helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

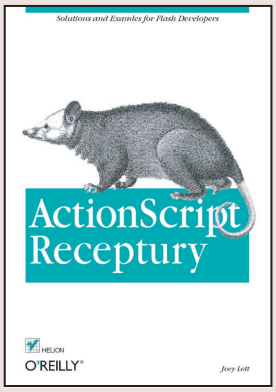
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

ActionScript. Receptury



Autor: Joey Lott
Tłumaczenie: Marzena Baranowska (rozdz. 1 – 11),
Anna Witerkowska (rozdz. 12 – 22),
Łukasz Zieliński (rozdz. 23 – 28)
ISBN: 83-7361-331-5
Tytuł oryginału: ActionScript Cookbook
Format: B5, stron: 872

Zamiast koncentrować się na abstrakcyjnych niuansach języka ActionScript, można skorzystać z gotowych sposobów rozwiązywania typowych problemów. Receptury ułatwiają pisanie skryptów w języku ActionScript, ucząc przy tym praktycznych technik, które na pewno będą przydatne jeszcze wiele razy.

Ogrom możliwości języka ActionScript może przytłaczać. W niniejszej książce analizujemy zaawansowane technologie jako zespoły prostych czynności, z których każda ma praktyczne znaczenie i z których każdą warto zrozumieć. Książka „ActionScript. Receptury" jest interesująca zarówno dla młodego kodera, jak i dla doświadczonego programisty, ponieważ pozwala spojrzeć z nowej perspektywy i podejść w nowy sposób do zagadnień programowania w języku ActionScript, jednocześnie umożliwiając ugruntowanie zdobytych już umiejętności.

Książka „ActionScript. Receptury" zawiera ponad 300 receptur związanych z niezliczoną ilością tematów, wśród których znajdują się m.in.:

- rysowanie kształtów za pomocą instrukcji skryptów;
- programistyczne sterowanie odtwarzaniem klipów filmowych;
- obsługa wprowadzania tekstu i operacje na łańcuchach znaków;
- udostępnianie strumieni audio i wideo dzięki technologii Flash Communication Server;
- wykorzystanie technologii Flash Remoting do tworzenia połączeń z zapleczem bazodanowym;
- posługiwanie się zestawami rekordów i macierzami danych;
- wiele, wiele innych rozwiązań w 20 pełnych receptur rozdziałach.

Na początku książki omówione są krótkie, proste receptury. Później, stopniowo, pojawiają się dłuższe i bardziej skomplikowane skrypty wykonujące coraz bardziej wymyślne zadania. Taki układ wiedzy umożliwia łączenie poznanych skryptów w rozwiązania, na których można oprzeć swoje aplikacje utworzone we Flashu. Miła niespodzianka czeka i tego, kto nad jednostkowe receptury wyżej ceni duże aplikacje: w książce opisano siedem kompletnych, złożonych projektów we Flashu.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Wstęp *17*

Część I Receptury lokalne *23*

Rozdział 1. Podstawy ActionScript *25*

1.0. Wprowadzenie 25
1.1. Użycie operatorów matematycznych 30
1.2. Sprawdzanie równości lub porównywanie wartości 32
1.3. Warunkowe wykonywanie instrukcji 35
1.4. Sprawdzanie kilku warunków 41
1.5. Wielokrotne powtarzanie operacji 42
1.6. Wykonanie akcji jednorazowo w każdej klatce 46
1.7. Powtarzanie zadania w określonych interwałach czasu 47
1.8. Tworzenie kodu do wielokrotnego użycia 49
1.9. Uogólnianie funkcji pod kątem wielostronnej używalności 52
1.10. Wychodzenie z funkcji 54
1.11. Uzyskiwanie wyniku z funkcji 55
1.12. Unikanie konfliktu zmiennych 56
1.13. Ponowne użycie i organizacja kodu w kilku filmach 58

Rozdział 2. Środowisko wykonawcze *61*

2.0. Wprowadzenie 61
2.1. Uzyskiwanie informacji o wersji Flash Playera 61
2.2. Uzyskiwanie informacji o systemie operacyjnym 65
2.3. Uzyskiwanie informacji o języku systemowym 66

2.4. Uzyskiwanie informacji o ustawieniach ekranu 68
2.5. Skalowanie filmu 69
2.6. Modyfikacja wyrównania 71
2.7. Uzyskiwanie informacji o właściwościach urządzenia dźwiękowego 72
2.8. Skłonienie użytkownika do zmiany ustawień Playera 74
2.9. Ukrywanie pozycji menu aplikacji Flash Player 75
2.10. Udoskonalanie aplikacji Standalone Projector 76
2.11. Określanie rozmiarów projektora 78
2.12. Określanie miejsca wyświetlania projektora 79

Rozdział 3. Kolor 81

3.0. Wprowadzenie 81
3.1. Ustawianie koloru klipu filmowego 82
3.2. Określanie wartości RGB 84
3.3. Dekodowanie wartości RGB 85
3.4. Ustawianie wartości RGB względem bieżącej wartości 86
3.5. Zabarwianie klipu filmowego 87
3.6. Ustawianie przezroczystości klipu filmowego 89
3.7. Przekształcanie bieżącego koloru klipu filmowego 90
3.8. Przywracanie oryginalnego koloru klipu filmowego 91
3.9. Sterowanie kolorem klipu filmowego za pomocą suwaków 92

Rozdział 4. Rysowanie i maskowanie 95

4.0. Wprowadzenie 95
4.1. Rysowanie prostej 96
4.2. Rysowanie krzywej 98
4.3. Rysowanie prostokąta 99
4.4. Rysowanie prostokąta z zaokrąglonymi narożnikami 100
4.5. Rysowanie okręgu 105
4.6. Rysowanie elipsy 107
4.7. Rysowanie trójkąta 109
4.8. Rysowanie wieloboków foremnych 111
4.9. Wypełnianie kształtu jednolitym lub półprzezroczystym kolorem 113
4.10. Tworzenie kształtu z wypełnieniem gradientowym 114
4.11. Tworzenie kształtu ze złożonym wypełnieniem gradientowym 117
4.12. Tworzenie masek za pomocą skryptów 120

Rozdział 5. Liczby i matematyka ***123***

5.0. Wprowadzenie 123
5.1. Reprezentowanie liczb w różnych systemach liczbowych 124
5.2. Przekształcenia między różnymi systemami liczbowymi 125
5.3. Zaokrąglanie liczb 127
5.4. Wstawianie zer wiodących i końcowych 130
5.5. Formatowanie liczb wyświetlanych 132
5.6. Formatowanie kwot pieniężnych 134
5.7. Generowanie liczby losowej 136
5.8. Symulacja rzutu monetą 138
5.9. Symulacja rzutu kostką do gry 141
5.10. Symulacja gry w karty 143
5.11. Generowanie liczby jednoznacznej 148
5.12. Przekształcanie wartości kątów 149
5.13. Obliczanie odległości między dwoma punktami 150
5.14. Określanie punktów na okręgu 152
5.15. Przekształcenia jednostek miar 155
5.16. Obliczanie wzrostu wartości środków finansowych (wartość przyszła) 157
5.17. Obliczenie oszczędności emerytalnych 160
5.18. Obliczenie wysokości kredytu (zastaw hipoteczny) 162
5.19. Obliczenie amortyzacji pożyczki lub opłat rocznych 163

Rozdział 6. Tablice ***165***

6.0. Wprowadzenie 165
6.1. Dołączanie elementów na początek lub koniec tablicy 166
6.2. Przetwarzanie tabeli w pętli 168
6.3. Wyszukiwanie elementów w tablicy 169
6.4. Usuwanie elementów 171
6.5. Wstawienie elementów do środka tablicy 172
6.6. Przekształcanie łańcucha na tablicę 173
6.7. Przekształcanie tablicy na łańcuch 174
6.8. Tworzenie osobnej kopii tablicy 175
6.9. Przechowywanie danych złożonych i wielowymiarowych 177
6.10. Sortowanie lub odwracanie tablicy 179
6.11. Realizacja sortowania niestandardowego 181
6.12. Tworzenie tablicy asocjacyjnej 184
6.13. Odczytywanie elementów z tablicy asocjacyjnej 186

Rozdział 7. Klipy filmowe 189
7.0. Wprowadzenie 189
7.1. Odwołania do klipów filmowych przez ActionScript 191
7.2. Odwoływanie się do klipów filmowych z dynamicznymi nazwami 191
7.3. Sterowanie odtwarzaniem 193
7.4. Zmiana kierunku odtwarzania 195
7.5. Użycie klipów filmowych jako przycisków 196
7.6. Definiowanie obszarów aktywnych w klipach filmowych 199
7.7. Sprawdzanie położenia wskaźnika myszy 200
7.8. Wielokrotne wykonywanie akcji w klipach filmowych 203
7.9. Zanikanie klipu filmowego 205
7.10. Odkrywanie zagnieżdżonych klipów filmowych 208
7.11. Uzyskiwanie unikatowej głębokości 210
7.12. Określanie granic klipu filmowego 212
7.13. Tworzenie klipów filmowych możliwych do przeciągania 214
7.14. Tworzenie własnego wskaźnika myszy 216
7.15. Sprawdzanie, czy klipy filmowe na siebie zachodzą (wykonywanie testów kolizji) 219
7.16. Zmiana kolejności klipów filmowych w stosie 226
7.17. Przekształcenia między układami współrzędnych 229
7.18. Powielanie klipów filmowych 230
7.19. Dodawanie klipów filmowych z biblioteki z poziomu ActionScript 231

Rozdział 8. Tekst 237
8.0. Wprowadzenie 237
8.1. Odwoływanie się do pola tekstowego za pomocą języka ActionScript 238
8.2. Tworzenie pola tekstowego 239
8.3. Tworzenie obrysu pola tekstowego 241
8.4. Tworzenie tła pola tekstowego 241
8.5. Tworzenie pola tekstowego z możliwością wprowadzania danych przez użytkownika 242
8.6. Tworzenie pola służącego do wprowadzania hasła 244
8.7. Filtrowanie wprowadzanego tekstu 245
8.8. Ustawienie maksymalnej długości pola 246
8.9. Dynamiczne wyświetlanie tekstu w czasie odtwarzania filmu 247
8.10. Wyświetlanie tekstu sformatowanego za pomocą języka HTML 248
8.11. Kondensowanie odstępów 251
8.12. Zmiana rozmiaru pól tekstowych pod kątem dopasowania ich do zawartości 251
8.13. Przewijanie tekstu za pomocą komponentu ScrollBar 254

