МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Демиденко М.Г., Федченко О.В.

Крос-платформні мови програмування

Конспект лекцій

Суми - 2010

Зміст

ОСНОВНІ ПОНЯТТЯ ТЕОРІЇ ПРОГРАМУВАННЯ

Мова програмування — система позначень для опису алгоритмів та структур даних. Мову програмування визначає набір лексичних, синтаксичних і семантичних правил, що задають зовнішній вигляд програми і дії, які виконує виконавець (комп'ютер) під її управлінням.

Тип даних — характеристика, яку явно чи неявно присвоєно об'єкту (змінній, функції, полю запису, константі, масиву тощо). Тип даних – фундаментальне поняття теорії програмування. Тип даних визначає множину припустимих значень, формат їхнього збереження, розмір виділеної пам'яті та набір операцій, які можна робити над даними.

Цілі числа (англ. integer) не можуть містити у собі дріб. Для від'ємного числа треба ставити знак мінус (–) перед значенням (числом). Неможна використовувати кому у введені такого числа, бо інакше буде викликана синтаксична помилка.

Дійсні числа можуть містити у собі як цілі, так і дробові значення з точкою відокремлення від цілої частини. Для від'ємного числа треба ставити знак мінус (–) перед значенням (числом).

Рядки (англ. string) — нечисловий тип даних, та використовується для збереження букв та слів. Усі рядки складаються з символів. Рядки можуть містити цифри та числа, але все одно будуть оброблятися як текст.

Логічний тип. Має два значення: 1 і 0. Можуть застосовуватися в логічних операціях. Використовується в операторах розгалуження і циклах.

*Масив.*Це індексований набір елементів одного типу. Одновимірний масив – вектор, двовимірний масив – матриця.

Файловий тип. Зберігає тільки однотипні значення, доступ до яких здійснюється тільки послідовно (файл з довільним доступом, що входить у деякі системи програмування, фактично є неявним масивом).

Інші типи даних. Якщо описані вище типи даних представляли будь–які об'єкти реального світу, то інші типи даних представляють об'єкти комп'ютерного світу, тобто є виключно комп'ютерними термінами.

Покажчик. Зберігає адресу в пам'яті комп'ютера, який вказує на будь–яку інформацію, як правило – покажчик на змінну.

Посилання.

Кросплатформенність — здатність програмного забезпечення працювати більш ніж на одній платформі або операційній системі

Кросплатформенними можна назвати більшість сучасних високорівневих мов програмування. Наприклад, C, C++ і Free Pascal — кросплатформні мови на рівні компіляції, тобто для цих мов є компілятори під різні платформи. Java і C# — кросплатформні мови на рівні виконання, тобто їх виконавчі файли можна запускати на різних платформах без попередньої перекомпіляції. PHP, ActionScript, Perl, Python, Tcl і Ruby — кросплатформні інтерпретовані мови, їх інтерпретатори існують для багатьох платформ.

ActionScript — це об'єктно–орієнтована скриптова мова програмуваня, що дозволяє запрограмовувати Adobe Flash–кліпи та додатки. ActionScript, як і JavaScript,

базується на ECMAScript — стандарті скриптових мов, тому в обох мовах дуже схожий синтаксис. ActionScript компілюється в байткод, який включається до SWF–файл.

SWF–файли виконуються Flash Player–ом. Flash Player існує у вигляді плагіна до веб–браузера, а також як самостійно виконуємий додаток (standalone). У другому випадку можливе створення виконуваних exe–файлів (projector), коли swf–файл включається у Flash Player.

За допомогою ActionScript можна створювати інтерактивні мультимедіа–додатки, ігри, веб–сайти та багато іншого.

Об'єктно–орієнтоване програмування (ООП) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. В ній основними концепціями є поняття об'єктів і класів.

Парадигма програмування – це сукупність ідей і понять, що визначає стиль написання програм. Парадигма програмування визначає те, в яких термінах програміст описує логіку програми.

Фундаментальні поняття ООП

Клас – визначає абстрактні характеристики сутності (її атрибути або властивості) та дії, які вона здатна виконувати (її поведінки, методи або можливості). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять модульність та структурованість в об'єктно–орієнтовану програму.

Об'єкт – окремий екземпляр класу. Наприклад, клас Собака відповідає всім собакам шляхом опису їхніх

спільних рис; об’єкт Рекс є одним окремим собакою, окремим варіантом значень характеристик. Собака має хутро; Рекс має коричнево–чорне хутро. Об’єкт Рекс є екземпляром класу Собака. Сукупність значень атрибутів окремого об’єкту називається станом.

Метод – можливості об’єкту. Оскільки Рекс — Собака, він може гавкати. Тому гавкати() є одним із методів об’єкту Рекс. Він може мати і інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об’єкт; всі Собаки можуть гавкати, але треба щоб гавкала лише одна окрема собака, наприклад Рекс.

Успадкування

В деяких випадках, клас може мати «підкласи», спеціалізовані версії класу. Наприклад, клас Собака може мати підкласи Вівчарка та Такса. В цьому випадку, Рекс буде екземпляром підкласу Вівчарка. Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні.

Приховування інформації (інкапсуляція)

Приховування деталей про роботу класів від об’єктів, що їх використовують або надсилають їм повідомлення. Так, наприклад, клас Собака має метод гавкати(). Реалізація цього методу описує як саме повинно відбуватись гавкання (приміром, спочатку вдихнути() а потім видихнути() на обраній частоті та гучності). Хазяїн пса Рекса, не повинен знати як він гавкає. Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об’єкту. Як наслідок, кожен об’єкт представляє кожному іншому класу певний інтерфейс –

члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись. Це дозволить полегшити внесення змін, тобто, без потреби змінювати і користувачів інтерфейсу. Наприклад, інтерфейс може гарантувати, що щенята можуть додаватись лише до об'єктів класу Собака кодом самого класу. Часто, члени класу позначаються як *публічні (англ. public)*, *захищені (англ. protected)* та *приватні (англ. private)*, визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено.

Абстрагування

Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. Наприклад Собака Рекс більшу частину часу може розглядатись як Собака, а коли потрібно отримати доступ до інформації специфічної для собак породи вівчарка – як Вівчарка і як Тварина (можливо, батьківський клас Собака) при підрахунку тварин Хазяїна.

Поліморфізм

Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по різному на однакові повідомлення. Наприклад, якщо Собака отримує команду голос(), то у відповідь можна отримати «Гав»; якщо Свиня отримує команду голос (), то у відповідь можна отримати «Хрю».

1. Знайомство з інтерфейсом Flash

1.1 Технологія Flash

Flash – це система для створення векторної анімації, що має достатньо розвинені можливості програмування. Оскільки формат Flash став стандартом і підтримується всіма поширеними браузерами (в першу чергу, завдяки його компактності і відмінній підтримуваності), Flash в даний час використовується не тільки для анімації, але і для інтерактивних додатків і навіть у якості інтерфейсу користувача для складних веб-орієнтованих систем.

Средовище розробки Flash

Для початку, ознайомимося з інтерфейсом середовища розробки Flash. Спершу розберемо панель інструментів. Вона розташована за замовчуванням ліворуч, на ній знаходяться основні інструменти для роботи з графікою. При виборі одного з них під панеллю інструментів з'являється панель додаткових опцій для кожного інструменту. А тепер детальніше про кожен з них (елементи перераховано зліва направо, зверху вниз):

1 – виділення (виділення об'єктів);
2 – субвиділення (переміщення об'єктів);
3 – пряма (дозволяє намалювати пряму лінію);
4 – ласо (захоплення частини зображення);
5 – ручка (дозволяє намалювати лінію по точках);
6 – текст (створення тексту);
7 – коло (дозволяє намалювати коло або овал);
8 – квадрат (дозволяєнамалювати прямокутник);

9 – олівець (малювання від руки, олівцем);

10 – пензель (малювання пензлем);

11 – трансформація (обертання, зміна розмірів);

12 – зміна заливки (дозволяє управляти заливкою);

13 – чорнило (надає додаткову товщину);

14 – заливка (дозволяє заливати об’єкти кольором);

15 – піпетка (дозволяє вибрати будь-який колір з меж робочої області Flash);

16 – стиральна гумка (дозволяє стирати елементи на робочій області).

Перейдемо до практичного застосування даних інструментів управління. Для прикладу створимо коло. В результаті на робочій області з’являється коло (рис. 1.1).

Рис. 1.1 – Коло, створене на робочій області.

Якщо виділити даний об’єкт, то з’являється вікно властивостей (рис. 1.2).

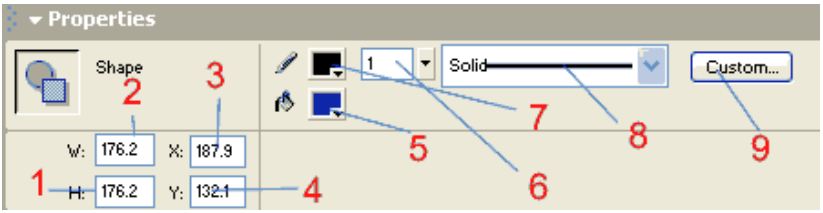


Рис. 1.2 – Вікно властивостей: 1 – висота об’єкту; 2 – ширина об’єкту; 3 – положення по осі X; 4 – положення по осі Y; 5 – заливка; 6 – товщина ліній обводки; 7 – колір ліній обводки; 8 – вигляд ліній обводки; 9 – настройка ліній обводки.

Якщо об’єкт коло виділити, то основними елементами управління будуть наступні (рис. 1.3):

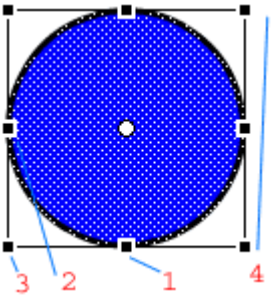


Рис. 1.3 – Елементи управління об’єктом: 1 – зміна розмірів висоти; 2 - зміна розмірів ширини; 3 – одночасна зміна розмірів висоти і ширини; 4 – обертання

Розглянемо панель управління кольором (рис. 1.4). За допомогою даної панелі можна як змінювати колір, так і здійснювати градієнтну заливку.

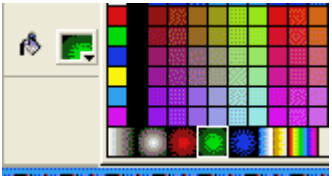


Рис. 1.4 – Панель управління кольором.

У результаті вибору градієнтної заливки можна отримати зображення (рис. 1.5).

Рис. 1.5 – Об’єкт «коло» після градієнтної заливки.

Далі розберемо створення анімації на прикладі створеного об’єкту «коло».

1.2 Кліпи і лінійка часу.

Флеш–роликом називається готовий відкомпільований swf–файл (який можна запустити в браузері або зовнішньому флеш–плеєрі).

Все, що намальовано в готовому флеш–ролику – так чи інакше намальовано всередині кліпів. Кліп є мінімальною порцією/частиною "намальованого матеріалу", з якою можна поводитися як з цілим: рухати, повертати, змінювати прозорість і порядок накладення.

Кліпи можуть бути досить великими, зокрема, містити в собі інші кліпи, а ті, у свою чергу – ще якісь кліпи і т.д.

Кожен кліп містить у собі так звану "лінійку часу" (Timeline). У процесі редагування кліпів цю лінійку можна побачити – вона розташована над вікном редагування і являє собою "зебру" з маленьких прямокутників, витягнутих у ланцюжок зліва направо (рис. 1.6).

Кожен прямокутник символізує собою окремий кадр. Після компіляції і запуску лінійки часу не видно, однак поняття "лінійки часу" залишається корисним: можна говорити про те, який по порядку кадр є поточним в даному кліпі. Як правило, лінійки усіх кліпів (у тому числі й вкладених один в одного) незалежні, хоча зміна поточних кадрів у всіх кліпах здійснюється синхронно. Незалежність проявляється в тому, що при зміні кадру кліп не обов’язково переходить на наступний кадр; він може повернутися на перший кадр, зупинитися або взагалі опинитися на будь–якому кадрі, який йому задали. Таке управління здійснюється програмно.

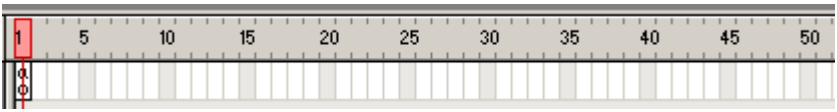


Рис. 1.6 – Зовнішній вигляд лінійки часу

Важливим моментом є те, що кожен з незалежних кліпів переходить на той кадр, який йому задали (якщо не задали нічого – переходить на наступний), і ці кадри в різних кліпах не пов’язані між собою.

Лівіше від лінійки часу знаходиться вікно опцій кадрів (рис. 1.7).

