, который сделает объект 5 невидимым при нажатии клавиши мыши. В сценарии для объекта 3 пишем:

```
onClipEvent (mouseDown) {
_root.obj5._visible = false;
}
```

В этом фрагменте использовался абсолютный путь. Для использования относительного пути, необходимо написать следующее:

```
_parent._parent._parent.obj5._visible = false;
```

Попробуйте активно пользоваться этим в дальнейшей работе.

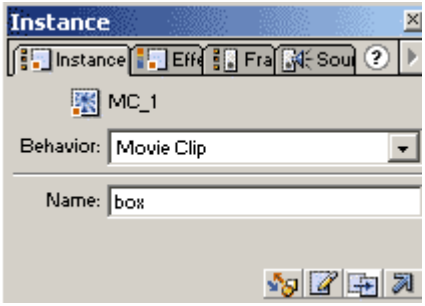
DuplicateMovieClip();

duplicateMovieClip – создание копии конкретного MC, используя Instance или указатель на текущий MC. Хотелось бы продемонстрировать несколько примеров создания и использования этого метода. Он присутствует в двух ипостасях: во-первых, осталась акция от скрипта Flash 4, которая находится в Action, во вторых, это метод для управления MC. Итак:

Пример № 1

Создадим MC, Например, нарисовав квадратик, выделив его при помощи инструмента «черный курсор» и нажав горячую клавишу F8, выбрав при этом Movie Clip.

Для дублирования MC необходимо, как и говорилось выше, задать Instance (Instance Name), что легко осуществить в панели Instance. Если она недоступна, необходимо ее вывести на экран (Window->Panels->Instance).



Теперь создайте кнопку, по нажатию на которую будет происходить дублирование MC. Для этого нарисуйте квадрат, выберите инструмент «черная стрелочка» и выделите получившийся квадрат, после чего необходимо нажать горячую клавишу F8 и выбрать пункт Button – то есть кнопка.

Для вызова функции **duplicateMovieClip** по нажатию на кнопку, скрипт должен иметь вид:

```
on (release) {
duplicateMovieClip ("box", "box2", 1);
}
```

box – **Instance** – дублируемого MC, должно полностью совпадать с именем, которое было задано в поле Instance.

box2 – имя нового MC

1 – Depth – глубина нового MC, глубина с номером 0 занята оригиналом. Требуется обратить внимание на то, что в приведенном случае после запуска тестового Movie, по нажатию на кнопку у вас не произойдет видимого изменения на сцене, поскольку один MC с именем «box2» будет продублирован в позицию с теми же координатами, что и оригинал, т.е. необходимо для наблюдения изменить координату при помощи акции **setProperty** или явного задания свойств _x и _y.

Пример № 2

На кнопке необходимо дописать существующий скрипт, который должен теперь выглядеть следующим образом:

```
on (release) {
duplicateMovieClip("box", "box2", 1);
setProperty ("box2", _x, "200");
}
```

или

```
on (release) {
duplicateMovieClip("box", "box2", 1);
_root.box2._x=200;
}
```

Обе эти записи скрипта эквивалентны полностью, за исключением того, что, используя адресацию _root.box2._x, существует возможность сколь угодно сложного обращения.

Используя метод **duplicateMovieClip** к клипу box, можно записать следующий скрипт:

```
on (release) {
_root.box.duplicateMovieClip("box2", 1);
box2._x=200;
}
```

Запишем этот скрипт на кнопке. Итак, был продублирован один МС, теперь необходимо создать целую серию продублированных объектов.

Пример № 3

Создадим скрипт, при помощи которого можно продублировать сразу несколько МС. При этом же необходимо посмотреть циклы:

```
on (release) {
count = 1;
while (count<20) {
_root.box.duplicateMovieClip("boxx"+count, count);
count += 1;
}
}
```

Как видно из приведенного скрипта, используется цикл while do, который проверяет выполнение условия count<20, аналогичным образом можно применить цикл for который мне в силу привычки нравится больше ☺

```
on (release) {
for (count=1;count<20, count++) {
_root.box.duplicateMovieClip("boxx"+count, count);
}
}
```

Хочется отметить, что для динамического получения имени продублированного МС можно применять также следующие варианты

```
Newname=”box”+String(count);
```

Или

```
Newname=”box”&counter;
```

Как вы, наверное, догадались, counter играет роль счетчика, при помощи которого динамически задается имя нового МС и динамически задается глубина каждого из них – «depth». Необходимо отметить, что для каждого МС ДОЛЖНА быть своя глубина.