8.14. Przewijanie tekstu za pomocą skryptu 255
8.15. Odpowiadanie na zdarzenia przewijania 258
8.16. Formatowanie istniejącego tekstu 260
8.17. Formatowanie tekstu wprowadzonego przez użytkownika 261
8.18. Formatowanie fragmentu zawartości pola tekstowego 262
8.19. Określanie czcionki w polu tekstowym 263
8.20. Osadzanie czcionek 264
8.21. Tworzenie tekstu, który można obrócić 266
8.22. Wyświetlanie tekstu w standardzie Unicode 266
8.23. Ustawianie zaznaczenia (fokusu) na polu tekstowym 268
8.24. Zaznaczanie tekstu z poziomu ActionScript 269
8.25. Ustawianie punktu wstawiania w polu tekstowym 269
8.26. Działania po zaznaczeniu i usunięciu zaznaczenia tekstu 270
8.27. Reagowanie na tekst wprowadzony przez użytkownika 271
8.28. Dodawanie łącza do tekstu 272

Rozdział 9. Łańcuchy tekstowe 275

9.0. Wprowadzenie 275
9.1. Łączenie łańcuchów 275
9.2. Użycie cudzysłowów i apostrofów w łańcuchach 278
9.3. Wstawianie specjalnych znaków odstępów 279
9.4. Wyszukiwanie podłańcucha 280
9.5. Wydobywanie podłańcucha 285
9.6. Porównywanie z wzorcem przy użyciu wyrażeń regularnych 287
9.7. Wyszukiwanie łańcuchów określonym wzorcem 292
9.8. Rozbiór łańcucha na wyrazy 293
9.9. Usuwanie i zastępowanie znaków 296
9.10. Przetwarzanie do pojedynczych znaków 298
9.11. Zmiana wielkości liter 300
9.12. Eliminowanie odstępów 301
9.13. Odwracanie łańcucha — zmiana kolejności wyrazów lub liter 303
9.14. Przekształcanie łańcucha między formatami Unicode i ASCII 304

Rozdział 10. Data i czas 307

10.0. Wprowadzenie 307
10.1. Uzyskiwanie informacji o bieżącej dacie i czasie 307
10.2. Uzyskiwanie informacji o nazwie dnia lub miesiąca 310
10.3. Formatowanie daty i czasu 311

10.4. Formatowanie milisekund jako minut i sekund 314
10.5. Przekształcenia między formatem DMYHMSM a milisekundami epoki 315
10.6. Obliczanie czasu minionego lub interwałów pomiędzy datami 317
10.7. Wydobycie daty z łańcucha 322
10.8. Tworzenie stoperów i zegarów 325

Rozdział 11. Formularze 329
11.0. Wprowadzenie 329
11.1. Dodawanie komponentów interfejsu użytkownika w czasie odtwarzania 330
11.2. Rozmieszczanie elementów formularza 331
11.3. Dodawanie menu do formularza 331
11.4. Tworzenie menu zależnych 333
11.5. Zmiana rozmiaru menu w celu dopasowania go do zawartości 335
11.6. Wykrywanie wybranych pozycji menu 337
11.7. Dodawanie grupy przycisków opcji 338
11.8. Automatyczne wyrównywanie przycisków opcji 341
11.9. Uzyskiwanie wartości wybranego przycisku opcji 344
11.10. Dodawanie pól wyboru do formularza 345
11.11. Uzyskiwanie wartości pól opcji 347
11.12. Tworzenie złożonego formularza 348
11.13. Wysyłanie formularza 349
11.14. Sprawdzanie poprawności informacji wprowadzonych do formularza 352
11.15. Ostrzeganie użytkowników o błędach walidacji 355
11.16. Tworzenie formularza wielostronicowego 357
11.17. Wysyłanie formularza wielostronicowego 361
11.18. Sprawdzanie poprawności wypełnienia formularza wielostronicowego 362
11.19. Bezpieczne przesyłanie danych 363
11.20. Wstępne wypełnienie formularza 367
11.21. Zmiana kolejności elementów formularza 370
11.22. Użycie tabel do rozmieszczenia elementów formularza 372
11.23. Tworzenie automatycznie wypełnianych pól tekstowych 384
11.24. Dostosowywanie wyglądu komponentu 388
11.25. Dostosowywanie wyglądu wszystkich komponentów 391

Rozdział 12. Obiekty i komponenty własne 393
12.0. Wprowadzenie 393
12.1. Wykorzystanie metod i właściwości obiektów wbudowanych 395
12.2. Tworzenie kopii klasy 396

12.3. Dodawanie właściwości do kopii obiektu ... 398
12.4. Dodawanie własnych metod do kopii obiektu ... 400
12.5. Tworzenie własnej klasy ... 403
12.6. Tworzenie właściwości pobierz-ustaw ... 404
12.7. Definiowanie właściwości tylko do odczytu ... 406
12.8. Tworzenie klasy podrzędnej ... 408
12.9. Implementacja metod klasy nadrzędnej w klasie podrzędnej ... 410
12.10. Wykrywanie zdarzeń ... 412
12.11. Dodawanie odbiorców do własnych klas ... 414
12.12. Komponent umożliwiający tworzenie klas podrzędnych klipu filmowego ... 419
12.13. Program: komponent selektora kolorów ... 423

Rozdział 13. Programowanie dźwięków ... 427

13.0. Wprowadzenie ... 427
13.1. Tworzenie obiektu do regulacji odtwarzania dźwięku ... 428
13.2. Dynamiczne dołączanie dźwięków ... 431
13.3. Uruchamianie i zatrzymywanie odtwarzania dźwięku ... 432
13.4. Pobieranie czasu odtwarzania ... 433
13.5. Zapętlanie dźwięków ... 434
13.6. Definiowanie punktu początku i końca odtwarzania ... 435
13.7. Wstrzymywanie i wznawianie odtwarzania ... 438
13.8. Wykonywanie akcji po zakończeniu odtwarzania dźwięku ... 440
13.9. Odtwarzanie dźwięków po kolei ... 441
13.10. Dodawanie dźwięków do przycisków i komponentów interfejsu użytkownika ... 442
13.11. Ustawienia głośności dźwięku ... 445
13.12. Sterowanie panoramą dźwięku ... 446
13.13. Tworzenie zaawansowanych efektów stereo ... 448
13.14. Efekt narastania dźwięku ... 449
13.15. Efekt zanikania dźwięku ... 452
13.16. Program: komponent regulatora odtwarzania dźwięku ... 455

Część II Receptury zdalne ... 467

Rozdział 14. Serwer FlashCom ... 469

14.0. Wprowadzenie ... 469
14.1. Tworzenie nowej aplikacji FlashCom ... 470
14.2. Łączenie się z serwerem FlashCom ... 471
14.3. Dynamiczne dodawanie obiektu wideo ... 474

14.4. Przechwytywanie materiału wideo z kamery w sieci i wyświetlanie go 475
14.5. Przechwytywanie i odtwarzanie dźwięku odbieranego przez mikrofon 478
14.6. Sterowanie odtwarzaniem strumieni audio FlashCom 480
14.7. Korzystanie z zawartości audio-wideo 481
14.8. Tworzenie listy odtwarzania 483
14.9. Nagrywanie i publikacja zawartości wideo i audio 484
14.10. Publikowanie zawartości „na żywo" 486
14.11. Wstrzymywanie i wznawianie odtwarzania strumienia sieciowego 487
14.12. Przewijanie zawartości strumienia sieciowego 488
14.13. Przeszukiwanie względem całkowitej długości strumienia 489
14.14. Implementacja serwerowych skryptów ActionScript 492
14.15. Śledzenie użytkowników połączonych z aplikacją 494
14.16. Wywołanie serwerowej funkcji z filmu klienta 495
14.17. Wywołanie klienckiej funkcji z serwera 496

Rozdział 15. Ładowanie zasobów 499

15.0. Wprowadzenie 499
15.1. Ładowanie zewnętrznych plików SWF 500
15.2. Ładowanie zewnętrznych plików SWF z zaufanej domeny 502
15.3. Ładowanie zewnętrznych plików JPEG 503
15.4. Ładowanie zewnętrznych plików graficznych (we wszystkich formatach) 505
15.5. Ładowanie zewnętrznych plików MP3 509
15.6. Ładowanie zawartości poprzez serwer proxy 510
15.7. Ustalanie, czy zasób został załadowany 514
15.8. Ustalanie stanu załadowania zasobu 516
15.9. Monitorowanie ładowania z użyciem komponentu Progress Bar 518
15.10. Monitorowanie ładowania bez komponentu Progress Bar 519
15.11. Wykonywanie akcji po zakończeniu ładowania zasobu 522
15.12. Ukrywanie grafiki i tekstu paska postępu ładowania 523

Rozdział 16. Przechowywanie informacji trwałych 525

16.0. Wprowadzenie 525
16.1. Lokalne zapisywanie i odczytywanie danych trwałych 526
16.2. Dodawanie danych do klienckiego obiektu współdzielonego 527
16.3. Czytanie danych z klienckiego obiektu współdzielonego 530
16.4. Zapisywanie lokalnego obiektu współdzielonego 530
16.5. Współdzielenie informacji pomiędzy filmami w tej samej domenie 532

16.6. Przechowywanie trwałych danych na serwerze 535
16.7. Zapisywanie danych zdalnego obiektu współdzielonego 537
16.8. Sprawdzanie, czy wykonywane są aktualizacje danych odległego obiektu współdzielonego 539
16.9. Odczytywanie wartości z serwerowego obiektu współdzielonego 541
16.10. Dodawanie danych do serwerowego obiektu współdzielonego 541