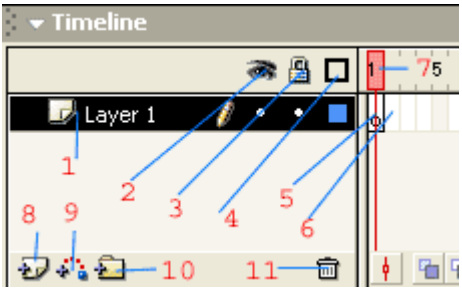


Рис. 1.7 – Зовнійшій вигляд вікна опцій: 1 – шар (їх може бути необмежена кількість); 2 – сховати об’єкти поточного шару; 3 – заборонити редактувати поточний шар; 4 – відображати лише контури шару; 5 – активний кадр; 6 – наступний кадр; 7 – покажчик поточного шару; 8 – створити новий шар; 9 – створити направляючий шар; 10 – створити директорію для збереження шарів; 11 – видалити шар.

Тепер докладніше про те, що таке шар. Можна легко зрозуміти, якщо припустити, що у вас є прозоре скло, під ним лежать листки паперу, на яких щось намальовано. Шари необхідні для того, щоб розділяти як відособлені

об’єкти, так й анімовані композиції, що складаються з декількох об’єктів. Розмір файлу не змінюється від кількості шарів. Під Flash є два способи малювання анімації. Перший – коли малюються вся кадри користувачем. У другому способі користувач вказуєте перший і кінцевий кадр, а програма автоматично домальовує відсутні проміжні кадри.

Розберемо спосіб автоматичного створення кадрів. Наприклад створимо об’єкт (рис. 1.8).

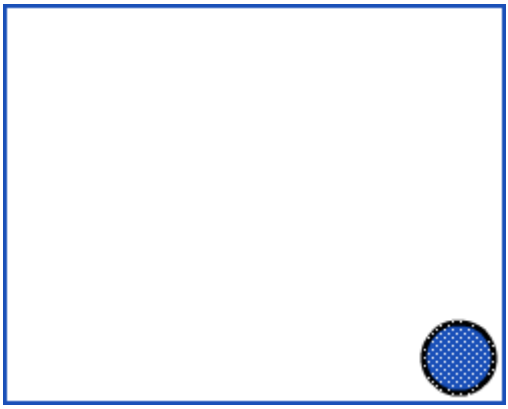


Рис. – 1.8 Об’єкт коло.

Далі перетворимо створений об’єкт в ролик. Для цього необхідно вибрати меню Convert to Symbol або натиснути F8 (рис. 1.9).

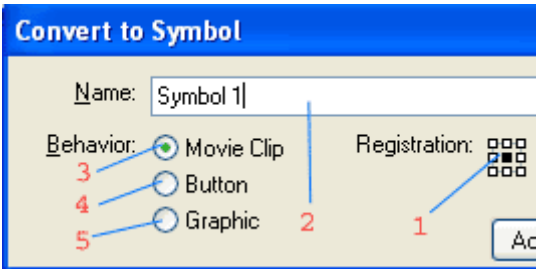
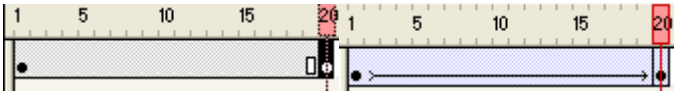


Рис. 1.9 – Панель перетворення об’єктів: 1 – вирівнювання; 2 - назва символу (може бути будь-яка); 3 - анімаційний кліп (варто перетворювати, якщо об’єкт динамічний); 4 - кнопка (має чотири фіксованих положення); 5 - графіка (варто перетворювати, якщо об’єкт статичний).

Вибираємо movie clip, ставимо галочку і тиснемо ОК. На лінії часу вставляємо ключовий кадр (F6) на позначці 20 (рис. 1.10а).



а б

Рис. 1.10 – Лінійка часу.

У кадрі номер двадцять коло з правого нижнього кута переміщаємо в лівий верхній кут (рис. 1.11).

Рис. 1.11 – Зображення останнього кадра ролика.

Потім, виділяємо перший кадр і натискаємо правою кнопкою, вибираємо create motion tween або виділяємо перший кадр і в панелі властивостей вибираємо Tween - motion. Лінійка часу тут же забарвлюється в фіолетовий колір (рис. 1.10б). Тепер тиснемо Ctr + Enter і Flash автоматично створює кадри 2-19.

1.3 Символи та екземпляри (instances) кліпів

Символи та екземпляри – це дуже важлива концепція. Говорячи про кліп, можна говорити про його символи (аналог класу) і про примірник (аналог об’єкта класу). Тобто кожному унікальному кліпу відповідає свій символ; але можна зробити безліч екземплярів кліпу, породжених одним і тим же символом. Список всіх символів кліпу можна подивитися в панелі "Library" (рис. 1.12), яка з’являється після натискання Ctrl + L. Бібліотека-це місце де зберігаються всі об’єкти. Можна імпортувати в неї також растрові зображення та звуки. Достатньо один раз створити будь-який об’єкт або помістити його в

бібліотеку, щоб використовувати його потім, причому міняючи його розмір місце положення колір.

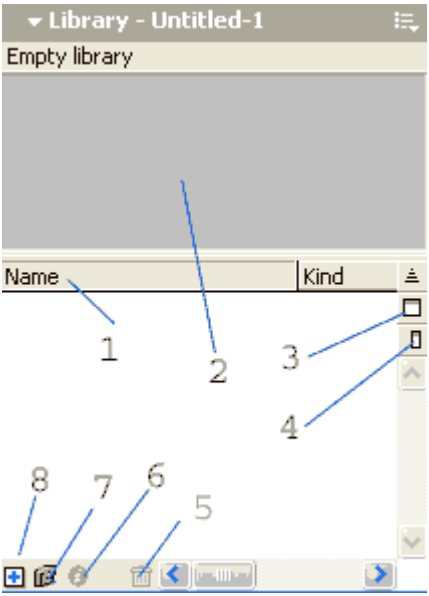


Рис. 1.12 – Зовнішній вигляд вікна панелі Library: 1 - назва об’єкта; 2 - область перегляду об’єкту; 3 - розгорнути вікно; 4 - згорнути вікно; 5 - видалити об’єкт; 6 - властивості об’єкта; 7 - створити папку; 8 - новий символ

1.4 Інтерфейс користувача середовища

Інтерфейс середовища Flash складається з набору панелей. Якщо зайтидо розділу головного меню – Window, то можна побачити повний перелік панелей, кожну з яких можна прибирати і знову встановлювати.

1.5 Сценарій ролика

Для сценарія ролика використовуємо мову програмування ActionScript. Але перш за все необхідно

задати програму дій, яка допоможе втілити в життя запропонований сценарій ролика.

Подивитися структуру флеш–ролика можна за допомогою панелі Movie Explorer.

1.6 Ключові і звичайні кадри

Слід добре уявляти, як зробити звичайний кадр, а як – ключовий. Якщо виділити будь–який порожній кадр і натиснути F5 (або вибирати Insert Frame з контекстного меню або меню Insert), то лінійка часу від найближчого заповненого кадру і до виділеного зафарбується сірим. Це означає, що анімація продовжена до вибраного місця. При цьому і малюнок, і код, і анімаційні ефекти відповідають тим, що обрані в найближчому ключовому кадрі сторінки.

1.7 Додавання програмного коду

Для додавання програмного коду необхідно перейти на панель Actions, натиснувши F9(рис. 1.13).

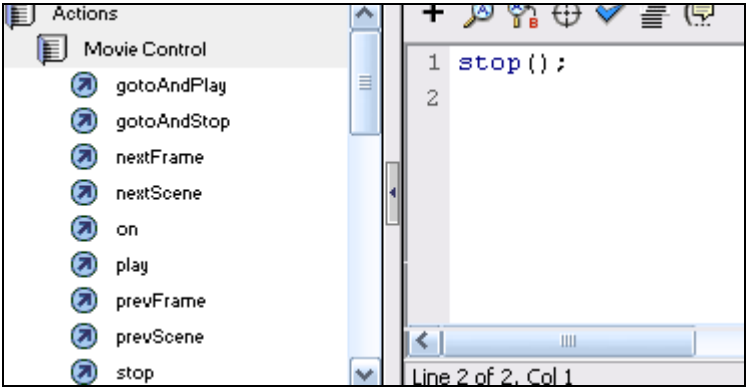


Рис. 1.13 – Зовнішній вигляд панелі Action.

2. Середовище розробки Flex

2.1 Основні теоретичні поняття

Adobe Flex– технологія для створення Rich Internet Applications. Flex – це заснована на Flash технологія, призначена прискорити і спростити розробку насичених web-додатків. Flex розширює базові можливості flash, дозволяючи описувати інтерфеси програми мовою XML. Логіка програми пишеться на ActionScript. Результатом компіляції є SWF файл призначений для запуску та виконання в інтернет-браузері (на платформі Adobe Flash Player), або як самостійний додаток (на платформі AIR).

Flex – це великий набір класів розширюють можливості Flash. Flex-framework включає можливості локалізації, стилізації програми, розробки модульного програми, вбудовані валідатори і Форматори текстових полів. Всі ті інструменти, які потрібні розробникам додатків, що працюють online. Але все готові рішення, включені у Flex-framework, компілюються в результуючий SWF файл, збільшуючи його розмір. У випадку з Flash SWF результуючий файл містить лише ті класи, які написав програміст.

Rich Internet application (RIA, «багатий Інтернет-додаток») – це додаток, доступне через Інтернет, багатий функціональністю традиційних настільних додатків, що не підтримуються безпосередньо браузерами. Як правило, додаток RIA:

- передає веб-клієнту необхідну частину інтерфейсу користувача, залишаючи велику частину даних (ресурси програми, дані та ін.) на сервері;

- запускається в браузері і не вимагає додаткової установки ПЗ;
- запускається локально в середовищі безпеки (sandbox).

ActionScript– це мова програмування, що дозволяє створювати Flash додатки. Команди ActionScript відповідають чіткій схемі. Для написання програм використовується середовище програмування Flex. Додатки Flex пишуться на мові MXML, що описує компоновку і зв'язок між компонентами, а ActionScript служить для описання логіки програми.

Flex SDK (Flex Software Development Kit (SDK)) – набір засобів розробки програмного забезпечення. Flex SDK складається з компілятора, засобів створення документації, великої бібліотеки компонентів інтерфейсу користувача та утиліт, що спрощують процес розробки. Flex SDK може бути використане зі звичайним текстовим редактором або з Flex Builder, потужним середовищем розробки.

2.2 Створення нового проекту Flex

Щоб створити новий проект, необхідно вибрати меню File і далі NewFlex Project.При цьому відкриється діалогове вікно (рис. 2.1), в якому потрібно задати налаштування проекту. Назва проекту вказується у полі Projectname. У даному випадку це буде HelloWorld.

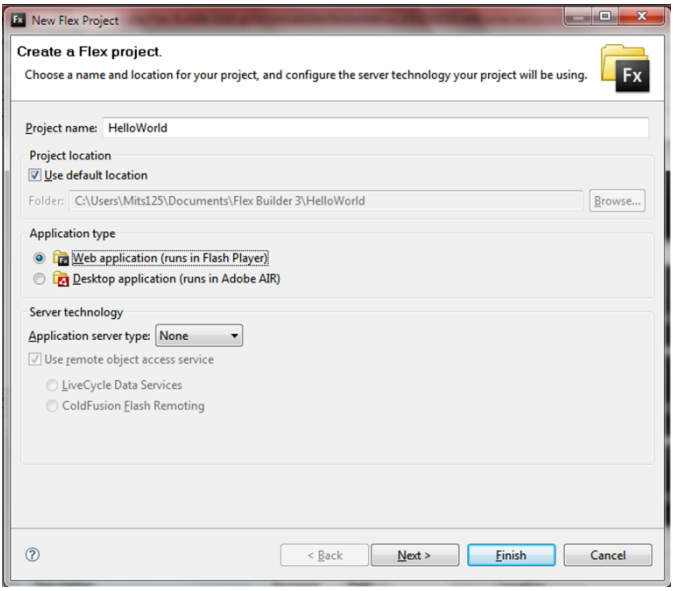


Рис. 2.1 – Зовнішній вигляд діалогового вікна створення проекту.

У діалоговому вікні можна бачити три розділи: Project location, Application type і Server technology. Project location змінює розташування файлів і папок проекту на комп'ютері. За замовчуванням вказано шлях до папки робочого простору користувача, але можна вказати будь-який шлях, знявши прапорець навпроти Use default location.

У назві проекту не повинно бути пробілів та інших незвичайних символів - ім'я може складатися тільки з букв, цифр, знаку $ і символу підкреслення.

У другому розділі Application type необхідно зробити вибір між розробкою веб-додатку або настільного додатку, що запускається за допомогою Adobe AIR.

У третьому розділі Server Technologies потрібно вказати серверні технології, які треба використовувати. Цю опцію використовують в тому випадку, якщо

планується підключати до додатку базу даних, файл XML або інші дані, що знаходяться на віддаленому сервері.

Установки проекту можна змінити в будь-який час, вибравши пункт Project → Properties в меню Flex Builder.