Запишем этот скрипт на кнопке.

Пример № 4

Этот пример демонстрирует использование случайных значений (рандомизации) с использованием функции **random()** и задания при помощи введенной рандомизации случайных свойств продублированных МС. Каждый продублированный МС получает свои случайные свойства:

_x – позиция по оси X,
_y – позиция по ости У,
_xscale – масштабирование по оси X,
_xscale – масштабирование по ости У,
_alpha - прозрачность

Приведенный ниже скрип демонстрирует управление указанными свойствами МС. Данный скрипт необходимо написать на кнопке, по нажатию на которую у вас происходит дублирование МС.

```
on (release) {
count = 1;
while (count<20) {
_root.box.duplicateMovieClip("boxx"+count, count);
_root["boxx"+count]._x = random(550);
_root["boxx"+count]._y = random(150);
_root["boxx"+count]._xscale = random(150);
_root["boxx"+count]._yscale = random(150);
_root["boxx"+count]._alpha = random(100);
count += 1;
}
}
```

Если задать анимацию для вашего МС, то вы сможете наблюдать очень симпатичную картинку на экране

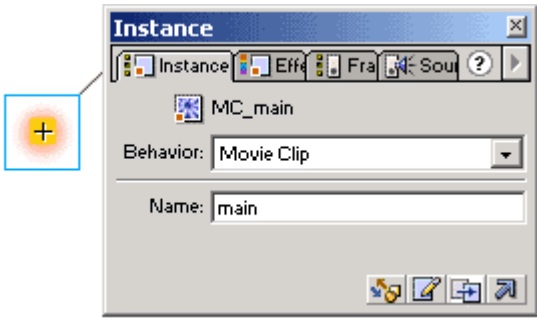
attachMovieClip

attachMovie – это метод, загружающий и запускающий МС из Библиотеки (Library) как только поступил запрос. При этом МС, загруженный

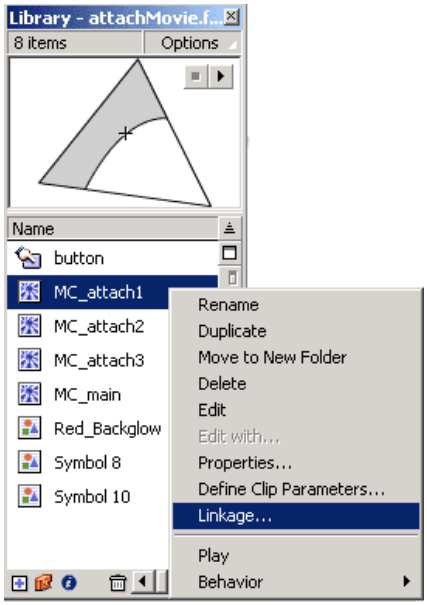
на сцену получает свое уникальное имя, по которому к приаттаченному MC можно будет обратиться и заставить его выполнить определенную последовательность действий или задать то или иное свойство и глубину (depth). В общем случае обращение к MC выглядит следующим образом:

someMCInstanceName.attachMovie(**idName**, **newname**, **depth**);

someMCInstanceName – это instance name MC, к которому приаттачивается MC. В нашем случае мы имеем Movie Clip (MC_main) с instance name – «main»