Rozdział 17. Komunikacja między filmami 543

17.0. Wprowadzenie 543
17.1. Komunikacja między filmami znajdującymi się na tym samym komputerze 545
17.2. Przesyłanie danych poprzez połączenie lokalne 547
17.3. Potwierdzanie odbioru w komunikacji przez połączenie lokalne 549
17.4. Akceptacja połączeń z innych domen 551
17.5. Komunikacja między filmami znajdującymi się na różnych komputerach 553
17.6. Przesyłanie danych do klientów zdalnych obiektów współdzielonych 554

Rozdział 18. Wysyłanie i ładowanie zmiennych 557

18.0. Wprowadzenie 557
18.1. Ładowanie zmiennych z pliku tekstowego 558
18.2. Ładowanie zmiennych ze skryptu uruchomionego po stronie serwera 561
18.3. Sprawdzanie postępu ładowania danych 563
18.4. Wysyłanie danych do skryptu serwerowego 564
18.5. Wysyłanie zmiennych i obsługa zwróconego wyniku 566

Rozdział 19. XML 569

19.0. Wprowadzenie 569
19.1. Struktura XML (odczytywanie i zapisywanie XML) 572
19.2. Tworzenie obiektu XML 574
19.3. Dodawanie elementów do obiektu XML 576
19.4. Dodawanie węzłów tekstowych do obiektu XML 577
19.5. Tworzenie obiektu XML z tablicy 579
19.6. Dodawanie atrybutów do elementu XML 581
19.7. Odczytywanie elementów z drzewa XML 583
19.8. Odszukiwanie elementów na podstawie ich nazwy 585
19.9. Odczytywanie węzłów tekstowych i ich wartości 586
19.10. Odczytywanie atrybutów elementu 587
19.11. Wczytywanie XML 588
19.12. Usuwanie niepotrzebnych odstępów z obiektu XML 590
19.13. Przesyłanie danych XML 592

19.14. Wysyłanie danych XML i odbieranie odpowiedzi 593
19.15. Przeszukiwanie obiektu XML 598
19.16. Użycie danych XML do inicjalizacji filmu....601

Rozdział 20. Flash Remoting 603

20.0. Wprowadzenie....603
20.1. Ustanawianie połączenia przez Flash Remoting....605
20.2. Konfiguracja Flash Remoting dla platformy ColdFusion....608
20.3. Konfiguracja Flash Remoting dla platformy .NET 608
20.4. Konfiguracja Flash Remoting dla platformy J2EE 610
20.5. Konfiguracja Flash Remoting dla języków PHP lub Perl 612
20.6. Wywołanie zdalnej funkcji usługowej 612
20.7. Obsługa odpowiedzi z Flash Remoting....614
20.8. Rozróżnienie wyników otrzymywanych z wielu odwołań do jednej usługi 616
20.9. Wywoływanie z Flasha funkcji ASP.NET 617
20.10. Wywoływanie z Flasha funkcji ColdFusion 619
20.11. Przekazywanie nazwanych parametrów do metod komponentu ColdFusion....621
20.12. Przekazywanie złożonych parametrów do metod komponentu ColdFusion....622
20.13. Wywoływanie z Flasha funkcji Javy lub JSP 623
20.14. Transmisja własnych typów danych do zaplecza Flash Remoting....624
20.15. Odbieranie obiektów typu w ColdFusion....625
20.16. Odbieranie obiektów typu w ASP.NET....626
20.17. Odbieranie obiektów typu w Javie....627
20.18. Zwracanie obiektów typu z ColdFusion 629
20.19. Zwracanie obiektów typu z ASP.NET 630
20.20. Zwracanie obiektów typu z Javy 632
20.21. Pisanie funkcji serwerowych w ActionScript 633
20.22. Zapytania do baz danych w serwerowych skryptach ActionScript 634
20.23. Zapytania HTTP w serwerowych skryptach ActionScript....636
20.24. Korzystanie z usług WWW poprzez Flash Remoting dla platformy .NET lub ColdFusion 639

Rozdział 21. Zestawy rekordów 641

21.0. Wprowadzenie....641
21.1. Tworzenie zestawu rekordów....641
21.2. Odczytywanie zestawu rekordów....643
21.3. Filtrowanie zestawu rekordów 644

21.4. Sortowanie zestawu rekordów według wybranej kolumny 647
21.5. Wypełnianie danymi komponentów menu 648
21.6. Wyświetlanie rekordów na siatce danych 651

Część III Aplikacje 655

Rozdział 22. Tworzenie aplikacji graficznej Flash Paint 657

Planowanie aplikacji 658
Budowanie komponentów 658
Złożenie aplikacji Flash Paint 679
Korzystanie z aplikacji 688

Rozdział 23. Prosta animacja tworzona etapami 689

Etap pierwszy 690
Etap drugi 692
Etap trzeci 695
Etap czwarty 697
Etap piąty 703
Podsumowanie 706

Rozdział 24. Centrum teleczatu i wiadomości wideo 707

Omówienie procesu tworzenia aplikacji 707
Tworzenie aplikacji serwerowej 708
Tworzenie klienta dostępowego 715
Tworzenie klienta administracyjnego 719
Uruchomienie gotowego systemu 725

Rozdział 25. Przeglądarka grafiki z funkcją pokazu slajdów 727

Plan aplikacji 728
Początek pracy 728
Tworzenie komponentów 729
Złożenie kompletnej aplikacji 766
Podsumowanie 767

Rozdział 26. Sieciowy system odtwarzania plików MP3 769

Projekt aplikacji 769
Budowanie narzędzi wyboru plików 770
Budowanie aplikacji odtwarzacza 785
Podsumowanie 797

Rozdział 27. Aplikacja „mojej" strony spersonalizowanej 799
Formułowanie planu aplikacji 800
Tworzenie aplikacji szkieletu 801
Tworzenie modułów usługowych 816
Uruchamianie kompletnej aplikacji 829
Rozbudowa aplikacji 829
Podsumowanie 839

Rozdział 28. Terminarz 841
Projektowanie struktury aplikacji 841
Przygotowanie komponentów 842
Uruchomienie kompletnej aplikacji 862
Przygotowanie aplikacji do udostępnienia w internecie 864
Wykorzystanie obiektu LoadVars i formatu XML 866
Podsumowanie 869

Dodatki 871

Skorowidz 873

10

Data i czas

10.0. Wprowadzenie

Data i czas są ważne w wielu aplikacjach ActionScript, szczególnie w tych najbardziej rozbudowanych, oferujących usługi dla użytkowników. Wartości daty i czasu są istotne przy określaniu długości czasu, jaki upływa w operacjach czasowych, przy określaniu ważności wersji próbnych, przechowywaniu dat transakcji itd.

ActionScript przechowuje dane o dacie i czasie w *milisekundach epoki*, które są liczbą milisekund, jakie upłynęły od początku tzw. epoki Uniksa, a mianowicie od 1 stycznia 1970 roku (według Coordinated Universal Time — UTC). Dla celów tej książki przyjmiemy, że UTC jest odpowiednikiem czasu Greenwich Mean Time (GMT). (Pod adresem *http://aa.usno.navy.mil/faq/docs/UT.html* znajduje się więcej informacji o różnicach między jednym a drugim czasem). Wiele języków programowania przechowuje daty i czasy, korzystając z terminologii epoki (często w sekundach zamiast milisekund); dlatego łatwo można pracować z wartościami dat i czasów zaimportowanymi z innych źródeł (i odwrotnie).

Ponadto klasa *Date* — przy wykorzystaniu takich metod, jak *getFulYear()*, *setFullYear()*, *getMonth()* i *setMonth()* — umożliwia ustawienie i uzyskanie informacji o wartościach daty i czasu określanych przy użyciu lat, miesięcy, dni itd. Metody te zostały zaprojektowane dla wygody twórców, ale wartości daty i czasu zawsze są przechowywane wewnętrznie jako milisekundy epoki.

10.1. Uzyskiwanie informacji o bieżącej dacie i czasie

Problem

Chcemy wiedzieć, jaka jest aktualna data i czas.

Rozwiązanie

Należy utworzyć nową wartość daty za pomocą konstruktora *Date()*, nie określając żadnych parametrów. Można także użyć skryptu CGI lub napisanego w innym języku skryptu działającego po stronie serwera, aby uzyskać informację o dacie i z niej utworzyć nowy obiekt *Date*.

Analiza

Data i czas, jakie w języku ActionScript są szacowane, są oparte na ustawieniach daty i czasu komputera klienta. Dlatego jeśli w komputerze użytkownika jest ustawiony nieprawidłowy czas, taki sam czas będzie również w klipie Flasha. Pamiętając o tym przekłamaniu, można uzyskać aktualny czas i datę po stronie klienta, tworząc obiekt *Date* za pomocą konstruktora *Date()* bez parametrów w następujący sposób:

```
// Utworzenie nowego obiektu Date.
today = new Date();

// Wyświetlenie daty i czasu komputera klienta
trace(today);
```



Należy unikać używania nazwy `date` dla własnej tworzonej zmiennej. W przeciwieństwie do języka JavaScript w identyfikatorach języka ActionScript (włącznie z nazwami zmiennych i klas) nie są rozróżniane małe i wielkie litery. `Date` jest nazwą funkcji konstruktora klasy *Date*, dlatego nazwanie zmiennej `date` spowoduje wykluczenie konstruktora *Date*.

Jeśli dostępne jest aktywne połączenie internetowe, klip Flasha może uzyskać datę i czas z serwera. Technika ta gwarantuje dokładniejsze wskazania zegara. Choć ustawienia czasu serwera mogą nie być dokładne, czas będzie ten sam dla wszystkich klipów klienta.

Podstawowy proces odczytywania czasu z serwera polega na wykonaniu poniższych czynności:

1. Utwórz skrypt CGI na serwerze WWW, którego efektem działania będzie para nazwy i wartości, w której wartość jest liczbą sekund, jakie upłynęły od 1 stycznia 1970 roku.
2. W klipie Flasha użyj obiektu *LoadVars* (lub metody *loadVariables()*), aby pobrać sekundy epoki.
3. Przekształć pobrane sekundy z łańcucha na liczbę, mnożąc je przez 1000 i skonstruuj nowy obiekt *Date* przez przekazanie wartości do konstruktora *Date()*.