2.3 Структура Flex-проекту

Розглянемо структуру каталогу навігатора проекту. Традиційно в кожен Flex-проект входять папки з назвами bin-debug, html-template, libs і src (рис. 2.2). В табл. 2.1 докладно описано, для чого призначена кожна з папок, а на рис. 2.8 представлена їхня структура, яка відображається на панелі Flex Navigator.

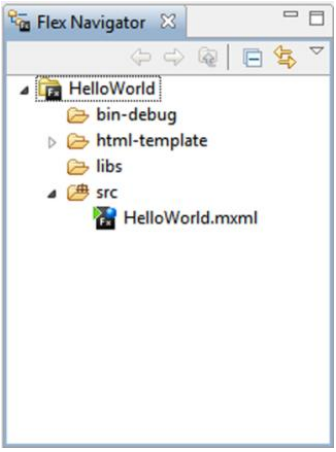


Рис. 2.2 – Зовнішній вигляд вікна навігатора проекту

Таблиця 2.1 – Структура проекту

Назва папки	**Призначення**	**Опис**
bin-debug	Скомпільований код	Містить скомпільований код (файл SWF). При розробці для веб містить також HTML-контейнер і допоміжні файли. При розробці AIR-додатків – файл опису програми (Application Descriptor File).
html-template	Шаблон html сторінки	Містить HTML-шаблон, на основі якого генерується файл HTML-контейнера (тільки для веб-додатків).
libs	Бібліотеки	Містить файли додаткових бібліотек.
srs	Вихідний код	Містить вихідний код у вигляді файлів з розширенням .mxtml або .as, а також файл опису додатку (тільки для AIR-додатків).

3. Робота в режимі Design

Режим Design дозволяє редагувати додаток за принципом WYSIWYG (від англ. what-you-see-is-what-you-get – режим точного відображення). У розпорядженні розробника є великий набір візуальних компонентів, які перетягуються мишею в потрібне місце при побудові інтерфейсу програми. Всі властивості цих компонентів можна змінювати у візуальному режимі

3.1 Основні компоненти Flex

Компонент – це фрагмент коду, що багаторазово використовується як інструмент для створення додатку. Компоненти стають елементами готової системи. Однією з переваг Flex є багатий набір компонентів. Але незважаючи на те, що використання компонентів економить масу часу і не вимагає багато спеціальних знань, на більш глибоке освоєння техніки їх застосування та конфігурації буде потрібно якийсь час, який не буде витрачено даремно.

Нижче приведено список найбільш поширених компонентів.

3.2 Елементи управління

Елементи керування – основні візуальні компоненти інтерфейсу користувача, такі як кнопки або текст. Назва пояснюється тим, що вони призначені для керування додатками при роботі користувача.

- *Button*

Елемент виглядає і функціонує як справжня кнопка. Її можна використовувати і як перемикач завдяки властивості toggle, а її стан регулюється властивістю selected.

 *CheckBox*

Виконує функцію перемикача.

 *ComboBox*

Компонент, що складається зі списку і кнопки, представляє собою компактний випадаючий або спливаючий список з декількох пунктів. Властивості selectedItem (об'єкт) або selectedIndex (ціле число) дозволяють отримати або встановити вибраний об'єкт (або його індекс).

 *Image*

Цей елемент дозволяє підключати зовнішні ресурси. Можливі формати - GIF, JPEG, PNG, а також SWF-файли. Посилання на зображення повинне бути вказане у властивості source.

 *Label*

Елемент для створення простої однорядкового позначки.

 *List*

Цей елемент відображає список, що складається з декількох пунктів. Якщо для виведення усіх пунктів не вистачає місця, з'являються смуги прокручування.

 *ProgressBar*

Цей елемент добре підходить для відображення ходу таких процесів, як завантаження файлу.

 *RadioButton*

Цей елемент використовується в групі перемикачів, причому установка одного з них скидає вибір будь-якого іншого.

 *Text*

Цей елемент використовується для розміщення фрагмента тексту без смуг прокручування. Розмір елемента буде змінений, щоб вмістити весь текст.

 *TextInput*

Призначається для введення рядка тексту. Властивість text дає можливість отримати або задати значення рядка.

3.3 Контейнери розміщення

Контейнери розміщення, або просто контейнери, - візуальні компоненти, що контролюють вирівнювання елементів програми або їх взаємне розташування. Іноді вони відображаються візуально (наприклад, рядок заголовка компонента Panel), але частіше за все вони лиш змінюють розміщення елементів (або навіть інших контейнерів) певним чином, залишаючись при цьому невидимими.

 *Canvas*

Контейнер без певної схеми вирівнювання. Для розміщення елемента в конкретному місці необхідно задати значення його координат по осях x та y.

 *Form*

Цей контейнер дозволяє створити щось на зразок HTML-форми. При використанні разом з компонентами FormItem, розміщує їх по вертикалі, при цьому поля форми вирівнюються по лівому краю.

 *HBox*

Вирівнює елементи по горизонталі.

 *Panel*

Цей компонент нагадує вікно з рядком заголовка. При додаванні Panel в режимі Design за замовчуванням розміщення внутрішніх елементів буде «абсолютним», тобто заданими координатами по осі x і y (як і при використанні Canvas). Але вирівнювання можна також змінити на горизонтальне або вертикальне, що означає можливість повного управління розташуванням дочірніх елементів.

 *VBox*

Вирівнює елементи по вертикалі.

3.4 Навігатори

Навігатори – комбінований тип візуальних компонентів, що представляє собою щось на зразок поєднання контейнера розміщення та елементу управління. Вони призначені для створення набору контейнерів, з якого видно в кожен момент тільки один. Завдяки наявності таких компонентів значно спрощується процес створення модульних інтерфейсів.

 *Accordion*

Схожий на описаний нижче Tab Navigator, за тим винятком, що він має в своєму розпорядженні контейнери вертикально і додає анімаційний ефект при виборі одного з розділів. Зазвичай використовується спільно з групою контейнерів Form для розділення великої форми, що вимагає прокручування, на кілька секцій.

 *Tab Navigator*

Використовується з набором таких контейнерів, як HBox або Panel, забезпечуючи одночасну видимість тільки одного з них. При цьому створюються закладки, схожі на папки з файлами, з мітками, що відповідають значенням властивості label вкладених контейнерів. Видимим стає тільки контейнер, який відповідає вибраній користувачем закладці, решта контейнери при цьому прихована.

3.5 Загальні властивості компонентів

Основою всіх візуальних компонентів (редагованих в режимі Design) є базовий компонент UIComponent. Він має властивості, які регулюють видимість, розміри, реакцію при взаємодії з користувачем за допомогою миші та клавіатури, фокусування та інші параметри. Кнопки, поля для введення тексту і контейнери є розширеними варіаціями цього основного компонента, тобто до успадкованої від нього функціональності додаються характерні тільки для даного конкретного компонента функції. Тому вони мають також всі властивості основного компонента.

Нижче представлений список найбільш часто використовуваних властивостей основних компонентів з прикладами використання.

id

id (скорочено від identifier (англ. ідентифікатор) – дуже важлива властивість, що привласнює ім'я певного компонента. Якщо у додатку є дві кнопки, то назвавши їх button1 і button2, можно легко розрізняти їх у коді. Ще краще дати елементам описові імена, пов'язані з виконуваною ними функцією в додатку (наприклад, submitButton (кнопка відправки) і refreshButton (кнопка оновлення). При необхідності Flex визначить значення даної властивості автоматично, але набагато ефективніше зробити це самостійно.

```
<mx:Button id="submitButton"/>
```

x

Числова властивість, що визначає зміщення елемента вправо по відношенню до батьківського елементу. Тобто при установці значення даної властивості 20 пікселів для елемента, розташованого всередині контейнера, буде зроблено відповідний відступ від лівогокраю цього контейнера.

```
<mx:Button x="20"/>
```

y

Числова властивість, дія якої аналогічно властивості x з тією лише відмінністю, що воно регулює розташування елемента по вертикалі (відступ відраховується від верхнього краю батьківського елементу).

```
<mx:Button y="10"/>
```

visible

Властивість, що керує видимістю елемента на сцені. При присвоєнні властивості значення false елемент стає невидимим, однак займатиме деяку область простору. Приміром, якщо в контейнері Hbox розташовані чотири кнопки, вирівняні по горизонталі, і зробити другу з них невидимою, сусідні кнопки залишаться на своїх місцях, а на місці невидимою кпопки залишиться порожній простір.

```
<mx:HBox>
<mx:Button label="One"/>
<mx:Button label="Two" visible="false"/>
<mx:Button label="Three"/>
<mx:Button label="Four"/>
</mx:HBox>
```

includeInLayout

Якщо встановити значення цієї властивості false, то контейнер не буде відображати елемент, причому при цьому не виникатиме порожнього простору. Цю властивість зручно використовувати, якщо необхідно зробити який-небудь елемент невидимим, при цьому не змінюючи розташування елементів.

```
<mx:HBox>
<mx:Button label="One"/>
<mx:Button label="Two" visible="false"
includeInLayout="false"/>
<mx:Button label="Three"/>
<mx:Button label="Four"/>
</mx:HBox>
```

toolTip

Завдяки цій властивості при наведенні курсору миші на елемент на екрані з’являється невелика спливаюча підказка, що вказує на його призначення.

```
<mx:Button toolTip="Click Me"/>
```

label

Цією властивістю володіє більшість компонентів. Для кнопки – це відображений на ній текст. При використанні з контейнерами має сенс включити їх в навігатор. Тобто при завданні даної властивості для компонента всередині TabNavigator воно буде відображатися в назві відповідної йому закладки.

```
<mx:CheckBox label="I have read the terms and
conditions"/>
<mx:Button label="Submit"/>
```

text

Використовується для зміни відображуваного тексту в таких компонентах, як TextInput, TextArea і Label.

```
<mx:Label text="Enter Your Name:"/>
<mx:TextInput text="Alaric"/>
```

enabled

Значення false даної властивості забезпечує блокування компонента, що, як правило, виявляється в його відображення в сірих тонах і відсутності будь-якої реакції при наведенні на нього курсору миші і т. п. Особливо ефективно використання даної властивості для кнопки, яка повинна залишатися неактивнію до певного моменту, але його також можна використовувати з контейнерами та їх дочірніми елементами. Можна встановити значення false для всього компонента Panel, при цьому всі його елементи також будуть заблоковані.

```
<mx:Panel title="Enabled Panel" enabled="true">
<mx:ColorPicker/>
<mx:NumericStepper/>
<mx:Button label="Button"/>
<mx:CheckBox label="Checkbox"/>
<mx:DateField/>
```

```
   <mx:ComboBox/>
   </mx:Panel>
   <mx:Panel title="Disabled Panel"
enabled="false">
   <mx:ColorPicker/>
   <mx:NumericStepper/>
   <mx:Button label="Button"/>
   <mx:CheckBox label="Checkbox"/>
   <mx:DateField/>
   <mx:ComboBox/>
   </mx:Panel>
```

`source`

Дозволяє вказати посилання на зовнішнє джерело файлу в таких елементах, як Image і подібних. Спільне використання елементів Image і ProgressBar дозволяє відобразити хід процесу завантаження зображення. Для Image властивість source визначає шлях до джерела зображення, для ProgressBar – вказує на об'єкт, завантаження якого необхідно відстежувати.

```
   <mx:ProgressBar source="{photo}"/>
   <mx:Image id = "photo" source = "http:// ..." />
```

4 Будова Flex-додатків

Відкривши вихідний код простого додатка, названого HelloWorld, можна побачити наступне:

```
   <?xml version = "1.0" encoding = "utf-8"?>
   <mx:Application xmlns: mx =
"http://www.adobe.com/2006/mxml"
   layout = "absolute">
   <mx:Panel x="10" y="10" layout="absolute"
title="Howdy Ya'll">
   <mx:Label text="My name is:" fontWeight="bold"
x="10" y="14"/>
   <mx:TextInput x="5" y="41"/>
   <mx:CheckBox label="I'm a Flex Expert" x="10"
y="71"/>
```

```
</mx:Panel>
</mx:Application>
```

Розглянемо даний код по частинах. У першому рядку знаходиться оголошення XML; це необов'язковий елемент, характерний для мови XML. Власне «Flex» починається з наступного рядка, а саме з розташованого на ній тега Application.

Будь-який Flex-додаток починається з кореневого тега. Якщо це веб-додаток, він буде називатися Application, якщо це AIR-додаток, це буде WindowedApplication, що відрізняється від попереднього лише декількома додатковими параметрами, характерними для настільних додатків. Так як додаток буде запускатися через броузер, кореневим буде тег <mx:Application>. Неохідно звернути увагу на його атрибут layout="absolute". Це значення визначається налаштуваннями Flex за замовчуванням і означає, що будь-який елемент, щорозміщується всередині, буде розташований в залежності від значень координат x та y. Так як Application є контейнером, для внутрішніх елементів необхідно вказати значення координат або встановити розташування по горизонталі або вертикалі.