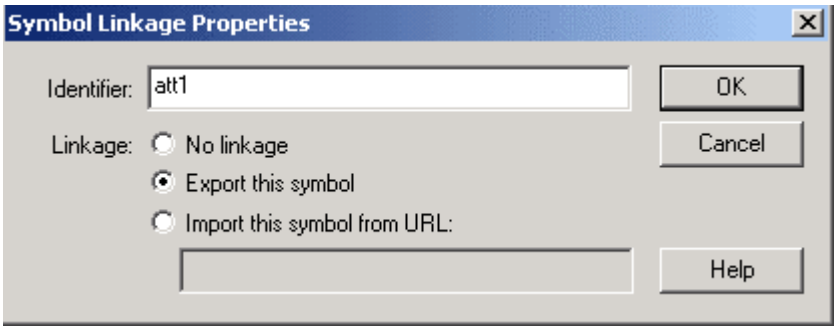
idName – это имя приаттачиваемого MC в библиотеке (Library). Если библиотека недоступна, то необходимо ее открыть последовательностью действий: Window->Library. Это имя устанавливается в поле идентификации в Symbol диалоговом окне Linkage Properties. Для его выставления необходимо выполнить следующую последовательность действий:



1) Открыть окно Library

2) Выбрать MC (в библиотеке), нажать правой клавишей мыши и выбрать **Linkage...**

3) В пункте Symbol Linkage Properties выбрать «Export this symbol» и выбрать пункт **idName**



newname – это уникальное instance name для MC, который приаттачивают (выбирается согласно вашим пристрастиям).

depth – целочисленная спецификация глубины нахождения MC (depth)

Теперь научимся выполнять метод attachMovieClip по нажатию на кнопку. Как и ранее, нарисуем кнопку. Создадим три MC, удалим их со сцены и в библиотеке по описанной выше методике зададим им параметр Linkage – «att1», «att2», «att3». На кнопках напишем:

Кнопка 1

```
on (release) {
main.attachMovie("att1", "newname1", 1);
}
```

Кнопка 2

```
on (release) {
main.attachMovie("att2", "newname2", 1);
}
```

Кнопка3

```
on (release) {
main.attachMovie("att3", "newname3", 1);
}
```

По каждой кнопке указывется глубина 1 depth(1), в следствии чего и происходит загрузка и отгрузка MC. Используйте removeMovieClip или unloadMovie акции и методы для принудительной отгрузки MC.

Задание на сам. работу: Создайте несколько MC с рандомными свойствами и повторите созданное при помощи duplicateMovieClip.

Создание своего курсора

Пример № 1

Для того чтобы создать свой собственный курсор необходимо выполнить следующую последовательность операций:

Создать свой собственный МС и разместить его на сцене,

Перейдите в режим написания скрипта на МС и в этом режиме запишите следующий скрипт:

```
onClipEvent(load){
startDrag(this, true);
}
```

После этого запустите тестовый режим и посмотрите, что у вас получилось. Как вы видите, сразу после загрузки ролика за курсором мышки передвигается нарисованный вами МС.

Теперь сделаем курсор невидимым, для этого допишем скрипт на МС:

```
onClipEvent(load){
mouse.hide()
startDrag(this, true);
}
```

И все. Курсор мышки пропадает, а вместо него по экрану движется тот курсор, который вы сами нарисовали.

К аналогичному результату приводит следующий скрипт:

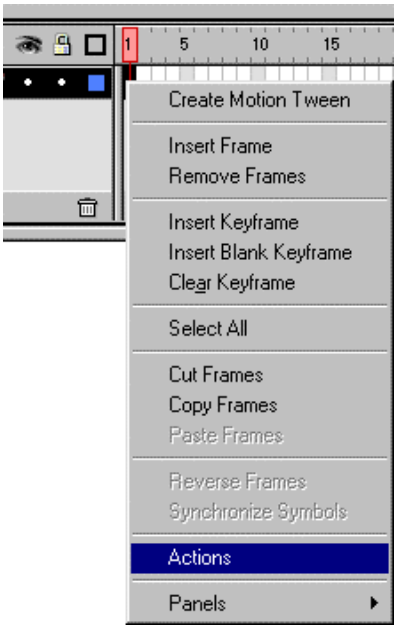
```
onClipEvent(load){
mouse.hide()
startDrag(“”, true);
}
```

Проверьте его работоспособность на своем курсоре.