PHP jest językiem skryptowym, wykorzystywanym na wielu serwerach WWW. Łatwo jest utworzyć stronę PHP z rezultatem aktualnej daty i czasu, podawanymi jako liczba sekund od początku epoki. Wszystko, czego potrzeba, to dokument PHP z poniższym wierszem kodu, który należy wczytać na serwerze:

```
now=<?php echo time();?>
```

Jeśli na serwerze nie jest zainstalowane PHP lub wygodniej jest użyć języka Perl (jeden z bardziej uniwersalnych języków, często dostępny na serwerach WWW), powyższy kod można utworzyć właśnie w tym języku:

```
#!/usr/local/bin/perl
print "Content-type:text/plain\n\n";
print "now=";
print time;
```

Podczas instalacji skryptu na serwerze należy pamiętać o kilku niżej wymienionych sprawach.

- Pierwszy wiersz wskazuje miejsce, w którym znajduje się interpreter języka Perl. Wartość podana w przykładzie jest dość uniwersalna, choć zawsze można sprawdzić te informacje u administratora serwera WWW.
- Na wielu serwerach zdalne wykonywanie skryptów jest zablokowane w przypadku, gdy skrypt nie ma wymaganego rozszerzenia nazwy. Rozszerzenie *.cgi* jest dozwolone, dlatego odpowiednią nazwą skryptu będzie np. *getDate.cgi*.
- Większość serwerów WWW ogranicza dostęp do skryptów CGI tylko do określonych folderów. Zwykle znajdują się one w folderze głównym konta (lub w głównym folderze stron WWW) i mają nazwę *cgi* lub *cgi-bin*. Należy upewnić się, czy skrypt został zapisany we właściwym folderze.
- Na serwerach z systemem UNIX skrypt CGI musi mieć ustawione uprawnienia o wartości 755. Większość programów FTP umożliwia taką zmianę uprawnień — można jej dokonać za pomocą wiersza kodu:

```
chmod 755 nazwa_pliku
```

W dokumencie Flasha należy wczytać wartości daty i czasu z serwera. Najlepiej jest to wykonać za pomocą obiektu *LoadVars*. Klasa ta została wprowadzona we Flash MX. W celu uzyskania zgodności z wcześniejszymi wersjami Flasha zamiast niej należy użyć metody *MovieClip.loadVariables()*.

W przypadku podania jednej wartości konstruktorowi *Date()* ActionScript zinterpretuje ją jako liczbę milisekund od początku epoki i utworzy nowy obiekt *Date*, odpowiadający tej wartości. Dlatego należy pomnożyć wartość zwróconą przez skrypt (będzie ona podana w sekundach) przez 1000.

```
// Utworzenie obiektu LoadVars.
lv = new LoadVars();

// Użycie metody load() do uruchomienia skryptu CGI na serwerze.
// W PHP powinna zostać wskazana odpowiednia strona, np. getDate.php,
// zamiast poniższej.
lv.load("http://www.person13.com/cgi-bin/getDate.cgi");

// Metoda onLoad() jest wywoływana automatycznie
// po zwróceniu odpowiedzi przez serwer.
lv.onLoad = function () {
```

```
    // Utworzenie obiektu Date przez przekazanie wyniku (sekundy * 1000)
    // do konstruktora Date(). Wartością zwróconą przez serwer jest
    // this.nameOfVariable — w tym przypadku this.now. Następnie
    // wykonywane jest przekształcenie wartości łańcucha
    // na liczbę — przed pomnożeniem jej przez 1000.
    var serverDate = new Date(Number(this.now) * 1000);

    // Wyświetlenie daty i czasu zwróconych przez serwer
    // (w tym przykładzie z przesunięciem strefowym właściwym
    // dla ustawień komputera użytkownika).
    trace(serverDate);
};
```

Data jest zawsze przechowywana w języku ActionScript jako liczba milisekund od początku epoki, ale zawsze jest wyświetlana z właściwym przesunięciem — przy uwzględnieniu lokalnej strefy czasowej użytkownika (o ile nie zostaną określone metody UTC). Dlatego też jeżeli na komputerze użytkownika ustawiona jest niewłaściwa strefa czasowa, wyświetlenie prawidłowego czasu może być niemożliwe. Jednak aktualna data (w milisekundach) nadal będzie prawidłowa.

Zobacz także

Jeśli czas serwera nie jest rzetelny lub nie odpowiada Twoim potrzebom, możesz skorzystać z jednego z wielu serwerów internetowych podających informacje o aktualnej dacie i czasie. Jednym z nich jest na przykład *http://tycho.usno.navy.mil*. Więcej informacji o synchronizacji czasu przy wykorzystaniu odpowiedniego protokołu (Network Time Protocol) znajduje się na stronie WWW pod adresem *http://www.ntp.org* oraz w recepturze 10.7.

10.2. Uzyskiwanie informacji o nazwie dnia lub miesiąca

Problem

Chcemy uzyskać informacje o nazwie dnia lub miesiąca.

Rozwiązanie

Należy utworzyć tablice zawierające wartości łańcucha dla nazw dni tygodnia i nazw miesięcy roku. Aby uzyskać wartości łańcucha z tablicy, należy użyć liczby dnia lub miesiąca.

Analiza

W klasie *Date* języka ActionScript dostępne są metody *getDay()* i *getMonth()*, które zwracają wartości liczb całkowitych reprezentujące dzień tygodnia (od 0 do 6) i miesiąc (od 0 do 11). Jednak zamiast liczb można uzyskać nazwę dnia lub miesiąca. W tym celu należy

utworzyć tablice zawierające nazwy dni i miesięcy. Najlepiej jest to wykonać przez zdefiniowanie tych tablic jako właściwości klasy *Date* i zapisanie informacji w pliku *Date.as* w celu wykorzystania tych danych w późniejszych projektach:

```
// Utworzenie days i months jako właściwości klasy Date.
Date.days = ["Niedziela", "Poniedziałek", "Wtorek", "Środa", "Czwartek",
            "Piątek", "Sobota"];
Date.months = ["Styczeń", "Luty", "Marzec", "Kwiecień", "Maj", "Czerwiec",
                  "Lipiec", "Sierpień", "Wrzesień", "Październik", "Listopad",
                   "Grudzień"];

// Utworzenie obiektu Date dla niedzieli, 1 grudnia 2002.
myDate = new Date(2002, 11, 1);

// Wyświetlenie: Niedziela
trace(Date.days[myDate.getDay()]);

// Wyświetlenie: Grudzień
trace(Date.months [myDate.getMonth()]);
```

Zobacz także

Recepturę 10.3.

10.3. Formatowanie daty i czasu

Problem

Chcemy wyświetlić sformatowane wartości daty i czasu.

Rozwiązanie

Należy użyć metody *Date.toString()* lub utworzyć własną metodę *Date.format()*, która będzie zwracać datę i czas jako łańcuch w żądanym formacie.

Analiza

Metoda *Date.toString()* zwraca wartość łańcucha obiektu *Date* w zapisie tradycyjnym, jak na przykład:

```
// Wyświetlenie: Mon May 26 11:32:46 GMT-0400 2003-10-30
trace(new Date().toString());
```

Ponieważ ActionScript automatycznie wywołuje metodę *toString()* w odniesieniu do obiektu użytego w zawartości łańcucha, ten sam rezultat można uzyskać nawet wtedy, jeśli metoda *toString()* zostanie pomięta, jak w poniższym kodzie:

```
// Wyświetlenie: Mon May 26 11:32:46 GMT-0400 2003-10-30
trace(new Date());
```

Metodę *Date.toString()* można zmodyfikować tak, aby uzyskać inny format daty:

```
Date.prototype.toString = function () {
  return "Milisekundy od epoki: " + this.getTime();
};
// Obydwa wiersze pozwalają wyświetlić tekst:
// Milisekundy od epoki: 1053963542360
trace(new Date().toString());
trace(new Date());
```

Jednak lepszym rozwiązaniem jest implementacja własnej metody formatowania, przyjmującej parametr określający żądany format daty. Na szczęście istnieje standardowa implementacja formatowania dat w takich językach, jak Java, którą można naśladować. Tabela 10.1 zawiera zestawienie symboli, które można wykorzystać w tworzeniu formatowanego łańcucha, który jest następnie przekazywany do metody własnej *format()*.

Tabela 10.1. Symbole daty i czasu

Symbol	Znaczenie	Przykład
`Y`	Rok	2002
`M`	Miesiąc	Grudzień lub 12 (w zależności od kontekstu: MM oznacza 12; mm oznacza grudzień)
`D`	Dzień	01
`H`	Godzina przed południem lub po południu	12
`H`	Godzina (0 – 23)	21
`M`	Minuta	24
`s`	Sekunda	33
`S`	Milisekunda	960
`E`	Dzień tygodnia	Sobota lub Sob (w zależności od kontekstu: E oznacza Sobota; EEE oznacza Sob)
`a`	A.M lub P.M. (przed południem lub po południu)	AM lub PM