Всередині Application розташований тег Panel, який також є контейнером, і за замовчуванням елементи, що розташувані в ньому, також є абсолютним (layout="absolute"). Неохідно зауважити, що завдяки наявності атрибутів x="10" і y="10" Flex має в своєму розпорядженні панель з відступом в 10 пікселів від лівого кута Application. Для елементів Label, TextInput і CheckBox також задані значення координат, але при цьому відлік ведеться відносно краю батьківського елементу – контейнера Panel.

Природно, у цих тегів можуть бути й інші атрибути. Подивіться на Panel: її заголовок визначений значенням

атрибута title. Значення елемента Label визначає відображений елементом текст, а властивість fontWeight забезпечує жирне написання шрифту. Що стосується елемента CheckBox, властивість label визначає його позначку.

5 Основи MXML

Вихідний код програми заснований на використанні мови розмітки MXML. Він досить простий з точки зору написання і читання коду. MXML є різновидом XML, тому доцільно розглянути основні особливості останнього.

5.1 XML у MXML

XML – це структурований текст, будь-який текст з кутовими дужками (<i>). Дані представлені структуровано, за допомогою тегів, утворених кутовими дужками. У XML відсутній власний словник тегів, тому що автор коду створює свої власні теги в кожному конкретному випадку. XML - це лише синтаксис і структура.

Зараз ви читаєте книгу, яка складається з різних частин - глав і розділів. Щоб уявити структуру книги засобами XML, можна створити свої власні теги, такі як <book>, <chapter> і <section>, а потім з їх допомогою додати інформацію в документ.

```
<book>
<chapter>
<section/>
</chapter>
</book>
```

XML розшифровується як «розширена мова розмітки» (англ. Extensible Markup Language). «Розширена», тому що

дозволяє створювати власні теги, «розмітки» – завдяки можливості подання з його допомогою не тільки тексту, а й додаткової інформації про нього, такий як форматування тексту.

Для Flex-розробника ще важливіше, що XML призначений для опису ієрархії та структури будь-якого об’єкта. Можна сказати, що MXML – це набір створених спеціально для розробки Flex-додатків тегів XML.

5.2 Основоположні принципи XML

Для ефективної роботи у Flex, необхідно дотримуватись наступних правил:

Все, що було відкрито, необхідно закрити.

Дуже важливе правило використання XML полягає в необхідності закриття всіх тегів, тобто, якщо тег був відкритий, то в якомусь місці він повинен бути закритий.

Визначення тега складається з відкриваючої дужки (<), імені тега і закриваючої дужки (>), наприклад, <book>. Такий тег вважається відкритим. Щоб показати, що тег завершений і не містить більш вкладених елементів, використовується ліва коса риска (/).

Теги можна закривати двома способами. По-перше, можна використовувати закривання тегу. Наприклад, якщо є відкриваючий тег <book>, йому повинен відповідати закриваючий тег </book>. По-друге, є ще більш швидкий спосіб, але його можна застосовувати лише за умови відсутності вкладених тегів. Для цього необхідно додатикосу риску прямо перед правою дужкою тега, наприклад: <book/>. Таким чином, вираз <book></ book> еквівалентно <book/>.

Регістр має значення.

XML чутливий до регістру символів, тобто прописні і рядкові символи не рівнозначні, і відповідно теги <book> і <Book> не тотожні. Причому <mx:Text> і <mx:text> також вважаються різними тегами.

Оголошення не обов'язкові, але бажані.

Перший рядок документа, написаного на XML, може (хоча й не обов'язково) містити оголошення про те, якою мовою він написаний і яке в ньому використовується кодування. Виглядає приблизно так:

```
<?xml version="1.0" encoding = "utf-8"?>
```

Якщо використовувати Flex Builder, цей рядок генерується автоматично, так що про це можна не турбуватися.

MXML є версією XML, а отже, вона успадковує всі ці правила.

5.3 Тег

Тег може представляти інформацію в якості атрибута або вмісту. Вміст - це текст, розташований між двома тегами, а атрибут розміщується в відкриваючому тезі, при цьому інформація, яка надається (значення атрибуту) розміщується в лапках. Подивіться на наступний код:

```
<book title="Learning Flex" author="Alaric Cole">
<chapter title="Getting Up to Speed"/>
<chapter title="Setting Up Your Environment"/>
</book>
```

У даному прикладі <book> є кореневим тегом, а інформація про назву і автора представлена за допомогою його атрибутів title і author.

Вкладені теги представляють два розділи. Розглянемо код, який містить ту ж саму інформацію, але надану іншим способом:

```
<book>
<title> Learning Flex </title>
<author> Alaric Cole </author>
<chapter>
<title> Getting Up to Speed </title>
</chapter>
<chapter>
<title> Setting Up your Environment </title>
</chapter>
</book>
```

По суті цей код схожий на попередній, але більш громіздкий. У першому випадку використовуються атрибути, у другому - вкладені теги. Інформація, представлена за допомогоюатрибутів, виглядає набагато компактніші, а це важливо для сприйняття. Порівняємо попередні приклади з наступним кодом на MXML:

```
<mx:Label text="Learning Flex"/>
```

Але те ж саме можна записати і так:

```
<mx:Label>
<mx:text> Learning Flex </mx:text>
</mx:Label>
```

У першому випадку використовується атрибут для додавання властивості text, у другому з цією ж метою використовуються вкладені теги. Доцільно використовувати атрибути для компактності і зручності сприйняття. Проте в певних випадках буде

кращескористатися другим способом представлення інформації. Вкладені теги дозволяють розміщувати дані з більш складною організацією, ніж прості текстові рядки, тому їх варто використовувати, якщо вміст не можна представити як атрибут.

Це ще важливіше у випадку з даними, структурованими певним чином. Наприклад, для елемента списку потрібно надати дані, що представляють собою послідовність елементів, а не окремий текстовий рядок. Це справедливо і для інших властивостей, значеннями яких може бути набір даних, наприклад, для властивості columns елемента DataGrid, при цьому можливо використовувати властивості кожної окремої колонки. Щоб переконатися в цьому на практиці, можна перетягнути цей елемент на робочу область в режимі Design. Згенерований код буде виглядати приблизно таким чином:

```
  <mx:DataGrid>
  <mx:columns>
  <mx:DataGridColumn headerText="Column 1"
dataField="col1"/>
  <mx:DataGridColumn headerText="Column 2"
dataField="col2"/>
  <mx:DataGridColumn headerText="Column 3"
dataField="col3"/>
  </mx:columns>
  </mx:DataGrid>
```

Властивість columns елемента DataGrid не може бути використана в якості атрибута, тобто його використання має на увазі кілька вкладених тегів DataGridColumn, для яких у свою чергу визначені свої властивості.

6 ActionScript, вбудований в MXML

Вбудований ActionScript – сценарій, розташований прямо всередені MXML-тегу. Як правило, сценарій виконується при здійсненні користувачем будь-яких дій, наприклад при натисканні мишею по об'єкту або при введенні тексту з клавіатури. Можна змусити компонент реагувати певним чином, коли відбувається певна подія. Типовий приклад - користувач натискає на кнопку, і це викликає зміну властивості іншого компоненту програми. Цього можна досягти за допомогою наступного ActionSrcipt-коду:

```
  <mx:Button id="myButton"
click="someComponent.someProperty ='Something' "/>
```

Все, що потрібно було зробити, - це додати тегу Button атрибут click і в цей атрибут помістити код на ActionScript. Нижче приведено приклад з реальними значеннями параметрів, які можна використовувати у додатку:

```
  <mx:Button label="Click me"
click="fullNameTextInput.text ='John Smith'"/>
```

Цей код вказує на те, що при натисканні на кнопку властивості text елемента fullNameTextInput повинно бути присвоєно значення John Smith.

fullNameTextInput.text = 'John Smith' - код на ActionScript, поміщений усередині тега. Це найпростіший спосіб додавання коду в додаток - розміщення його прямо в тезі MXML.

7 Присвоєння

Для зміни властивості потрібно здійснити операцію присвоювання. Для присвоєння нового значення властивості елемента використовується знак рівності (=), за яким слідує значення, що присвоюється. Отже, змінити значення text елемента TextInput за допомогою ActionScript можна таким чином:

```
fullNameTextInput.text = "John Smith";
```

Технічно ніякої різниці немає. Однак атрибути задають значення властивостей елементів у момент їх створення. За допомогою ActionScript присвоювання можна здійснити в потрібний момент або відкласти виконання цієї операції до появи певної події, наприклад натискання користувача на кнопку. Ви, напевно, помітили, що значення атрибута click укладено в одинарні лапки (‘), а не в подвійні ("). Це пояснюється тим, що використання подвійних лапок може привести до помилки при компіляції через неправильну інтерпретацію закінчення атрибута.

Якщо при натисканні на кнопку повинно відбуватися два або більше події, можете додати декілька виразів ActionScript в значення атрибуту click. Припустимо, при натисканні на кнопку необхідно активувати прапорець CheckBox. Код такої операції буде трохи довшим. Необхідно поставити id елементу CheckBox (expertCheckBox) і додати наступні кілька рядків у атрибут click:

```
  <mx:Button label="Click me"
click="fullNameTextInput.text = ‘John Smith’;
  expertCheckBox.selected = true "/>
```

У цьому випадку присвоюються властивості selected елемента CheckBox значення true, це означає, що елемент буде активований. Тут також здійснюється операція присвоювання за допомогою знака рівності. Зверніть увагу, що присвоювання розділяються крапкою з комою. Це при компіляції програми інтерпретується як дві операції, що відбуваються одночасно. Якщо одночасно має відбуватися три або чотири операції присвоювання, значення атрибуту click може розтягнутися на кілька рядків. На щастя, код можна розміщувати куди більш зручним чином - використовуючи функції. До того ж це дасть масу інших переваг.

8 Функції

Функція - це фрагмент коду, який може бути використаний багаторазово. Ви ставите деякий код ActionScript всередину функції, даєте функції ім'я і, коли вам потрібно виконати цей код, посилаєтеся на цю функцію.

Код на ActionScript можна розмістити в будь-якому місці MXML-файлу в якості вмісту тега <mx:Script/>. Цей спеціальний тег може бути доданий тільки в режимі Source, так як він не є візуальним компонентом. Даний тег виглядає наступним чином:

```
<mx:Script>
<![CDATA[
//Тут буде знаходитися ваш код
]]>
</mx:Script>
```

CDATA – це спеціальний об'єкт XML, який повідомляє компілятору, що не потрібно обробляти його вміст як

XML-код. Так як поміщений в нього код на ActionScript може містити такі символи, як лапки і кутові дужки (<i>), які можуть бути неправильно інтерпретовані аналізатором XML, він заключається в блок CDATA.

8.1 Створення функції

Для створення функції використовується ключове слово function, за яким слідує її ім’я та пара дужок:

```
function setForm ()
```

За описом функції необхідно помістити пару фігурних дужок ({ і }). Вони служать контейнером, що вміщує код функції. Тому далі можна вставити код з атрибуту click між ними.

```
function setForm ()
{
fullNameTextInput.text = ‘John Smith’;
expertCheckBox.selected = true;
}
```

Тепер обидві операції входять до складу функції. Для їх виконання залишається лише викликати функцію. Для цього використовується ім’я функції з круглими дужками:

```
<mx:Button label="Click me" click="setForm()"/>
```

Тепер при натисканні на кнопку буде визначено значення елемента TextInput та активований елемент CheckBox.

8.2 Параметри функції

Наявність круглих дужок відразу вказує на те, що це функція. Але вони використовуються не тільки з естетичних міркувань, а для передачі функції інформації, яка називається параметрами. По суті, параметр передає в функцію динамічну інформацію в момент її виклику. Припустимо, якщо необхідно передати функції текстове значення елементу TextInput, як у вже розглянутому прикладі. Це можна зробити через параметр функції.

```
public function setForm (txt)
{
fullNameTextInput.text = txt;
expertCheckBox.selected = true;
}
```

В даному випадку визначається строковазмінна txt, доступна тільки функції, яка всередині її містить. Значення txt задається при виконанні функції передачею значення в круглих дужках:

```
<mx:Button label="Click me" click="setForm('John
Smith')"/>
```

Тепер відображений елементом текст не статичний, а може бути змінений за допомогою даної функції. Функція може мати і кілька параметрів:

```
public function setForm (txt, sel)
{
fullNameTextInput.text = txt;
expertCheckBox.selected = sel;
}
```

Тепер виклик функції можна здійснити наступним чином:

```
<mx:Button label="Click me" click="setForm(‘John
Smith’, true)"/>
```

Досить зручно встановити значення параметрів за замовчуванням. Це дає можливість вибору при виклику функції: якщо не вводиться нове значення, використовуються налаштування за замовченням. Вони задаються в описі функції шляхом присвоювання параметрами певних значень наступним чином:

```
public function setForm (txt = "John Smith", sel
= true)
{
fullNameTextInput.text = txt;
expertCheckBox.selected = sel;
}
```

Тепер можна викликати функцію без параметрів, з одним чи з двома параметрами:

```
<mx:Button label="Click me" click="setForm()"/>
```

Використання одного параметру:

```
<mx:Button label="Click me" click="setForm(‘John
Smith’)"/>
```

Виклик функції з двома параметрами:

```
<mx:Button label="Click me" click="setForm(‘John
Smith’, true)"/>
```

9 Методи

По суті функції є методами. Говорячи про метод, мається на увазі функцію, яка є частиною класу. Наприклад створимо функцію setForm (); при цьому вона стає методом програми. Як правило, класам властиві як властивості, так і методи, і більшість елементів управління та інші об'єкти мають певні методи.