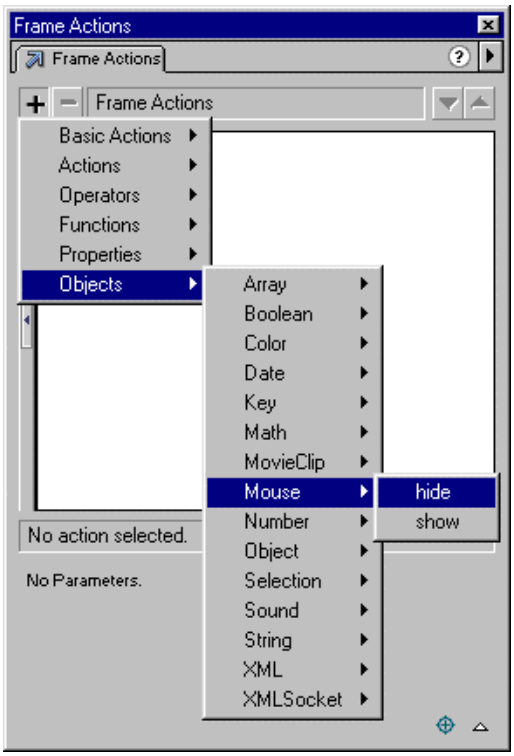
Пример № 2

Аналогичного действия можно добиться при обращении к курсору мыши не через МС, а через линейку кадров (timeline).

В Actions на сцене можно зайти, нажав на правую кнопку, а затем на подменю.

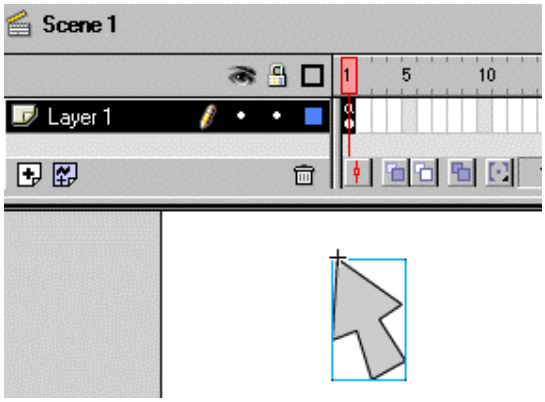


В меню Actions нажмите знак «+»для того, чтобы добавить действие. Затем выберите Objects ->Mouse -> Hide.

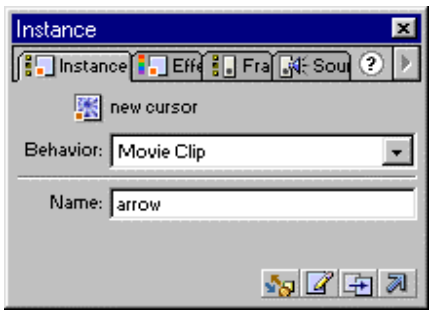


Теперь надо нарисовать сам курсор, на который будет меняться старый. Вы можете импортировать курсор из файлов с расширениями .cur, .ani, .ico. Или нарисовать свой. Нарисуем свой курсор. Создайте новый MC, нажав на Insert -> New Symbol, страницу, которая откроется, назовите «new cursor». Будьте уверены, что вы создаете «Movie Clip». Теперь будем рисовать. Вы можете нарисовать все, что хотите, но тут есть примечание, после того, как вы

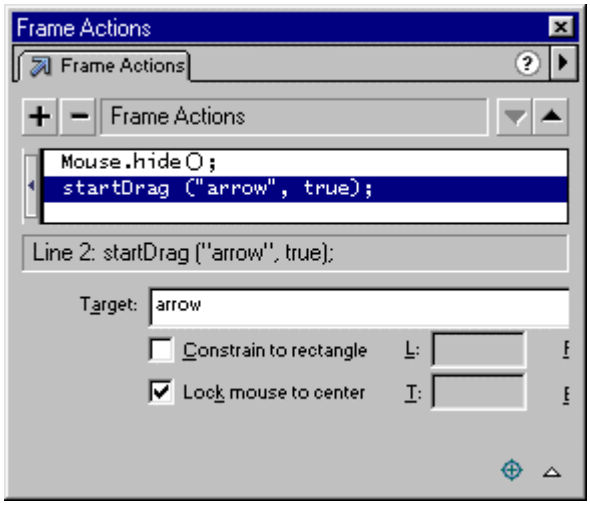
нарисовали, поместите место, которым будет нажиматься в центр крестика. В данном случае это кончик стрелочки.