Metoda własna *format()* powinna przyjmować jako parametr łańcuch wskazujący format daty i czasu. Należy użyć instrukcji *switch*, aby uzyskać wartość w żądanym formacie. Tablice `Date.days` i `Date.months` (patrz receptura 10.2) muszą być dostępne tak, aby kod poniższego przykładu działał pomyślnie. (Należy włączyć ten kod do pliku *Date.as* wraz z tablicami `Date.days` i `Date.months`).

```
Date.days = ["Niedziela", "Poniedziałek", "Wtorek", "Środa", "Czwartek",
             "Piątek", "Sobota"];
Date.months = ["Styczeń", "Luty", "Marzec", "Kwiecień", "Maj", "Czerwiec",
               "Lipiec", "Sierpień", "Wrzesień", "Październik",
               "Listopad","Grudzień"];

// Ta metoda własna przekształca wartość liczbową na dwucyfrowy łańcuch,
// tak aby wartości jednocyfrowe były prawidłowo sformatowane
// (np. 5 zostanie przekształcone na 05).
```

```
Date.toTens = function (val) {
  if(val < 10) {
    return "0" + val;
  }
  else {
    return String(val);
  }
};

// Przekształcenie czasu na łańcuch. Zegar 24-godzinny
// (w którym godziny są określane od 0 do 23) jest przekształcany
// na 12-godzinny ze wskazaniem AM/PM.
Date.toTwelveHour = function (hour, min) {
  var amPm = "AM";
  if(hour > 12) {
    hour = hour - 12;
    amPm = "PM";
  }
  if(hour == 0) {
    hour = 12;
  }
  return Date.toTens(hour) + ":" + Date.toTens(min) + " " + amPm;
}

Date.prototype.format = function (format) {

  // Utworzenie zmiennych lokalnych z wartościami dat.
  var day       = this.getDay();
  var monthDate = this.getDate();
  var month     = this.getMonth();
  var year      = this.getFullYear();
  var hour      = this.getHours();
  var min       = this.getMinutes();
  var sec       = this.getSeconds();
  var millis    = this.getMilliseconds();

  // Zwrócenie łańcucha z datą i czasem w żądanym formacie, np. "MM-dd-rrrr".
  // Można dopisać więcej formatów, korzystając z zestawienia w tabeli 10.1.
  switch (format) {
    case "MM-dd-yyyy":
      return Date.toTens(month + 1) + "-" + Date.toTens(monthDate) + "-" + year;
    case "MM/dd/yyyy":
      return Date.toTens(month + 1) + "/" + Date.toTens(monthDate) + "/" + year;
    case "dd-MM-yyyy":
      return Date.toTens(monthDate) + "-" + Date.toTens(month + 1) + "-" + year;
    case "dd/MM/yyyy":
      return Date.toTens(monthDate) + "/" + Date.toTens(month + 1) + "/" + year;
    case "hh:mm a":
      return Date.toTwelveHour(hour, min);
    case "EEE, MMM dd, yyyy":
      return Date.days[day].substr(0, 3) + ", " + Date.months[month] + " " +
             Date.toTens(monthDate) + ", " + year;
    case "E, MMM dd, yyyy":
      return Date.days[day] + ", " + Date.months[month] + " " +
             Date.toTens(monthDate) + ", " + year;
  }
};
```

Oto kilka przykładowych zastosować metody *format()*:

```
// Utworzenie daty dla niedzieli 1 grudnia 2002 roku. Czas - 18:09.
myDate = new Date(2002, 11, 1, 18, 9);

trace(myDate.format("MM-dd-yyyy"));    // Wyświetlane jest: 12-01-2002
```

```
trace(myDate.format("dd/mm/yyyy"));    // Wyświetlane jest: 12/01/2002
trace(myDate.format("dd-MM-yyyy"));    // Wyświetlane jest: 01-12-2002
trace(myDate.format("dd/MM/yyyy"));    // Wyświetlane jest: 01/12/2002
trace(myDate.format("hh:mm a"));       // Wyświetlane jest: 06:09 PM
// Wyświetlane jest: Nie, Grudzień 01, 2002
trace(myDate.format("EEE, MMM dd, yyyy"));
// Wyświetlane jest: Niedziela, Grudzień 01, 2002 06:09 PM
trace(myDate.format("EEE, MMM dd, yyyy") + " " + myDate.format("hh:mm a"));
```

Zobacz także

Metodę *format()* można udoskonalić, dodając więcej przypadków do instrukcji *switch*. Więcej informacji można znaleźć pod adresem *http://www.php.net/manual/en/function.date.php* i w recepturze 10.4.

10.4. Formatowanie milisekund jako minut i sekund

Problem

Chcemy wyświetlić milisekundy w formacie minut i sekund (*mm:ss*).

Rozwiązanie

Należy utworzyć własną metodę statyczną *Date.formatMilliseconds()*.

Analiza

W języku ActionScript wiele wartości, na przykład długość dźwięku, jest podawanych w milisekundach. Ale w większości sytuacji chcemy, aby wartości były podawane w minutach i sekundach — szczególnie te przeznaczone do wyświetlania. Można to uzyskać za pomocą metody własnej, dodanej do klasy *Date*.

Poniżej przedstawiona została właśnie ta metoda własna — statyczna, wywoływana bezpośrednio z klasy, nie z kopii obiektu — przekształcająca milisekundy na format *mm:ss*. Warto zapisać ją w pliku *Date.as* w celu wykorzystania jej w późniejszych projektach:

```
Date.formatMilliseconds = function (millis) {

  // Określenie minut i sekund.
  vars seconds = millis / 1000;
  var m = Math.floor(seconds/60);
  var s = Math.round(seconds- (m * 60));

  // Dodanie zer wiodących do wskazań jednocyfrowych.
  if (m < 10) {
```

```
    m = "0" + m;
  }
  if (s < 10) {
    s = "0" + s;
  }
  return m + ":" + s;
};
```

Oto przykład zastosowania metody:

```
#include Date.as

// Przyjęto założenie, że mySound_sound jest obiektem Sound,
// podającym czas trwania w milisekundach.
  dur = mySound_sound.duration;

// Utworzenie pola tekstowego do wyświetlenia wartości.
this.createTextField("displayTime_txt", 1, 0, 0, 100, 20);

// Wyświetlenie wartości czasu sformatowanej jako mm:ss.
displayTime_txt.text = Date.formatMilliseconds(dur);
```

Zobacz także

Receptury 10.3 i 10.5.

10.5. Przekształcenia między formatem DMYHMSM a milisekundami epoki

Problem

Chcemy przekształcić format DMYHMSM (dni, miesiące, lata, godziny, minuty, sekundy, milisekundy) na milisekundy epoki.

Rozwiązanie

Należy użyć metod *getTime()* i *setTime()*.

Analiza

Większości z nas najłatwiej pomyśleć o datach i godzinach jak o takich danych, jak godziny, dni i lata, niż o milisekundach epoki czy sekundach. Wygodniej jest ludziom rozmawiać o dacie 3.55 rano, w piątek 13 października 1978 roku niż o odpowiadającej jej wartości 277 124 100 000. Jednak w takich językach, jak ActionScript czas przechowywany jest w formacie milisekund (lub sekund) epoki. Dlatego ważne jest, by poradzić sobie z przekształceniem różnych formatów, szczególnie gdy data lub czas mają być wyświetlane lub gdy współużytkujemy dane między aplikacjami z różnymi formatami.

W momencie tworzenia daty w języku ActionScript można użyć rozwiązania DMYHMSM:

```
// Konstruowanie daty dla piątku 13 października 1978 roku, godziny 3:55 rano.
myDate = new Date(1978, 9, 13, 3, 55, 0, 0);
```

ActionScript automatycznie przekształci i zapisze datę w formacie odpowiednim dla milisekund epoki. Aby uzyskać tę wartość, należy wywołać metodę *getTime()* z obiektu *Date*:

```
// Dla Pacific Standard Time wyświetlana jest wartość 277 124 100 000.
// Rezultat może być inny, w zależności od strefy czasowej.
trace(myDate.getTime());
```

Wartość sekund epoki zwróconą przez metodę *getTime()* można przekazać do innej aplikacji (np. do skryptu CGI) lub użyć jej do wykonania obliczeń daty (patrz receptura 10.6).

Z drugiej strony można ustawić datę za pomocą milisekund epoki. Na przykład w recepturze 10.1 skrypt CGI zwraca do Flasha aktualny czas serwera, podany w milisekundach epoki (milisekundy są następnie przekształcane poprzez pomnożenie ich przez 1000). Ponadto przy szacowaniu daty można ustawić ją według milisekund epoki. Można to wykonać na dwa sposoby. Pierwszy polega na przekazaniu liczby milisekund do konstruktora *Date* jako jedynego parametru, a drugi — na przekazaniu liczby milisekund do wartości *setTime()* istniejącej daty. Obydwa sposoby przynoszą te same rezultaty. Zaletą użycia metody *setTime()* jest to, że do istniejącej daty można przydzielić własne właściwości, które mogą być utracone przy wywołaniu konstruktora.

```
// Konstruowanie obiektu new Date dla godziny 3.55 rano w piątek,
// 13 października 1978 roku. W przykładzie użyto wartości wyświetlonej
// w oknie Output z poprzedniego przykładu.

// Wyświetlane jest: Fri Oct 13 03:55:00 GMT-0700 1978
// (przesunięcie strefy czasowej może być różne)
trace(myDate);

// Utworzenie obiektu Date.
myDate = new Date();

// Przydzielenie właściwości własnej do obiektu Date.
myDate.label = "To specjalny dzień!";

// Użycie metody setTime() do ustawienia daty na Fri Oct 13 03:55:00 rano 1978.
// Właściwość label nie jest zastępowana, jak w przypadku
// wywołania konstruktora Date.
myDate.setTime(277124100000);

// Wyświetlane jest: Fri Oct 13 03:55:00 GMT-0700 1978
// (przesunięcie strefy czasowej może być różne)
trace(myDate);
```

Zobacz także

Po przydzieleniu wartości do daty za pomocą metody *setTime()* można użyć metod *getDate()*, *getMonth()*, *getYear()*, *getHours()*, *getSeconds()*, *getMinutes()* i *getMilliseconds()*, aby uzyskać indywidualne wartości DMYHMSM w czasie lokalnym. Aby uzyskać datę i czas

UTC, zamiast powyższych metod należy użyć metod *getUTCDate()*, *getUTCMonth()*, *getUTCYear()*, *getUTCHours()*, *getUTCSeconds()*, *getUTCMinutes()* i *getUTCMilliseconds()*. Zobacz recepturę 10.6.

10.6. Obliczanie czasu minionego lub interwałów pomiędzy datami

Problem

Chcemy oszacować miniony czas, datę lub czas względny.

Rozwiązanie

Aby oszacować miniony czas, należy dodać i odjąć od milisekund epoki wartość zwróconą przez *getTimer()*. W bardziej złożonych przekształceniach należy utworzyć metody własne *Date.doMath()* i *Date.elapsedTime()*.