Наприклад метод addEventListener (), який реєструє в системі обробник подій. Метод addChild () поміщає задані компоненти в список для відображення. У кожного елемента управління є метод setFocus () - він робить елемент активним. Приміром, він поміщає курсор у поле елемента TextInput, так що можна відразу починати введення тексту, і користувачеві не потрібно самостійно вибирати мишею відповідне поле форми. Додамо цю функцію до додатком, щоб зробити роботу користувача з ним ще більш зручною. Виклик функції setFocus () для елемента fullNameTextInput повинен бути здійснений після закінчення завантаження програми (якщо це відбудеться раніше, функція може працювати не так, як передбачалося). Для цього потрібно зареєструвати в системі обробник події applicationComplete. Для цього необхідно додати наступний атрибут в тег Application:

```
applicationComplete="fullNameTextInput.setFocus
()"
```

10 Змінні

Змінні використовуються для зберігання інформації і подальшого її використання у додатку. Наприклад, додаток може містити такі дані, як імена користувачів; їх можна зберігати у вигляді змінних. У ActionScript змінні

визначаються за допомогою виразу var, за яким слідує ім’я змінної.

```
var userName;
```

Потім можна присвоїти змінної певне значення.

```
username = "Max";
```

Це можна зробити і відразу при створенні змінної:

```
var username = "Max";
```

Можна встановити різний рівень доступу до змінних, як і у випадку з функціями:

```
public var username = "Max";
```

11 Типи даних

У програмуванні типізацією називається спосіб визначення виду значень, який може бути присвоєний даній змінній. Це допомагає розробнику зорієнтуватися, якого роду інформація потрібна. Коли заздалегідь відомий тип даних, що обробляються, збільшується продуктивність програми.

Ще однією перевагою типізації даних є той факт, що різні типи даних володіють різними властивостями і методами. Безумовно, з різними типами даних можна виробляти абсолютно різні операції, і така передбачливість може істотно полегшитпрограмування.

Таблиця 11.1 – Основні типи даннях

Назва	Описання	Приклад	Значення за замовчуванням
String	Простий текст	var hi:String = “Hello!”	null
Number	Дійсне число	var pi:Number = 3,14	NaN
uint	Ціле число без знаку	var ultimate:uint = 25	0
int	Дійсне число	var neg:int =-40	0
Boolean	Логічна змінна	var isHappy:Boolean = true	false
void	Означає відсутність значень викликаної функції.	function doNothing():void { }	undefined

Припустимо, додаток містить форму, в якій користувач повинен вказати своє ім’я і вік. Очевидно, що ім’я буде представляти собою інформацію у вигляді рядка, а вік буде мати числове значення. Так як це позитивне ціле число, то, звернувшись до табл. 2, можна побачити, що краще всього використовувати тип uint, проте підійде і Number.

При оголошенні змінної можна з легкістю задати тип її значень. Для цього використовується двокрапка, що слідує відразу за ім’ям змінної (або за круглими дужками, якщо мова йде про функції). Для визначення типу значень змінної userName як String потрібно зробити наступне:

```
var userName: String = "Hello";
```

Тобто необхідно всього лише додати двокрапку і тип даних. Те ж саме і з числовим значенням. У наступному коді створюється змінна pi і тіпізіруется як Number:

```
var pi: Number = 3.14;
```

Все це справедливо і для функцій. Вони здатні повертати значення, яке можна типізувати. Для визначення типу значення можна ввести двокрапку після круглих дужок, а потім тип значення, а для повернення значення використовується ключове слово return. Наприклад, наступна функція повертає значення суми 2+2.

```
public function doSomeMath (): Number
{
    return 2+2;
}
```

Для подальшого звернення до отриманого значення можна присвоїти функці] значення змінної:

```
var myMath: Number = doSomeMath ();
//Значення змінної myMath буде дорівнює 4
```

Тепер можна вдосконалити створену раніше функцію setForm (), задавши тип змінних, що входять до неї. Можна типізувати параметри функції, а також саму функцію:

```
public function setForm (txt: String = "John
Smith", sel: Boolean = true): void
{
    fullNameTextInput.text = txt;
expertCheckBox.selected = sel;
}
```

12 Об’єкти

З точки зору об’єктно-орієнтованої мови програмування, такої як ActionScript, ми маємо справу виключно з різного роду об’єктами. У певному сенсі об’єкт – це універсальний контейнер. Він може містити текст або числове значення, метод управління даними і навіть входити до складу інших об’єктів. Об’єкт можна представити як щось, що володіє певним станом (тобто він може включати в себе змінні) і поведінкою (методи роботи з цими змінними). Це основа будь-якого Flex-додатку, тому що, він цілком і повністю складається з об’єктів. Додаток є об’єктом. Елементи програми також є об’єктами, будь-яка створювана змінна є об’єктом (наприклад, властивості елементів). Припустимо, потрібно створити об’єкт car з усіма притаманними такого роду об’єкту властивостями, такими як тип автомобіля або його колір. За допомогою ActionScript дана задача може бути виконана наступним чином. Спочатку створюється екземпляр Object, якому присвоюються певні властивості:

```
var car: Object = new Object ();
car.type = "sports car";
car.color = "red";
car.topSpeed = 270;
car.isInsured = false;
car.driver = undefined;
```

Таким чином, вийшов потужний спортивний автомобіль, причому незастрахований. Такому автомобілю необхідний водій. Вирішимо цю задачу шляхом створення ще одного об’єкту:

```
var person: Object = new Object ();
person.name = "Schumacher";
person.age = 41;
```

Тепер, коли є прекрасна кандидатура на роль водія (об'єкт), можна змінити властивість driver (також об'єкт) об'єкту car:

```
car.driver = person;
```

Якщо у додатку використовується безліч таких автомобілів та водіїв, причому кожен об'єкт володіє певним набором властивостей, то варто використати класи.

13 Класи

Клас – це сукупність інформації і властивостей, характерних для певної категорії об'єктів, при цьому об'єкт є екземпляр класу.

Наприклад, створений об'єкт car має властивості color і type. Щоб створити ще один подібний об'єкт, потрібно провести таку ж операцію, наділивши його необхідними властивостями. Можна піти більш логічним шляхом, створивши клас Car і вказавши всі властивості, які ви маєте намір використати для такого роду об'єктів. Таким чином, при створенні екземпляра даного класу car вже відомий набір властивих йому властивостей. Це дасть масу переваг, адже тепер можна не боятися помилок і немає необхідності згадувати, як називалася та чи інша властивість. Клас Car, створений на мові ActionScript, виглядає наступним чином:

```
public class Car
{
    var type: String;
    var color: uint;
    var topSpeed: int;
    var isInsured: Boolean;
```

```
    var driver: Person;
}
```

Зверніть увагу, що властивість driver класу Car має тип даних Person. Це вказує на існування класу Person, який може виглядати приблизно так:

```
public class Person
{(
    var name: String;
    var age: int;
}
```

Необхідно зауважити, що всі властивості класу суворо типізовані, тобто age об’єкта Person може бути лише цілим числом, а властивість isInsured - приймати одне зі значень true або false. Створюючи нові екземпляри даних класів, ви заздалегідь знаєте, якими властивостями (і яких типів) вони будуть володіти. Використовуючи нові класи, знову створимо об’єкти Car і Person.

```
var car: Car = new Car ();
car.type = "sports car";
car.color = 0xFF0000;
car.topSpeed = 270;
car.isInsured = false;
car.driver = undefined;
var person: Person = new Person ();
person.name = "Schumacher";
person.age = 41;
car.driver = person;
```

На практиці можна відчути переваги класів, тому що елементи управління, які використовуються у Flex засновані на класах. Наприклад, елемент Button є екземпляром класу Button, і незалежно від того, якою мовою написаний код (ActionScript або MXML), він буде мати один і той же набір властивостей.

14 Зв’язування даних

Метод зв’язування даних є одним з переваг Flex – він дозволяє легко оперувати інформацією. Це простий спосіб звернення до даних, що дозволяє відслідковувати їх зміни без додаткових труднощів. Дані можуть являти собою текст, наприклад ім’я, або інформацію у вигляді впорядкованого списку; дані – це просто інформація в будь-якій формі. По суті, зв’язування даних надає розробникам зручний спосіб передачі і використання інформації всередині додатків. Зв’язування може здійснюватися між певними властивостями різних елементів, між властивостями елемента і моделлю даних і між різними моделями даних.

14.1 Основні принципи використання

Як приклад, який демонструє сутність даного методу, наведемо процес зв’язування властивостей різних елементів. Властивість text елемента Label прив’язується до властивості text елемента TextInput:

```
  <mx:TextInput id="helloTextInput" text="Hello,
World"/>
  <mx:Label text="{ helloTextInput.text }"/>
```

У даному прикладі фраза «Hello, World» буде відображатися в обох елементах – TextInput і Label – завдяки здійсненню прив’язки властивості text елемента Label до відповідної властивості text елемента TextInput. Фігурні дужки ({ і }), що оточують посилання на властивість, до якої здійснюється прив’язка, використовуються для відмінності зв’язаних даних від звичайного тексту MXML – якщо їх опустити, властивості

text елемента Label буде присвоєно значення «helloTextInput.text». Цей приклад показує найбільш простий і часто використовуваний спосіб зв'язування даних у Flex. При запуску програми відображатися однаково буде не тільки вже існуючий текст, але, внесені до TextInput зміни, відіб'ються і в Label. Тобто після введення тексту в TextInput цей текст тут же автоматично буде відображатися і в Label. Необхідно звернути увагу на те, що прив'язка здійснюється не до елементу TextInput в цілому, а тільки до його властивості text. Таким чином, наступний код є некоректним:

```
<mx:TextInput id="helloTextInput" text="Hello,
World"/>
<mx:Label text="{ helloTextInput }"/>
```

При виконанні програми не відбудеться помилки, але і результат буде не таким як треба. Flex вважатиме, що здійснена прив'язка до всього елементу TextInput з ідентифікатором helloTextInput, і елемент Label відобразить щось на зразок «ApplicationName0.helloTextInput». В якості ще одного прикладу створимо дві змінні для імені та прізвища, використовуючи тег String. Тег String створює рядкову змінну, яку потім прив'яжемо до властивості text елемента Label.

```
<mx:String id="firstName">Max</mx:String>
<mx:String id="lastName">Demidenko</mx:String>
<mx:Label id="nameLabel" text="{firstName}"/>
```

14.2 Множинність адресатів прив'язки

Вище був описаний спосіб прив'язки імені до різних компонентів програми. Проте зв'язування даних не

обмежується єдиним джерелом і адресатом; кілька різних адресатів можуть бути прив'язані до одного й того ж джерела. Розглянемо приклад, в якому ім'я прив'язується до властивості text елемента Label і властивості label елемента Button.

```
  <mx:String id="displayName">Max</mx:String>
  <mx:Label id="nameLabel" text="{displayName}"/>
  <mx:Button id="nameButton"
label="{displayName}"/>
```

14.3 Двонаправлене зв'язування

За своєю природою зв'язування даних – процес однобічний. Визначається джерело та адресат прив'язки, і інформація передається від джерела до адресата. Проте можливості не обмежено таким механізмом. Можна здійснювати й двостороннє зв'язування наступним чином:

```
  <mx:TextInput id="oneTextInput"
text="{anotherTextInput.text}"/>
  <mx:TextInput id="anotherTextInput"
text="{oneTextInput.text}"/>
```

У цьому випадку два елементи TextInput прив'язані один до одного. При зміні значення одного з них, значення іншого змінюється відповідно. Того ж можна досягти і використовуючи тег <mx:Binding/>:

```
  <mx:Binding source="oneTextInput.text"
destination="AnotherTextInput.text"/>
  <mx: Binding source = "anotherTextInput.text"
destination ="OneTextInput.text" />
  <mx:TextInput id="oneTextInput"/>
  <mx:TextInput id="anotherTextInput"/>
```

14.4 Зберігання структурованої інформації

Flex – це зручність зберігання структурованої інформації в моделі даних. Модель даних – це окремий об'єкт з деякою кількістю заданих властивостей, тобто спосіб зберігання інформації в одному місці. Наприклад, замість того, щоб створювати окремі змінні, що містять ім'я і прізвище людини, краще включити ці дані в одну змінну name, що володіє властивостями first, middle, last, навіть title, suffix і т. д. Це і є модель даних. Можна ускладнити завдання і створити модель даних, яка містить інформацію про користувача, включаючи його адресу, телефон, адреса електронної пошти і навіть власне цей об'єкт name.