После того как вы создали свой курсор, вернитесь на сцену и нажмите на меню Edit->Edit Movie, откройте свой архив (Library). Перетащите на рабочую поверхность курсор. На счет размещения не волнуйтесь, курсор можно размещать в любом месте.



После размещения курсора, нужно дать ему имя. Т.к. курсор это MC, то он должен быть назван. Для этого нажмите на меню Window -> Panels -> Instance. В новом окошке вы найдете несколько опций, но нам нужен только **Instance**. В месте, где написано **Name,** напишите любое название. В примере это название «arrow». Теперь вернитесь на главное меню и во «Frame Actions», нажмите на «+». Потом на Actions ->startDrag.



Теперь внизу, где написано «Target», напишите имя курсора, у нас это «arrow». Так же выделите «Lock mouse to center». Вот и все. Теперь курсор готов.

Перетаскивание объектов (startDrag)

Создайте символ. Выделите его. Нажмите Insert>Convert to Symbol. Введите имя, и сделайте как Movie Clip. Правый клик и Edit. Выделите и жмите Insert>Convert to Symbol. Назовите его Object Button, выберите кнопкой.

Щелкните два раза на нем. Зайдите в Actions. И напишите это:

```
on(press, dragOver){
startDrag ("");
}
on(release, releaseOutside, dragOut){
stopDrag ();
}
```

Все, вы можете перетаскивать объект по сцене.

Получение координат мыши

Вы можете использовать координаты мыши (_xmouse & _ymouse) в своей работе с МС. В каждом МС содержится своя собственная информация.

Для наблюдения свойств _xmouse и _ymouse в основной линейке кадров и в линейке кадров МС необходимо обрабатывать координату мыши и помещать ее в переменную.

Следующее выражение может быть помещено в каждую из линеек кадров для того чтобы передавать координаты мыши в основную сцену, она же _root, она же _level0:

```
x_pos = _level0._xmouse;
```

Для того чтобы определить позицию мыши внутри МС необходимо задействовать его имя, которое задается в поле **Instance Name** панели **Instance**.

Итак, нарисуйте на сцене прямоугольник, выделите его, нажмите клавишу F8 и назовите ваш прямоугольник Movie Clip-ом, после чего задайте ему Instance name.

Создайте два динамических текстовых поля на сцене, в которые вы будете выводить глобальные координаты мыши и два поля внутри МС.

Для того чтобы получить глобальные координаты мыши в скрипте на ключевом кадре можно записать:

```
xpos = _root._xmouse;
ypos = _root._ymouse;
```

Как видно из написанного скрипта, мы используем 2 переменные, в которые передаются значения текущей позиции курсора.

Для определения позиции курсора внутри MC можно поместить аналогичную запись или на самом MC записать:

```
onClipEvent(enterFrame){
    xmousePosition = _xmouse;
    ymousePosition = _ymouse;
}
```

Для обработки координат мышки можно использовать и следующее творчество:

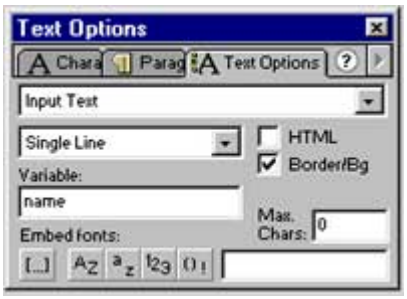
```
onClipEvent(mouseMove){
    x_pos = _root._xmouse;
    y_pos = _root._ymouse;
}
```

Теперь вы готовы к любым приключениям ☺

Создание формы

Чтобы создать текстовое поле возьмем инструмент Text Tool и, нажав левой кнопкой мыши в том месте, где надо поставить это поле, вытянем до нужных размеров. Рядом поставим название этого поля, например, пусть это будет Name:

Далее в окне Text Options выставляем параметр текста "Input Text" и задаем имя переменной (Variable), равной name. Также поставим галочку на "Border/bg".