Analiza

W przypadku prostego przekształcenia — w rodzaju dodania do daty lub odjęcia od niej liczby godzin, dni lub tygodni — należy po prostu dodać lub odjąć wartość milisekund epoki. W naszych rozważaniach należy zwrócić uwagę, że sekunda to 1 000 milisekund, minuta to 60 000 milisekund, a godzina to 3 600 000 milisekund itd. Jeśli nie masz spektakularnej zdolności zapamiętania tych wartości, najlepiej zapisać je w klasie *Date*. Dla wygody poniższe stałe można dodać do pliku *Date.as* .

```
// Sekunda ma 1 000 milisekund, minuta ma 60 sekund, godzina ma 60 minut,
// dzień ma 24 godziny, a tydzień ma 7 dni.
Date.SEC = 1000;
Date.MIN = Date.SEC * 60;
Date.HOUR = Date.MIN * 60;
Date.DAY = Date.HOUR * 24;
Date.WEEK = Date.DAY * 7;
```

Metody *Date.getTime()* można użyć do uzyskania informacji o wartości aktualnej daty w milisekundach epoki, a za pomocą metody *Date.setTime()* można ustawić nową wartość. Poniższy przykład służy dodaniu jednego dnia do danego obiektu *Date*.

```
#include "Date.as"
myDate = newDate(1978, 9, 13, 3, 55, 0, 0);

// Wyświetlane jest: Fri Oct 13 03:55:00 GMT-0700 1978
trace(myDate);

// Dodanie jednego dnia do poprzedniej daty przez ustawienie nowej daty i czasu
// w oryginalnej parze data/czas i Date.DAY (liczba milisekund dnia).
```

```
myDate.setTime(myDate.getTime() + Date.DAY);

// Wyświetlane jest: Sat Oct 14 03:55:00 GMT-0700 1978
trace(myDate);
```

Poniżej przedstawione zostały bardziej złożone przykłady zastosowania metod *Date.getTime()*, *Date.setTime()* i wcześniej wspomnianych stałych:

```
#include "Date.as"
myDate = newDate(1978, 9, 13, 3, 55, 0, 0);

// Odjęcie jednego tygodnia od daty.
// Wyświetlane jest: Fri Oct 6 03:55:00 GMT-0700 1978
myDate.setTime(myDate.getTime() - Date.WEEK);
trace(myDate);

// Dodanie jednego tygodnia i jednego dnia do daty.
// Wyświetlane jest: Sat Oct 14 03:55:00 GMT-0700 1978
myDate.setTime(myDate.getTime() + Date.WEEK + Date.DAY);
trace(myDate);

// Odjęcie 3 godzin i 55 minut od daty.
// Wyświetlane jest: Sat Oct 14 00:00:00 GMT-0700 1978
myDate.setTime(myDate.getTime() - (3 * Date.HOUR) - (55 * Date.MIN));
trace(myDate);
```

Często chcemy obliczać miniony czas, aby utworzyć stoper w grach lub innych aplikacjach. Zadanie to polega jedynie na zapisaniu czasu rozpoczęcia, a następnie porównaniu czasu zakończenia z aktualnym. Aby na przykład obliczyć, jak długo film ma być uruchomiony, można użyć poniższego kodu (proszę zwrócić uwagę, że konstruktor *new Date()* zawsze zwraca czas aktualny):

```
// Zapisanie czasu początkowego.
var startingTime = new Date();
// Utworzenie pola tekstowego do wyświetlania czasu aktualnego.
this.createTextField("timer_txt", 1, 100, 100, 50, 20);

// Sprawdzanie czasu minionego w trakcie każdego
// wyświetlenia klatki.
this.onEnterFrame = function () {
  // Określenie czasu minionego.
  var elapsedTime = new Date().getTime() - startingTime.getTime();
  // Przekształcenie milisekund na sekundy
  // i zaokrąglenie do najbliższej sekundy.
  this.timer_txt.text = Math.round(elapsedTime / 1000);
};
```

Funkcja globalna *getTimer()* zwraca liczbę milisekund od początku uruchomienia Playera — dlatego przez sprawdzanie jej wartości w kolejnych okresach czasu można określić czas miniony. Alternatywą do poprzedniego przykładu jest użycie funkcji globalnej *getTimer()* w miejsce *new Date()*:

```
var startingTime = getTimer();
this.createTextField("timer_txt", 1, 100, 100, 50, 20);
this.onEnterFrame = function () {
  var elapsedTime = getTimer() - startingTime;
  // Przekształcenie milisekund na sekundy
  // i zaokrąglenie do najbliższej sekundy.
  this.timer_txt.text = Math.round(elapsedTime / 1000);
};
```

Należy zauważyć, że poniższy kod nie jest niezbędną alternatywą poprzedniego. Funkcja globalna *getTimer()* zwraca liczbę milisekund od początku uruchomienia Playera, a nie od momentu uruchomienia klipu. W tym przykładzie stoper nie jest zerowany w momencie wczytania nowego klipu:

```
this.createTextField("timer_txt", 1, 100, 100, 50, 20);
this.onEnterFrame = function () {
  // Przekształcenie milisekund na sekundy
  // i zaokrąglenie do najbliższej sekundy.
  this.timer_txt.text = Math.round(getTimer()/ 1000);
};
```

Powyższy przykład można zaadaptować przy tworzeniu stopera z odliczaniem wstecz. Ten przykład prezentuje przejście do klatki o nazwie „OutOfTime" po 60 sekundach:

```
// Zapisanie czasu początkowego.
var startingTime = getTimer();
// Utworzenie pola tekstowego do wyświetlania czasu aktualnego.
this.createTextField("timer_txt", 1, 100, 100, 50, 20);
// Odliczanie wstecz od 60 sekund.
var maxTime = 60;

// Sprawdzanie czasu minionego w trakcie każdego
// wyświetlenia klatki.
this.onEnterFrame = function () {
  // Określenie czasu minionego.
  var elapsedTime = getTimer() - startingTime;
  // Przekształcenie milisekund na sekundy
  // i zaokrąglenie do najbliższej sekundy.
  elapsedTime = Math.round(elapsedTime / 1000);
  if (elaplsedTime >= maxTime) {
    this.timer_txt.text = "0";
    gotoAndPlay("OutOfTime");
  } else {
    this.timer_txt.text = maxTime - elapsedTime;
  }
};
```

Wróćmy do wcześniejszego przykładu, w którym czas miniony obliczany był za pomocą obiektów *Date*. Gdy trzeba dodać lub odjąć lata i miesiące od dat, nie można polegać na stałych, ponieważ liczba milisekund w miesiącu różni się od liczby dni w miesiącu, a lata przestępne mają więcej milisekund niż inne lata. Jednak klasa *Date* obsługuje obliczenia w momencie użycia metod pobierającej i ustawiającej. Najlepszym sposobem jest utworzenie metody *Date.doMath()*, przeznaczonej do wykonania obliczeń. Metoda może przyjąć do siedmiu parametrów numerycznych, ujemnych lub dodatnich:

years
: Liczba lat do dodania do daty.

months
: Liczba miesięcy do dodania do daty.

days
: Liczba dni do dodania do daty.

hours
: Liczba godzin do dodania do daty.

minutes
Liczba minut do dodania do daty.

seconds
Liczba sekund do dodania do daty.

milliseconds
Liczba milisekund do dodania do daty.

Przykład metody własnej *Date.doMath()*, który można dodać do swojego pliku *Date.as*, jest następujący:

```
Date.prototype.doMath = function (years, months, days, hours, minutes,
                                  seconds, milliseconds) {

  // Wykonanie przekształcenia na kopii oryginalnej daty,
  // tak aby nie zmieniać oryginalnej.
  var d = new Date(this.getTime());

  // Dodanie określonych interwałów do oryginalnej daty.
  d.setYear(d.getFullYear() + years);
  d.setMonth(d.getMonth() + months);
  d.setDate(d.getDate() + days);
  d.setHours(d.getHours() + hours);
  d.setMinutes(d.getMinutes() + minutes);
  d.setSeconds(d.getSeconds() + seconds);
  d.setMilliseconds(d.getMilliseconds() + milliseconds);

  // Zwrócenie nowej wartości daty.
  return d;
};

// Przykład zastosowania:
myDate = new Date(1978, 9, 13, 3, 55, 0, 0);

// Dodanie jednego roku, 3 godzin i 55 minut.
// Wyświetlane jest: Sat Oct 13 00:00:00 GMT-0700 1979
trace(myDate.doMath(1, 0, 0, -3, -55));

// Można także dodać 365 dni i odjąć 235 minut, aby uzyskać
// tę samą datę, co wyżej. Wyświetlane jest: Sat Oct 13 00:00:00 GMT-0700 1979
trace(myDate.doMath(0, 0, 365, 0, -235));

// Dodanie 24 lat.
// Wyświetlane jest: Sat Oct 13 03:55:00 GMT-0700 2002
trace(myDate.doMath(24));
```

W poprzednim przykładzie zademonstrowano sposób utworzenia obiektu *new Date* z istniejącego obiektu *Date* w oparciu o czas miniony. Jednak można obliczyć czas miniony między dwoma istniejącymi obiektami *Date*, co wcale nie jest zbyt trywialne. Można odjąć wartość zwróconą z metody *getTime()* jednego z obiektów *Date* od drugiego obiektu, ale nie pozwala to na obliczenie czasu minionego między dwoma obiektami *Date*. Chociaż operacja zwraca liczbę milisekund między dwiema datami, rezultat nie jest wygodny do obsługi, gdy czasy dotyczą tego samego dnia. Ręczne przekształcenie liczby milisekund na liczbę lat, miesięcy i dni jest trudne z uwagi na różną liczbę dni w miesiącach, lata przestępne itd. Co więcej, obsługa ujemnych czasów minionych może być trudna.