Основні принципи використання

Використання моделей даних істотно спрощує організацію коду, що є незаперечною перевагою. Якщо з часом знадобиться здійснювати доступ до даних, що зберігаються на сервері, практичніше відразу запитувати сукупність даних, ніж кожен раз надсилати запит, щоб отримати спочатку ім'я, потім адресу електронної пошти, адресу та т . д. Можна зберігати всю інформацію в моделі даних, тоді досить буде одного запиту для її отримання. Для цього використовується тег <mx:Model/>. Нижче приведено приклад створення моделі даних з ім'ям model за допомогою даного тега:

```
<mx:Model id="model">
<info>
<name>
<firstName> Tim </ firstName>
<lastName> O'Reilly </ lastName>
</name>
<email>max@gmail.com </email>
```

```
  <phone> (068) 641-43-43</phone>
  </info>
  </mx:Model>
  <mx:Binding source="areaCode.phone"
destination="nameLabel.text"/>
  <mx:Label id="nameLabel"/>
```

<mx:Model/>– єдиний MXML-тег, всередині якого можна структурувати дані за допомогою XML. Таким чином, чітко організована і представлена у зручному для сприйняття вигляді інформація про користувача зберігається в одному місці. Відомості про ім'я користувача, адресу його електронної пошти і номер телефону створені за допомогою XML-тегів. Зверніть увагу на кореневій тег <info/>. Його наявність необхідно для безпомилкової інтерпретації XML-тегів, але вінможе називатися й по-іншому.

Для доступу до цієї інформації у виразі прив'язки необхідно вказати ідентифікатор моделі даних і наступну через крапку властивість. Наприклад, звернення до номера телефону користувача буде виглядати так: model.phone.

14.5 Багаторівневе зв'язування

Вирази прив'язки можна використовувати всередині моделі. В наступному прикладі створюється строкова змінна з ім'ям areaCode, що містить текст «0038». Використання виразів прив'язки, укладених у фігурні дужки, між тегами <mx:Binding> і </mx:Binding> додає код країни в початок телефонного номера користувача, що є частиною цієї моделі.

```
  <mx:String id="areaCode">0038</mx:String>
  <mx:Model id="model">
  <info>
  <name>
```

```
   <firstName>Max</firstName>
   <lastName>Demidenko</lastName>
   </name>
   <email>max@gmai.com </email>
   <phone>{areaCode}068-641-63-63</phone>
   </info>
   </mx:Model>
   <mx:Binding source="model.phone"
destination="nameLabel.text"/>
   <mx:Label id="nameLabel"/>
```

Як і в попередніх випадках, властивість text елемента Label прив'язується до телефонного номеру в цій моделі даних. При запуску програми текст, відображений елементом Label, буде виглядати наступним чином: «0038-068-641-63-63».

Іншими словами, було здійснено багаторівневе зв'язування. Перш за все, при створенні телефонного номера використовується прив'язка до коду країни. Потім здійснюється прив'язка елемента Label до значення телефонного номера. Таким чином, зміна коду країни тягне за собою відповідну зміну номера телефону і відображається у Label тексту. Це дуже потужний інструмент з найширшими можливостями. У моделях даних доступні всі функції методу зв'язування з використанням фігурних дужок, в тому числі приведення типів даних. Це означає, що всередині моделі можна зв'язувати рядки і використовувати декілька синтаксичних конструкцій з фігурними дужками. Іноді це називається підгонкою даних (data massaging) – це конкатенація рядків, виконання математичних операцій і, можливо, форматування даних для відображення. Ми розглянули простий приклад, в якому для отримання повного номера телефону об'єднувалися код міста і номер телефону. Можна піти ще далі і розширити модель даних таким чином:

```
   <mx:Model id="model">
   <info>
   <name>
   <title> Mr. </ title>
   <firstName> Tim </ firstName>
   <lastName> O'Reilly </ lastName>
   <displayName>{model.name.title}{model.name.lastN
ame}
   </displayName>
   </name>
   <greeting> Hello, {model.name.displayName}</
greeting>
   <email> tim@oreilly.com </ email>
   <areaCode> 707 </ areaCode>
   <phone>{model.areaCode} 827-7000 </ phone>
   </info>
   </mx:Model>
   <mx:Binding source="model.greeting"
destination="nameLabel.text"/>
```

Такий код ілюструє найширші можливості методу багаторівневого зв'язування даних. У даному випадку створюється властивість displayName і прив'язується до прізвища користувача і способу звертання до нього. У свою чергу ця властивість використовується у властивості greeting, яка також прив'язана до елемента Label.

15 Можливості перевірки даних

За допомогою Flex можна з легкістю створювати зручні та привабливі форми; для цього достатньо оволодіти базовими інструментами перевірки даних.

Використання інструментів перевірки даних (валідації)

Щоб навчитися прийомам перевірки даних прямо на практиці, встановимо перевірку для кожного поля програми Contact Editor. Почнемо з першого поля, призначеного для введення імені, яке має бути заповнено в

обов’язковому порядку, що вказується в атрибуті required його контейнера FormItem. Однак додавання цього атрибуту лише вказує на те, що поле вимагає заповнення, але не перевіряє введені дані на наявність помилок. Для цього використаємо невидимий компонент StringValidator.

StringValidator

StringValidator – основний інструмент перевірки даних, що підтверджує наявність введеного тексту. Для його використання необхідно розмістити тег <mx:StringValidator/> у верхній частині MXML-коду, не виходячи за межі тега <mx:Application/>. Таке розташування необов’язково, але традиційно прийнято розміщати невидимі компоненти саме у такий спосіб; це полегшує їх пошук і зміну. Тегу <mx:StringValidator/> можна присвоїти атрибут source, який посилається на елемент, за яким має здійснюватися контроль. Атрибут property вказує на властивість цього елемента, що підлягає перевірці. Наприклад, щоб встановити перевірку для поля firstNameTextInput, потрібно використовувати даний тег наступним чином:

```
<mx:StringValidator id="firstNameValidator"
source="{firstNameTextInput}"
property="text"/>
```

EmailValidator

Наступним пунктом у списку є поле для введення адреси електронної пошти. Для перевірки правильності введених даних зручно використовувати компонент EmailValidator. Принцип його дії аналогічний StringValidator, так що просто можна додати даний тег в код додатку і зв’язати його з елементом emailTextInput. Щоб урізноманітнити програму, можна придумати власний

текст повідомлення про помилку, яке відображається за допомогою властивості requiredFieldError.

Однак даний компонент має безліч підготовлених повідомлень про помилки та можливостей їх налаштування – для цього існують властивості missingAtSignError, invalidDomainError і т. п. Зовсім не обов'язково писати свій власний текст для кожного з цих випадків, – значення за замовчуванням підійдуть у будь-якій ситуації. Однак варто знати про наявність такої можливості.

```
<mx:EmailValidator id="emailValidator"
source="{emailTextInput}"
property="text"
requiredFieldError="An at sign (@) is missing in
your e-mail address.1"/>
```

Тепер за відсутності введеного e-mail адреси програма видасть повідомлення про помилку. Якщо не змінювати тексти повідомлень, які використовуються за замовчуванням, при введенні неправильної адреси електронної пошти користувач отримає докладний сповіщення про те, що саме слід виправити.

PhoneNumberValidator

Наступний пункт – перевірка телефонного номера. Для цього використовується інструмент PhoneNumberValidator. Дане поле також не вимагає обов'язкового заповнення, але в тому випадку, коли дані введені, необхідно здійснити їх перевірку.

```
<mx:PhoneNumberValidator id="phoneValidator"
source="{phoneTextInput}"
property="text"
required="false"/>
```

Основною особливістю даного компонента є (за замовчуванням) прийняття численних можливих форматів номера телефону, тобто користувач може вводити (415) 555-8273, 415-5558273 або навіть 415 555 8273. Однак введена фраза I don't have a phone («У мене немає телефону») не пройде перевірку.

Інші валідатори

Завдяки широким можливостям елемента DateField немає необхідності валідації даних, що вводяться. Проте при бажанні з цією метою можна використовувати клас DateValidator. Крім того, існує маса інших валідаторів, таких як CreditCardValidator, CurrencyValidator, NumberValidator і SocialSecurityValidator.

Основні способи валідації

Валідатори видають повідомлення про помилку тільки в тому випадку, коли деякі дані введені (або не введені) і фокус зміщується на наступне поле. Це відбувається завдяки подіям, що викликають запуск процесу валідації. Будь-який компонент валідації має властивість trigger, що посилаються на елемент, за якими здійснюється контроль. За замовчуванням його значення співпадає із значенням властивості source. Також можна використовувати властивість triggerEvent, привласнюючи йому як значення назви події, після якого має бути здійснена перевірка даних. За замовчуванням такою подією є valueCommit; воно відбувається при закінченні введення даних, що як правило визначається зміщенням фокусу на інший елемент, але дана подія може бути викликана і при зміні значення елемента у сценарії.

Наприклад, для запуску перевірки даних у полі, щоб ввести адресу електронної пошти (emailTextInput), як

тільки користувач почне введення тексту, потрібно використовувати властивість triggerEvent зі значенням change, тобто подію, що відбувається при внесенні змін в полі TextInput. При цьому значення властивості trigger валідатора залишено за замовчуванням і збігаються зі значенням його властивості source (у даному випадку – emailTextInput).

```
  <mx:EmailValidator id="emailValidator"
  source="{emailTextInput}"
  property="text"
  requiredFieldError="Please enter your email.I
promise not to send spam."
  triggerEvent="change"/>
```

Даний приклад наочно демонструє особливості функціонування тригерів, однак не варто використовувати такий код в реальному додатку, оскільки протягом всього процесу введення даних користувачеві буде видаватися повідомлення про помилку до тих пір, поки адреса електронної пошти не буде введений повністю. Однак тригери можна застосовувати по-іншому. Можна додати у програму елемент Button, який буде виконувати функцію кнопки, що підтверджує введення або відправлення даних.При додаванні елемента Button у форму за допомогою режиму редагування Design він автоматично буде поміщений у відповідний контейнер FormItem, призначений для присвоєння елементу мітки і вирівнювання полів форми. Длявирівнювання досить скористатися FormItem, але в присвоєнні мітки елементу Button немає ніякого сенсу, оскільки вона у нього вже є. Тому в якості значення властивості label контейнера треба залишити порожній рядок. Далі необхідно задати кнопці ідентифікатор (id) submitButton.

Тепер саме час вказати валідатор EmailValidator на створену кнопку з допомогою його властивості trigger. Оскільки перевірка даних має буде запущена при натисканні на кнопку, необхідно задати властивості triggerEvent значення click:

```
<mx:EmailValidator id="emailValidator"
source="{emailTextInput}"
property="text"
requiredFieldError="Please enter your email.I promise notto send spam."
trigger="{submitButton}"
triggerEvent="click"/>
```

Тепер при натисканні користувачем на кнопку submitButton буде запущено процес валідації даних, введених в поле emailTextInput. Таким чином була змінена поведінка валідатора, що використовується за замовченням, і перевірка даних при зміні фокусу здійснюватися не буде – це відбудеться тільки при натисненні на кнопку. Для більш повноцінного контролю за здійсненням перевірки даних можна застосовувати метод validate () власне об'єкта валідатора, який дозволяє запустити даний процес у будь-який час. Для прикладу створимо функцію, яка викликає метод validate () об'єкта EmailValidator.

```
private function validateAndSubmit (): void
{
emailValidator.validate ();
}
```

Після того як для елемента submitButton буде встановлений обробник події click, який посилається на дану функцію, натискання на кнопку послужить тригером валідації даних, введених в поле для адреси електронної

пошти. Завдяки такому прийому перевірка даних буде здійснена при підтвердженні користувачем заповнених даних. При цьому є можливість зробити перевірку ще раз, перш ніж відправляти дані. Дана техніка валідації не передбачає виклику функції validate () для кожного окремого валідатора. Існує Допоміжна функція validateAll(), що є статичним методом, належить до класу mx.validators.Validator (базового класу для всіх валідаторів). Замість створення об'єкта Validator і виклику даного методу для об'єкта метод може бути викликаний для самого класу Validator. Метод validateAll () має єдиний параметр, значенням якого є масив валідаторів. Спочаткунеобхідно імпортувати клас mx.validators.Validator, а потім внести у функцію такі зміни:

```
private function validateAndSubmit (): void
{
var validators: Array = [firstNameValidator,
lastNameValidator, emailValidator,
phoneValidator, zipCodeValidator];
Validator.validateAll (validators);
}
```

Тепер натискання на кнопку submitButton буде сигналом до початку процесу перевірки даних для всіх перерахованих валідаторів. Цю форму можна зробити більш зручною з точки зору користувача. Передбачається, що якщо користувач натискає на кнопку, він упевнений, що форма заповнена правильно. Якщо це не так, поява червоних рамок навколо невірно заповнених полів може залишитися непоміченим. Є можливість використання компонента Alert, що дозволяє вивести повідомлення про помилку у спливаючому вікні. Для відкриття такого вікна використовується статичний метод show () класу Alert. Цей

метод має два параметри. Перший з них містить відображуваний текст повідомлення про помилку, другий – рядок заголовка вікна Alert. Таким чином, необхідно імпортувати клас mx.controls.Alert і здійснити виклик спливаючого вікна за допомогою Alert.show ("Please fix that stuff.", "There were problems with your form.").