Повторите эти шаги для текстовых полей Homepage, e-mail и comments. Для поля Comments установите параметр "Multiline" и "Word Wrap". "Multiline" позволит вводить в несколько строк, а "Word Wrap" будет гарантировать, что весь текст будет замечен.

Создайте две кнопки Send и Clear.

В кнопке Clear добавим:

```
on(release){
    name = ""
    homepage = ""
    email = ""
    comments = ""
}
```

К кнопке Send добавляем:

```
on (release){
Load Variables ("http://www.server.com/cgi-bin/myform.cgi", 1, vars=POST)
}
```

Движение маски за мышью

1) Создаем **Movie Clip (masked_object)**. В нем нарисуем круг без рамок.

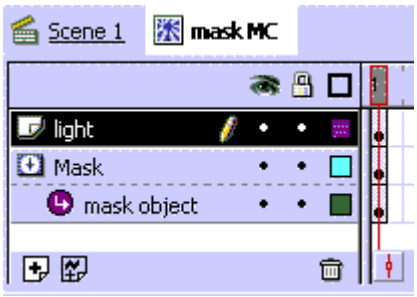
2) Далее создадим **Movie Clip (mask_MC)**

3) В нем должно находиться:

Слой **light** – кнопка, к которой применен эффект прозрачности.

Слой **Mask** - ваша маска (имеющая такую же форму и размер как кнопка)

Слой **mask object** - здесь находится тот самый Movie Clip, который мы создали в первую очередь. Имя для этого клипа в "Instance" должно быть задано - **masked_object**



4) Теперь Movie Clip помещаем на сцену? задав имя в "Instance" - **mask_MC** и вносим для него этот код:

```
onClipEvent (mouseMove) {
      x = _root._xmouse;
      y = _root._ymouse;
      this._x = x;
      this._y = y;
      this.masked_object._x = (20-x);
      this.masked_object._y = (30-y);
      updateAfterEvent();
}
```

Как это работает? Давайте посмотрим

Эти строки получают координаты мыши:

```
x = _root._xmouse;
y = _root._ymouse;
```

Эти линии перемещают маску клипа в координаты мыши.

```
this._x = x;
this._y = y;
```

Здесь задается положение клипа masked_object относительно координат x и y.

```
this.masked_object._x = (20-x);
this.masked_object._y = (30-y);
```

Password

Создайте новый мувиклип и установите в нем 2 слоя. Первый frame слоя 1 имеет:

1. "Input text" с именем переменной "login"

2. Кнопка "Send"

3. Actions для первого кадра : password = login;

Во втором слое фрейма 2 установите следующее:

```
GotoAndPlay (1);
```

Десятому фрейму этого же слоя дайте название "error"

Первый слой на 10 фрейме имеет кнопку "Retry".

Теперь добавим actions для кнопок:

```
 Кнопка "Send":
 on (release) {
 if (password eq "flashpower") {
 getURL ("http://flashpower.by.ru", "_blank");
 } else if (password eq "action") {
 getURL ("http://flashpower.by.ru", "_blank");
 } else if (password eq "goga") {
getURL ("http://flashpower.by.ru", "_blank");
 } else {
 gotoAndStop ("error");
 }
 }
Кнопка "Retry":
on (release, dragOver) {
gotoAndPlay (1);
}
```

Как это работает?

Когда мы вводим пароль, переменной "login" присваивается значение: login="то, что вы ввели в поле для пароля". Далее при нажатии на кнопки "Send" , проверяется, является ли это слово паролем. С начало оно проверяется с "flashpower". Если это тот пароль, то считывается следующая строка, в которой идет переход по заданной ссылке. Если нет, тогда проверяется с другим паролем, потом с другим пока не наступит "else"- что является переходом не фрейм "error".