Dlatego najłatwiej jest utworzyć metodę własną *Date.elapsedTime()*, która zwraca obiekt *Date* reprezentujący czas miniony. Pozwala to użyć wbudowanych metod klasy *Date* do obliczenia liczby lat, miesięcy i dni między dwoma obiektami *Date*. Jest tu jednak kilka utrudnień. Chociaż ActionScript przechowuje daty, zapisując je jako względne wobec początku epoki (północ 1 stycznia 1970), większość metod klasy *Date* zwraca wartości bezwzględne, a nie względne. Na przykład rok dla daty w 1971 zostanie zwrócony jako 1971, a nie jako 1. Aby obiekt czasu minionego był rzeczywiście przydatny, należy zdefiniować metody własne *elapsedYears()* i *elapsedDays()*, które będą zwracały wartości względne do czasu epoki. (Miesiąc, godzina, minuta, sekunda i milisekunda czasu epoki wynoszą 0, zatem nie ma potrzeby tworzenia metody własnej do zwracania przesunięć względem tych wartości).

Zaprezentowana poniżej metoda własna *elapsedYears()* tworzy obiekt *Date*, reprezentujący czas miniony między obiektem *Date*, z którego została wywołana, a drugim obiektem *Date*. Jeśli nie zostanie określony drugi obiekt *Date*, czas miniony jest obliczany względem czasu aktualnego (teraz). Czas miniony jest zawsze przechowywany jako liczba dodatnia — bez względu na to, który z obiektów *Date* przedstawia czas późniejszy. Metody *elapsedTime()*, *elapsedYears()* i *elapsedDays()* należy dodać do pliku *Date.as*, w celu późniejszego wykorzystania ich w projektach.

```
// Obliczenie czasu minionego (jako wartości bezwzględnej)
// między obiektem Date a określoną datą.
Date.prototype.elapsedTime = function (t1) {
  // Obliczenie czasu minionego względem określonego obiektu Date.
  if (t1 == undefined) {
    // Obliczenie czasu minionego od "teraz",
    // w przypadku nieokreślenia drugiego obiektu.
    t1 = new Date();
  } else {
    // Utworzenie kopii, aby nie zmieniać oryginału.
    t1 = new Date(t1.getTime());
  }
  // Użycie czasu oryginalnego obiektu Date jako końcowego.
  // Utworzenie kopii w celu zachowania oryginału.
  var t2 = new Date(this.getTime());

  // Zapewnienie, że czas miniony zawsze będzie obliczany jako wartość dodatnia.
  if (t1 < t2) {
   temp = t1;
   t1 = t2;
   t2 = temp;
  }
  // Zwrócenie czasu minionego jako nowego obiektu Date.
  var t = new Date(t1.getTime() - t2.getTime());
  return t;
};
```

Metody *elapsedYears()* i *elapsedDays()* zwracają rok i dzień względem czasu epoki.

```
Date.prototype.elapsedYears = function () {
  return this.getUTCFullYear() - 1970;
};

Date.prototype.elapsedDays = function () {
  return this.getUTCDate() - 1;
};
```

Oto kilka przykładów użycia metod własnych. Warto zauważyć, że wbudowane metody klasy *Date* mogą być użyte do uzyskania informacji o minionych miesiącach, godzinach, minutach, sekundach i milisekundach, a metody własne *elapsedYears()* i *elapsedDays()* mogą być użyte do uzyskania informacji o minionych latach i dniach.

```
#include "Date.as"
// Obliczenie wieku osoby w oparciu o datę urodzin.
birthdate = new Date(1966, 3, 17);
age = birthdate.elapsedTime();
trace("Minione lata "        + age.elapsedYears());
trace("Minione miesiące "    + age.getUTCMonth());
trace("Minione dni "         + age.elapsedDays());
trace("Minione godziny "     + age.getUTCGodziny());
trace("Minione minuty "      + age.getUTCMinutes());
trace("Minione sekundy "     + age.getUTCSeconds());
trace("Minione milisekundy " + age.getUTCMilliseconds());

// Obliczenie interwału między dwiema datami.
firstDay = new Date(1901, 0, 1, 0, 0, 0, 0);
lastDay = new Date(2000, 11, 31, 23, 59, 999);
century = firstDay.elapsedTime(lastDay);
trace("Minione lata "        + century.elapsedYears());
trace("Minione miesiące "    + century.getUTCMonth());
trace("Minione dni "         + century.elapsedDays());
trace("Minione godziny "     + century.getUTCGodziny());
trace("Minione minuty "      + century.getUTCMinutes());
trace("Minione sekundy "     + century.getUTCSeconds());
trace("Minione milisekundy " + century.getUTCMilliseconds());
```

Naturalnie istnieją inne sposoby implementacji funkcjonalności daty i czasu. Wybraliśmy najłatwiejsze w celu zilustrowania samej idei. Bardziej złożonym rozwiązaniem jest implementacja klasy *elapsedDate* jako podklasy klasy *Date*. Klasa własna może zastąpić metody wbudowane *getUTCFullYear()*, *getUTCDate()*, *getYear()*, *setUTCDate()*, *setUTCFullYear()* i *setYear()* i pozwolić na uzyskanie wartości oraz ustawienie ich względem epoki, a nie bezwzględnie. Takie implementacje pozostawiamy jako ćwiczenie dla Czytelnika.

Zobacz także

Recepturę 10.8.

10.7. Wydobycie daty z łańcucha

Problem

Chcemy utworzyć obiekt *Date* z łańcucha.

Rozwiązanie

Należy utworzyć własną metodę *parseDate()*, która zanalizuje składniki określonej daty (rok, miesiącitd.) z łańcucha używającego wyrażeń regularnych, a następnie złoży nową datę.

Analiza

ActionScript nie dostarcza swoich własnych metod do analizy łańcucha w obiekcie *Date*. Ograniczenie to nie stanowi jednak problemu. Zwrócenie obiektów *Date* z innych aplikacji umożliwia na przykład aplikacja Flash Remoting. Nawet, jeśli nie pracujemy z Flash Remoting, możemy wymieniać wartości pomiędzy Flashem a innymi aplikacjami z użyciem milisekund i sekund epoki. Jednak gdy istnieje konieczność analizy łańcucha daty, należy użyć metody własnej *parseDate()*. Przyjmuje ona wartość łańcucha jako parametr, analizuje poszczególne składniki daty (rok, godzinę itd.), a następnie zwraca nowy obiekt *Date*. Poniższy kod można dodać do pliku *Date.as* w celu wykorzystania go w późniejszych projektach. W poniższym kodzie wykorzystana została klasa *RegExp* — zobacz recepturę 9.6.

```
// Należy włączyć plik RegExp.as z receptury 9.6.
// Plik można dodać do pliku Date.as, w którym znajduje się metoda parseDate().
#include "RegExp.as"

// W tej metodzie wykorzystywana jest również tablica Date.months
// z receptury 10.2.
Date.parseDate = function (dateStr) {

  // Utworzenie zmiennych lokalnych do przechowywania roku, miesiąca,
  // dnia miesiąca, godzin, minut i sekund. Przyjęto założenie,
  // że w łańcuchu daty nie ma milisekund.
  var year, month, monthDate, hour, minute, second;

  // Użycie wyrażenia regularnego do sprawdzenia, czy w łańcuchu
  // używany jest format łańcucha daty zgodny z językiem ActionScript.
  // Na przykład: Thu Dec 5 06:36:03 GMT-0800 2002.
  var re = new RegExp(
  "[a-zA-Z]{3} [a-zA-Z]{3} [0-9]{1,2} [0-9]{2}:[0-9]{2} .* [0-9]{4}", "g");
  var match = re.exec(dateStr);

  // Jeśli łańcuch daty jest zgodny ze wzorcem, następuje analiza daty
  // i zwrócenie nowego obiektu Date z uzyskaną wartością.
  if (match != null) {

    // Rozdzielenie dopasowania na tablicę łańcuchów.
    // Rozdzielenie na obszary będące dniem, miesiącem, czasem,
    // strefą czasową i rokiem.
    var dateAr = match[0].split(" ");

    // Ustawienie miesiąca na drugi element tablicy. Jest to skrócona nazwa
    // miesiąca, ponieważ chcemy uzyskać numer miesiąca (od 0 do 11), zatem
    // przetworzenie w pętli obejmie tablicę Date.months i będzie trwało
    // do momentu znalezienia szukanego elementu. Później na ten indeks
    // zostanie ustawiony miesiąc.
    month = dateAr[1];
    for (var i = 0; i < Date.months.length; i++) {
```

```
      if (Date.months[i].indexOf(month) != -1) {
        month = i;
        break;
      }
    }

    // Przekształcenie monthDate i year z tablicy z łańcuchów na liczby.
    monthDate = Number(dateAr[2]);
    year = Number(dateAr[dateAr.length - 1]);

    // Pobranie składników z tablicy - godziny, minut i sekund.
    var timeVals = dateAr[3].split(":");
    hour = Number(timeVals[0]);
    minute = Number(timeVals[1]);
    second = Number(timeVals[2]);

    // Jeśli tablica ma sześć elementów, wliczone jest przesunięcie strefowe
    // (niektóre łańcuchy dat w tym formacie pomijają przesunięcie
    // stref czasowych).
    if (dateAr.length == 6) {
      var timezone = dateAr[4];

     // Zwielokrotnienie przesunięcia (w godzinach) o 60 w celu uzyskania minut.
     var offset = 60 * Number(timezone.substr(3, 5))/100;

     // Obliczenie strefy godzinnej między komputerem klienta
     // a przesunięciem uzyskanym z łańcucha daty.
     var offsetDiff = offset + new Date().getTimezoneOffset();

    // Dodanie przesunięcie strefy godzinowej w minutach.
    // Jeśli łańcuch daty i komputer klienta są w tej samej strefie,
    // różnica wyniesie 0.
    minute += offsetSiff;
}

      // Zwrócenie nowej daty.
      return new Date(year, month, monthDay, hour, minute, second);
      }

     // Jeśli łańcuch daty nie zgadza się ze standardowym formatem łańcucha
     // daty, następuje sprawdzenie, czy obejmuje on format MM-dd-yy(yy)
     // lub MM/dd/yy(yy).
     re = new RegExp("[0-9]{2}(/|-)[0-9]{2}(/|-)[0-9]{2,}", "g");
     match = re.exec(dateStr);
     if (match != null) {
       // Uzyskanie miesiąca, dnia i roku z dopasowania. Jako separator
       // został użyty ukośnik. Jeśli z tablicy zostanie zwrócony tylko
       // jeden element, zostanie użyty separator w postaci kreski.
       var mdy = match[0].split("/");
       if (mdy.length == 1) {
           mdy = match[0].split("-");
       }

        // Uzyskanie wartości numerów miesiąca i dnia miesiąca z łańcucha daty.
        month = Number(mdy[0]) - 1;
        monthDate = Number(mdy[1]);

        // Jeśli wartość roku jest dwuznakowa, należy dodać do niej wiek.
        if (mdy[2].length == 2) {
           twoDigitYear = Number(mdy[2]);
           // Przyjęto, że lata przed rokiem 50 należą do 21 wieku.
```

```
            year = (twoDigitYear < 50) ? twoDigitYear + 2000 : twoDigitYear +
            1900;
         } else {
            // Uzyskanie roku w zapisie czterocyfrowym.
            year = mdy[2];
         }
      }
// Sprawdzenie, czy łańcuch zawiera wartość czasu w postaci h(h):mm(:ss).
  re = new RegExp("[0-9]{1,2}:[0-9]{2}:[0-9]{2,})?", "g");
  match = re.exec(dateStr);
  if (match != null) {
    // Jeśli długość wynosi 4, czas jest podawany jako h:mm. Jeśli tak jest,
    // długość pierwszej części czasu (godziny) jest tylko jednym znakiem.
    // W przeciwnym razie są dwa znaki długości.
    var firstlength = 2;
    if (match[0].length ==4) {
      firstLength = 1;
    }

  // Uzyskanie informacji o części godzinowej i minutowej z łańcucha daty.
  // Jeśli długość jest większa niż pięć, ujęte zostaną także sekundy.
    hour = Number(dateStr.substr(match.index, first Length));
    minute = Number(dateStr.substr(match.index, first Length + 1, 2));
    if (match[0].length > 5) {
      second = Number(dateStr.substr(match.index + first:Length + 4, 2));
    }
  }

  // Zwrócenie nowej daty.
  return new Date(year, month, monthDate, hour, minute, second);
};
```