Для перевірки наявності або відсутності помилок при заповненні форми можна використовувати масив, що повертається функцією Validator.validateAll (). Значення його властивості length визначає довжину масиву, тобто кількість елементів,що містяться в ньому. Якщо довжина масиву більше нуля, значить були допущені помилки. З цією метою використовується умовний оператор, що дозволяє визначити істиний або хибний вираз. Скористаємося, наприклад, виразом if. Він здійснює виконання певного фрагмента коду при виконанні певних умов: якщо передається в нього значення істина, буде виконуватися код, розміщений у фігурних дужках. В іншому випадку він буде пропущений. Для прикладу розглянемо програму, у якій необхідно здійснити доступ до масиву, що повертається методом Validator.validateAll(). Якщо його довжина (length) перевищує нульове значення, необхідно вивести повідомлення про помилку у спливаючому вікні.

```
  private function validateAndSubmit():void
  {
  var validators:Array=[firstNameValidator,
  lastNameValidator, emailValidator,
  phoneValidator, zipCodeValidator];
  var
errors:Array=Validator.validateAll(validators);
  if (errors.length> 0)
  {
  Alert.show ("Please fix that stuff.",
  "There were problems with your form.");
```

```
}
}
```

16 Використання елементів управління списком

У Flex існує ряд елементів управління, призначених для відображення даних у вигляді списку. Всі вони працюють з даними зі структурою будь-якого ступеня складності та надають розробнику широкі можливості їх налаштування і багаторазового використання.

Якщо довжина списку перевищує розмір елемента, з'являються смуги прокручування, що дозволяють побачити його вміст повністю. Нижче представлений перелік таких елементів, найбільш часто використовуваних при розробці програми:

List

База для всіх елементів керування для відображення списків. Має у своєму розпорядженні пункти списку по вертикалі.

HorizontalList

Має у своєму розпорядженні пункти списку по горизонталі.

TileList

Елементи розміщуються подібно плитці.

ComboBox

Цей елемент нагадує TextInput, але дає користувачеві додаткову можливість вибору вводити значення за допомогою випадаючого списку. Його можна умовно порівняти з HTML-тегом <select/>.

DataGrid

Елемент з розширеними можливостями, призначений для табличної організації наборів даних. Можна сортувати ряди, а також змінювати розмір колонок і навіть міняти їх місцями, просто перетягуючи за допомогою миші.

Прості списки даних

Інформація, представлена у вигляді списку елементів, як правило, зберігається в масиві.Масив можна створити за допомогою MXML-тега <mx:Array/>. Елементи списку розташовуються всередині вкладених тегів, наприклад <mx:String/>.

Наприклад, список з назвами кольорів можна створити таким чином:

```
<mx:Array>
<mx:String> red </mx:String>
<mx:String> green </mx:String>
<mx:String> blue </mx:String>
</mx:Array>
```

Для створення масиву в ActionScript перелік його елементів, розділених комою, розміщується у квадратних дужках ([i]). Таким чином, точно такий же список можна створити і засобами ActionScript:

```
var colors: Array = ["red", "green", "blue"];
```

Перейдемо до способу відображення масивів за допомогою спеціальних елементів. Для цього використовується властивість dataProvider, що передає масив в елемент керування списком.

Наприклад, щоб вивести на екран перелік кольорів за допомогою елемента управління List, можна прив’язати дану властивість до відповідного масиву:

```
  <mx:Listwidth = "150"
  dataProvider="{[‘Red‘,’Orange‘,’Yellow‘,’Green‘,
’Blue‘,’Black‘,‘Violet’]}"/>
```

Складні списки даних

У попередніх двох прикладах джерелом даних для списку був масив рядків. Елемент керування списком може працювати з масивами довільних даних. Наприклад, необхідно скласти список улюблених пісень. Звичайно, можна обмежитися лише назвами, але можна додати і деяку додаткову інформацію, наприклад ім’я виконавця і альбом, в який входить дана пісня. У такому разі замість масиву текстових рядків набагато зручніше скористатися масивом об’єктів. Об’єкт є ідеальним контейнером для зберігання набору даних, оскільки йому можна присвоїти будь-які властивості. Для цього знадобляться властивості artist, album і song:

```
<mx:Listwidth = "150">
<mx:dataProvider>
<mx:Array>
<mx:Object
song="In My Secret Life"
album="Ten New Songs"
artist ="Leonard Cohen"/>
<mx:Object
song="Phantom Limb"
album="Wincing the Night Away"
artist="The Shins"/>
<mx:Object
song="Tinfoil"
album="Live at Schuba’s Tavern"
artist="The Handsome Family"/>
<mx:Object
song="Highway 253"
album="Extra Solar Sunrise"
artist="The Saturn V"/>
<mx:Object
song="Junk Bond Trader"
album="Figure 8"
```

```
artist="Elliott Smith"/>
<mx:Object
song="Stalled"
album="Through the Trees"
artist="The Handsome Family"/>
<mx:Object
song="Every Dull Moment"
album="Bring on the Snakes"
artist="Crooked Fingers"/>
</mx:Array>
</mx:dataProvider>
</mx:List>
```

У результаті вийшов список, пунктами якого являються об’єкти із заданими властивостями.

17 Стани програми

Відмінним інструментом для побудови динамічного і гнучкого інтерфейсу користувача є стани програми. Завдяки різним станам інтерфейс може виглядати по-різному в конкретний момент часу і в залежності від певних цілей. Наприклад, веб-додаток включає в себе як сторінку для авторизації користувача, так і сторінку призначенудля настройки користувача – ці сторінки можна вважати HTML-еквівалентами станів у додатку Flex.

Стан – це набір логічно згрупованих змін інтерфейсу користувача. По суті, стан – це сукупність змін властивостей, стилів і типів поведінки компонента. Аналогічного ефекту – внесення ряду одноразових змін до інтерфейс програми – можна добитися і за допомогою набору функцій, але стани Flex, які реалізуються з допомогою MXML, набагато простіше створювати і модифікувати.

Типові випадки використання станів

Розглянемо створений додаток для пошуку по заданих критеріях. При завантаженні даної програми користувач побачить поле для введення запиту і список результатів пошуку. Але дивно показувати список результатів до їх отримання після відправлення запиту. Звичайно, можна зробити список результатів невидимим, а при надходженні даних відобразити його, змінивши налаштування видимості. Але в ідеалі, звичайно, хотілося б внести більше змін до цієї сторінки. При відправці пошукового запиту користувачем поле пошуку можна перемістити у верхню частину програми, щоб звільнити місце для відображення результатів пошуку. Це не тільки виглядає стильно і живо, але дає користувачеві додаткові зручності при роботі з додатком, оскільки в кожний момент часу відображається саме те, що йому зараз потрібно.

Зміна властивостей, стилів і подій стану

Після створення нового стану воно буде активовано на панелі States. Це означає, що виготовлені зміни в режимі Design будуть застосовані саме до цього стану. Достатньо лише перетягнути кнопку до правого краю сцени і перейти між двома різними станами (основним і тільки що створеним станом stageRight) за допомогою панелі States. В результаті можна побачите, що в первинному стані кнопка знаходиться в лівій частині програми, але при переході в стан stageRight вона переміщається до її правого краю. Перехід між станами можна здійснювати і за допомогою випадаючогосписку, розташованого на панелі інструментів у режимі Design.

При виконанні вищеописаних операцій у режимі Design генерується необхідний MXML-код. Будь-який додаток Flex має властивість states. Воно містить список його можливих станів, які можуть бути визначені за допомогою

тега <mx:State/>. У цього тега є властивість name, що відповідає назві стану. Він може включати різні вкладені теги. У прикладі зі створенням двох станів програми згенерований код може виглядати приблизно таким чином:

```
  <mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"layout="a
bsolute">
  <mx:states>
  <mx:State name="stageRight">
  <mx: SetProperty
  target = "(button1)"
  name = "x"
  value = "570" />
  </mx: State>
  </mx: states>
  <mx: Button id = "button1"
  x = "10"
  y = "10"
  label = "Click to move"
  click = "currentState = ‘stageRight’" />
  </mx:Application>
```

Зверніть увагу на стандартний компонент Button в нижній частині коду – його можна назвати компонентом в основному стані. Між відкриваючими і закриваючими тегами <mx:states> і </mx:states> розташований єдиний тег з ім’ям stageRight. Його вміст описує зміни, які відбудуться при виборі цього стану. У прикладі зміни торкнуться властивості x елемента Button, яка має прийняти значення 570 замість 10. Процес зміни здійснюється за допомогою тега <mx:setProperty/>(трьох його властивостей). Властивість target приймає ім’я id модифікуємого компонента, name вказує на ім’я змінюваної властивості, а value відповідає за її нове значення в даному стані.

Для перемикання станів використовується властивість програми currentState. У прикладі перехід до стану stageRight відбудеться при натисканні на кнопку.

17.1Додавання компонентів

При використанні станів можна також додавати і видаляти компоненти. Це означає можливість модифікування цілих частин програми за допомогою MXML-коду. Наприклад,на веб-сторінках формахзазвичай пропонується авторизація користувача. Як правило, запитується ім'я користувача та пароль, але якщо користувач ще не реєструвався на даному сайті, доведеться перейти за вказаним URL на сторінку реєстрації, де потрібно заповнити своє справжнє ім'я, логін і пароль. Але було б набагато зручніше, якби поля, необхідні для реєстрації користувача, з'являлися у разі необхідності на тій же сторінці. У наступному прикладі створимо таку функцію.

На початку коду створюється панель Panel і вкладений у неї контейнер Form, що містить поля для введення імені користувача і пароля.

```
  <mx:Application        xmlns:        mx        =
"http://www.adobe.com/2006/mxml"
  verticalAlign = "middle">
  <mx: Panel id = "loginPanel"
  title = "Returning Users Sign In"
  horizontalAlign = "right"
  paddingLeft = "5"
  paddingRight = "5"
  paddingTop = "5"
  paddingBottom = "5">
  <mx:Form id="loginForm">
  <mx:FormItem id = "usernameFormItem"
  label="Username:">
  <mx:TextInput/>
```

```
</mx:FormItem>
<mx:FormItem id = "passwordFormItem"
label = "Password:">
<mx:TextInput
displayAsPassword = "true" />
</mx:FormItem>
</mx:Form>
<mx:Button id = "submitButton"
label = "Sign in" />
<mx:ControlBar
horizontalAlign = "right">
<!--Елемент LinkButton здійснює перехід
до іншого станом при клацанні по ньому мишею -->
<mx:LinkButton id = "registerLink"
label = "Don't have an account yet?"
color = "# 1B337B"
click = "currentState = 'registration'" />
</mx: ControlBar>
</mx: Panel>
</mx: Application>
```

18 Поведінки, переходи, фільтри

Всі візуальні компоненти Flex мають стандартний набір графічних фільтрів і ефектів, що дозволяють без особливих зусиль створювати виразні програми. Ефекти – це звукові або візуальні зміни компонентів, найчастіше з розряду анімації. Фільтри – статичні візуальні зміни, такі як розмивання меж або створення тіней.

Поведінки

Будь-який Flex-компонент має вбудований набір поведінок (реакцій), що дозволяють додавати виразні ефекти і оживляють програми. Їх різноманітність безмежна, але найчастіше використовуються ефекти, що створюють ілюзію поступового зникнення і появи компонента, або анімація, простежується шлях компоненту при його переміщенні. Також існують аудіоефекти, тобто

супровід певних подій або дій користувача певними звуками.

Поведінка – це ефект, що викликається тригером, тобто певною дією, що відбувається у додатку. Типовими прикладами таких подій є натиснення мишею по елементу, переміщення курсору або особливі події самого компонента, такі як його створення, відображення або приховування.

Налаштування ефектів

Поки ми використовували поведінки з ефектом, налаштованим за замовчуванням, тобто як значення поведінки вказувалося просто ім'я класу ефекту, наприклад, Fade або Resize. Це дуже зручно, але в певних випадках необхідний більш тонкий контроль за властивостями ефекту. З цією метою створюється об'єкт ефекту за допомогою відповідного MXML-тега. Це дозволить створювати різні види одного і того ж ефекту для застосування в різних випадках. Щоб створити об'єкт ефекту Resize, необхідно додати тег <mx:Resize/> у додаток ContactManager і задати йому ідентифікатор fastResize. Як і будь-який інший ефект, він має властивість duration, визначальним часовий проміжок, що вимірюється в мілісекундах, протягом якого має бути відображений ефект. Наприклад, можна поставити йому значення 300. Оскільки триста мілісекунд становить приблизно третину секунди, ефект повинен бути досить живим:

```
<mx:Resize id="fastResize" duration="300"/>
```

На оголошений ефект Resize можна в подальшому посилатися в коді програми за його id. Можна замінити поточну поведінку resizeEffect компонента ContactViewer

цим конкретним екземпляром. Замість стандартного ефекту Resize, що досягається шляхом присвоювання відповідного значення поведінки, можна передати йому спеціальним чином налаштований ефект fastResize:

```
<view:ContactViewer id = "contactViewer"
contact = "(contactsDataGrid.selectedItem)"
x = "318"
y = "10"
resizeEffect = "(fastResize)"
horizontalScrollPolicy = "off"
verticalScrollPolicy = "off">
</view: ContactViewer>
```

Спільне використання декількох ефектів

Resize чудово підходить для додавання ефекту анімації при збільшенні розмірів ContactViewer, коли здійснюється перехід з режиму відображення даних в режим редагування. Однак можливості Flex цим не обмежуються. Було б непогано додати ще й ефект Dissolve, що здійснює плавний перехід з одного режиму на інший. Розглянемо деякі ефекти.

Parallel

Для вирішення цього завдання зовсім не обов'язково замінювати ефект Resize ефектом Dissolve – останній можна просто додати. Завдяки наявності спеціального тега Parallel можна накладати кілька ефектів одночасно. Для цього всі вони повинні бути розміщені між тегами <mx:Parallel/>. Це дає розробнику можливість використання і налаштування комплексних ефектів. У нашому випадку повинно відбутися одночасне застосування ефектів Dissolve і Resize, тому необхідно замінити попередній код об'єкта Resize на наступний:

```
<mx:Parallel id="fadeAndResize">
```

```
<mx:Dissolve/>
<mx:Resize id="fastResize" duration="300"/>
</ mx: Parallel>
```

Наступний крок – зв'язування resizeEffect компонента ContactViewer з набором ефектів fadeAndResize, після чого при зміні розмірів компонента будуть відображені обидва ефекти: Dissolve і Resize. Це допоможе уникнути обрізання вмісту, що відбувається при використанні одного тільки ефекту Resize. Таке комплексне перетворення виглядає дуже привабливо і відразу дає користувачеві зрозуміти, що відбувається зміна режимів.

Sequence

Існує особливий тег <mx:Sequence/>, що дозволяє визначити послідовність ефектів. Подібно тегу Parallel він включає в себе набір ефектів, однак відображатися вони будуть по порядку (а не одночасно, як у попередньому прикладі). Для прикладу можна замінити тег <mx:Parallel/> на <mx:Sequence/> в попередньому коді.

Типові ефекти та їх властивості

Крім duration всі ефекти володіють спеціальними властивостями, що регулюють їх відображення. Такі властивості визначають початкове і кінцеве значення об'єкта трансформації, дозволяючи розробнику отримувати різні варіації одного і того ж ефекту. Приміром, для ефекту Resize можна задати початкове і кінцеве значення властивостей width і height, а для ефекту Move – властивостей x та y.

Даним властивостями не обов'язково привласнювати будь-які значення, оскільки Flex автоматично визначає їх залежно від змін відповідних властивостей компонента. Наприклад, якщо змінювати значення властивості x панелі

Panel з 0 на 100 і додати до неї ефект Move, властивості xFrom і xTo приймуть відповідні значення 0 і 100 автоматично.

Якщо значення властивостей ефекту не задані явно і Flex не може визначити їх значення з властивостей компонента, то використовуються налаштування ефекту за замовчуванням.

Нижче представлена інформація про типові ефекти та їх найбільш важливих властивостях.

Blur

Ефект Blur розмиває обриси зображення, як за відсутності фокусування.

Його дію можна налаштувати за допомогою наступних властивостей:

blurXFrom

Визначає початкову ступінь розмиття по горизонталі.

blurXTo

Визначає кінцеву ступінь розмиття по горизонталі.

blurYFrom

Визначає початкову ступінь розмиття по вертикалі.

blurYTo

Визначає кінцеву ступінь розмиття по вертикалі.

Даний ефект можна дуже вдало використовувати для імітації руху. Його можна застосовувати спільно з ефектом Move. З допомогою дуже сильного розмиття (зі значенням більше 20) можна домогтися враження трансформації.

Dissolve

Ефект Dissolve регулює властивість alpha прямокутного накладення, що створює враження поступової прояви або зникнення компонента. Його дію можна налаштувати за допомогою наступних властивостей:

alphaFrom

Визначає первинне значення властивості alpha.

alphaTo

Визначає кінцеве значення властивості alpha.

color

Визначає колір прямокутника, накладеного за цільовим компонентом. За замовчуванням воно збігається з кольором фону даного компоненту (встановленого за допомогою властивості backgroundColor), тому, як правило, не вимагає ніяких змін. Якщо значенняbackgroundColor компоненту не задано, використовується білий колір (0xFFFFFF).

Dissolve може замінити ефект Fade при використанні шрифтів пристрою.

Fade

Даний ефект впливає на властивість alpha компонента, змінюючи його прозорість.

Його дію можна налаштувати за допомогою наступних властивостей:

alphaFrom

Визначає первинне значення властивості alpha.

alphaTo

Визначає кінцеве значення властивості alpha.

Glow

Даний ефект використовується для підсвічування компонента, тобто створює враження вихідного від нього світла.

Його дію можна налаштувати за допомогою наступних властивостей:

alphaFrom

Визначає первинне значення властивості alpha.

alphaTo

Визначає кінцеве значення властивості alpha.

blurXFrom

Визначає початкову ступінь горизонтального розмиття підсвічування.

blurXTo

Визначає кінцеву ступінь горизонтального розмиття підсвічування.

blurYFrom

Визначає початкову ступінь вертикального розмиття підсвічування.

blurYTo

Визначає кінцеву ступінь вертикального розмиття підсвічування.

color

Визначає колір підсвічування.

inner

Обирає одну з двох можливостей - внутрішню або зовнішню підсвітку. За замовчуванням використовується значення false, відповідне зовнішньої підсвічуванні.

knockout

Задає режим «прозорого вікна», при якому колір самого об’єкта стане більш прозорим і крізь нього буде проступати колір фону компоненту. Значення за замовчуванням – false.

Iris

Даний ефект здійснює анімацію у вигляді прямокутної рамки, щорозширюється або звужується, подібно до діафрагми фотоапарата, з відцентруванням на об’єкті. Дана рамка розкриває або, навпаки, приховує вміст компонента, на який вона накладена.

Дія даного ефекту можна налаштувати за допомогою наступного властивості:

showTarget

Дозволяє визначити, чи повинен бути відображений компонент (true) або прихований (false, настройка за замовчуванням).

Move

Даний ефект дозволяє здійснити поступове переміщення компоненту.

Його дію можна налаштувати за допомогою наступних властивостей:

xFrom

Визначає початкове положення компонента по горизонтальній осі.

xTo

Визначає кінцеве положення компонента по горизонтальній осі.

yFrom

Визначає початкове положення компонента по вертикальній осі.

yTo

Визначає кінцеве положення компонента по вертикальній осі.

xBy

Визначає кількість пікселів, на яку потрібно зрушити компонент по горизонталі. Цю властивість можна використовувати замість xFrom або xTo, щоб визначити відстань, на яку зміщується компонент щодо початкового або кінцевого значення координати x. Якщо задані значення обох властивостей xFrom і xTo, значення даної властивості ігнорується.

yBy

Визначає кількість пікселів, на яку потрібно зрушити компонент по вертикалі. Цю властивість можна використовувати замість yFrom або yTo, щоб визначити відстань, на яку зміщується компонент щодо початкового або кінцевого значення координати y. Якщо задані

значення обох властивостей yFrom і yTo, значення даної властивості ігнорується.

Resize

Змінює значення ширини й висоти компонента у відведений часовий проміжок.

Його дію можна налаштувати за допомогою наступних властивостей:

widthFrom

Визначає початкову ширину компонента.

widthTo

Визначає кінцеву ширину компонента.

heightFrom

Визначає початкову висоту компонента.

heightTo

Визначає кінцеву висоту компонента.

widthBy

Встановлює потрібну зміну ширини компонента в пікселях. Дану властивість можна використовувати замість властивості widthFrom або widthTo, щоб визначити, на скільки потрібно змінити заданий початковий або кінцевий параметр розміру. Якщо задані значення обох властивостей widthFrom і widthTo, значення даної властивості ігнорується.

heightBy

Встановлює потрібну зміна висоти компонента в пікселях. Дану властивість можна використовувати замість

властивості heightFrom або heightTo, щоб визначити, на скільки потрібно змінити заданий початковий або кінцевий параметр розміру. Якщо задані значення обох властивостей heightFrom і heightTo, значення даної властивості ігнорується.

hideChildrenTargets

Дану властивість можна використовувати при застосуванні ефекту Resize по відношенню до контейнера Panel. Вона дозволяє приховати його вміст в процесі зміни розмірів. Приймає значення масиву панелей Panel. Коли від зміни розміру компонента, до якого застосовується ефект Resize, залежать розміри інших компонентів, даний ефект поширюється і на них. З такою ситуацією можна зіткнутися при використанні компонентами обмежувачів, що залежать від розміру основного компонента, або при відносному розташуванні компонентів програми.

Rotate

Даний ефект змінює положення компонента шляхом його повороту щодо певної точки (за замовчуванням, лівого верхнього кута додатку). Точку можна встановити самостійно; нею може бути, наприклад, центр компонента. Крім того, користувачем задаються значення початкового та кінцевого кутів повороту в межах 360 градусів. Якщо дане значення перевищить допустиме, воно буде прирівняно до максимально можливого – 360°.

Дію ефекту можна налаштувати за допомогою наступних властивостей:

angleFrom

Визначає початковий кут повороту.

angleTo

Визначає кінцевий кут повороту.

originX

Використовується для визначення точки, щодо якої здійснюється поворот, встановлюючи її положення по горизонталі щодо цільового компонента. За замовчуванням значення дорівнює 0.

originY

Використовується для визначення точки, щодо якої здійснюється поворот, встановлюючи її положення по вертикалі щодо цільового компонента. За замовчуванням значення дорівнює 0.

WipeLeft, WipeRight, WipeUp і WipeDown

Ці ефекти регулюють видимість компонента, відображаючи або приховуючи їх вміст. Їх використання подібно невидимому прямокутнику, переміщується над компонентом.

Дію даних ефектів можна налаштувати за допомогою наступної властивості:

showTarget

Дозволяє визначити, чи повинен вміст компонента бути відображеним (true) або прихованим (false, настройка за замовчуванням).

Zoom

Збільшує або зменшує компонент подібно об’єктиву фотоапарата. Даний ефект дозволяє здійснювати масштабування, створюючи враження, що компонент знаходиться дуже далеко або, навпаки, дуже близько.

Дії ефекту можна налаштувати за допомогою наступних властивостей:

zoomHeightFrom

Визначає початковий масштаб компонента по вертикалі. За замовчуванням приймається значення 1. Значення 2 збільшить в двічі компоненту, масштабуючи його відповідним чином, 3 –збільшить втричі і т. п.

zoomHeightTo

Визначає кінцевий масштаб компонента по вертикалі.

zoomWidthFrom

Визначає початковий масштаб компонента по горизонталі.

zoomWidthTo

Визначає кінцевий масштаб компонента по горизонталі.

originX

Використовується для визначення точки, щодо якої здійснюється масштабування, встановлюючи її положення по горизонталі щодо цільового компонента. За замовчуванням нею є центр компонента.

originY

Використовується для визначення точки, щодо якої здійснюється масштабування, встановлюючи її положення по вертикалі щодо цільового компонента. За замовчуванням нею також є центр компонента.

AnimateProperty

Цей ефект повністю визначається настройками користувача. Він дозволяє анімувати, тобто здійснювати перехід від одного числового значення конкретної властивості компонента до іншого.

У розпорядженні у розробника є наступні властивості для налаштування:

property

Визначає ім’я змінюваного властивості (приймає значення типу String).

fromValue

Визначає початкове значення властивості.

toValue

Визначає кінцеве значення властивості.

Наприклад, для створення ефекту, що нагадує Resize, але по відношенню до самої лише висоті компонента Panel, можназадати значення height властивості property даного ефекту, а також необхідні значення fromValue і toValue.

Література

1. Джои Лотт, Дерон Шалл, Кейт Питерс ActionScript 3.0 Сборник рецептов 2008
2. Аларик Коул Изучаем Flex 3 Руководство по разработке насыщеных интернет приложений 2009
3. Джошуа Ноубл, Тодд Андерсон Flex 3 Сборник рецептов 2009
4. Джои Лотт Flash Сборник рецептов 2007