Oto kilka przykładów użycia metody *parseDate()* w celu utworzenia nowego obiektu *Date* z łańcucha. Podobnie, jak w innych przykładach zawartych w tym rozdziale, aktualne rezultaty zależą od ustawień strefy czasowej.

```
#include "Date.as"

// Wyświetlane jest: Sat Oct 9 00:00:00 GMT-0700 1982
trace(Date.parseDate("10/09/1982"));

// Wyświetlane jest: Fri Oct 13 03:55:00 GMT-0700 1978
trace(Date.parseDate("To była 3:55 10-13-78"));

// Wyświetlane jest: Tue Dec 3 06:36:02 GMT-0800 2002
trace(Date.parseDate("Fri Dec 2 14:36:02 GMT+0800 2002"));
```

10.8. Tworzenie stoperów i zegarów

Problem

Chcemy utworzyć stoper lub wykonywać operacje w określonych interwałach.

Rozwiązanie

Należy użyć funkcji *setInterval()*.

Analiza

W tej recepturze wyjaśnione zostały sposoby wykonania akcji przy ustawionych interwałach i tworzenie zegara pokazującego czas bezwzględny. W recepturze 10.6 znajduje się więcej informacji o tworzeniu stoperów wyświetlających czas miniony.

Funkcja *setInterval()* dodana we Flash MX pozwala ustawić stoper, który wywołuje funkcję lub metodę w określonych interwałach. Funkcja zwraca odwołanie do interwału tak, że istnieje możliwość anulowania akcji w przyszłości. Można dokonać wyboru spośród kilku odmian, w zależności od przeznaczenia funkcji *setInterval()*. Jeśli chcemy wywołać funkcję w określonym interwale bez podawania jej parametrów, można wywołać *setInterval()* z odwołaniem do funkcji i liczbą milisekund między wywołaniami funkcji.

```
// W tym przykładzie użyta jest metoda własna format(),
// zatem wymagany jest plik Date.as.
#include "Date.as"

// Ta funkcja jest wywoływana przez setInterval() i wyświetla aktualny czas.
function displayTime() {
  var d = new Date();
  trace(d.format("hh:mm a"));
}

// Ten kod jest przeznaczony do wywoływania displayTime
// co 60 000 milisekund (raz na minutę).
dtInterval = setInterval(displayTime, 60000);
```



Funkcja *setInterval()* wywołuje funkcję lub metodę w określonym interwale. Jest on zależny od procesora komputera klienta i nie jest dokładny ani niezmienny.

Funkcji *setInterval()* można również użyć do przekazania parametrów do wywoływanej funkcji. Podawane parametry są w takiej sytuacji wartościami przekazywanymi do funkcji wywołanej w określonym interwale. Należy pamiętać, że parametry wysłane do funkcji nie mogą być dynamicznie aktualizowane przy każdym wywołaniu. Funkcja *setInterval()* wywołuje drugą funkcję kilkakrotnie, ale sama jest wywołana tylko raz:

```
// Utworzenie funkcji wyświetlającej wartość do niej przekazaną.
function displayValue (val) {
  trace(val);
}

// Utworzenie interwału, przy którym funkcja displayValue()
// jest wywoływana co pięć sekund z podanym parametrem.
// Należy zwrócić uwagę, że choć parametrem jest Math.random()
// (metoda generująca wartość losową między 0 a 1), do displayValue()
// przekazywana jest ta sama wartość, ponieważ Math.random()
// jest szacowana tylko raz.
dvInterval = setInterval(displayValue, 5000, Math.random());
```

Metody *setInterval()* można także użyć do wywołania metod obiektu. Należy wtedy podać przynajmniej trzy parametry. Pierwszy parametr to odwołanie do obiektu, drugi to nazwa metody w postaci łańcucha, a trzeci to liczba milisekund interwału.

```
// Włączenie pliku DrawingMethods.as z rozdziału 4.
// w celu uzyskania dostępu do metody drawTriangle().
#include "DrawingMethods.as"

// Utworzenie klipu filmowego i narysowanie w nim trójkąta.
_root.createEmptyMovieClip("triangle", 1);
triangle.lineStyle(1, 0x000000, 100);
triangle.drawTriangle(300, 500, 60);
// Dodanie metody przesuwającej trójkąt o 10 pikseli
// w kierunku x i y.
triangle.move = function(){
  this._x += 10;
  this._y += 10;
};

// Wywoływanie metody move() z klipu filmowego trójkąta co dwie sekundy.
trInterval = setInterval(triangle, "move", 2000);
```

Parametry do metody można przekazać przez funkcję *setInterval()*, wymieniając je po parametrze `milliseconds`:

```
// Włączenie pliku DrawingMethods.as z rozdziału 4.
// w celu uzyskania dostępu do  metody drawTriangle().
#include "DrawingMethods.as"

_root.createEmptyMovieClip("triangle", 1);
triangle.lineStyle(1, 0x000000, 100);
triangle.drawTriangle(300, 500, 60);

// Modyfikacja metody move() — jej zmiana na metodę przyjmującą parametry.
triangle.move = function (x, y) {
  this._x += x;
  this._y += y;
};

// Wywoływanie metody move() z klipu filmowego trójkąta co dwie sekundy
// i przekazanie jej wartości 10 i 10.
trInterval = setInterval(triangle, "move", 2000, 10, 10);
```

W poniższym kodzie metoda *setInterval()* jest używana do utworzenia klipu filmowego zegara, w którym wyświetlacz jest aktualizowany raz na sekundę:

```
// Włączenie pliku DrawingMethods.as z rozdziału 4.
// w celu uzyskania dostępu do  metody drawCircle().
#include "DrawingMethods.as"

// Utworzenie klipu filmowego clock_mc w _root, widocznego na scenie.
// Ten klip filmowy zawiera klipy filmowe ze wskazówkami i tarczą.
_root.createEmptyMovieClip("clock_mc", 1);
clock_mc._x = 200;
clock_mc._y = 200;

// Dodanie klipu filmowego face_mc do clock_mc i narysowanie w nim okręgu.
clock_mc.createEmptyMovieClip("face_mc", 1);
```

```
clock_mc.face_mc.lineStyle(1, 0x000000, 100);
clock_mc.face_mc.drawCircle(100);

// Dodanie wskazówek godzin, minut i sekund do clock_mc i narysowanie linii.
clock_mc.createEmptyMovieClip("hourHand_mc", 2);
clock_mc.hourHand_mc.lineStyle(3, 0x000000, 100);
clock_mc.hourHand_mc.lineTo(50, 0);
clock_mc.createEmptyMovieClip("minHand_mc", 3);
clock_mc.minHand_mc.lineStyle(3, 0x000000, 100);
clock_mc.minHand_mc.lineTo(90, 0);
clock_mc.createEmptyMovieClip("secHand_mc", 2);
clock_mc.secHand_mc.lineStyle(1, 0x000000, 100);
clock_mc.secHand_mc.lineTo(90, 0);

// Utworzenie metody dla klipu filmowego clock_mc, aktualizującego
// wyświetlacz zegara w oparciu o aktualny czas komputera klienta.
clock_mc.updateDisplay = function () {

  // Pozyskanie godziny, minut i sekund z aktualnego czasu
  var d = new Date();
  var hour      = d.getHours();
  var min       = d.getMinutes();
  var sec       = d.getSeconds();

  // Ustawienie obrotu wskazówek według godzin, minut i sekund.
  // Właściwość _rotation jest określana w stopniach, zatem każda wartość
  // jest obliczana przez znalezienie procentu godziny, minuty i sekundy
  // względem pełnego obrotu (i mnożona przez 360, aby uzyskać wartość
  // w stopniach). Dodatkowo od każdej wartości odejmowanych jest 90 stopni,
  // aby uzyskać właściwe przesunięcie z godziny 12, a nie 3.
  this.hourHand_mc._rotation = (((hour + min/60)/12) * 360) -90;
  this.minHand_mc._rotation = (((min + sec/60)/60) * 360) -90;
  this.secHand_mc._rotation = ((sec/60)) * 360) -90;
};

// Wywołanie metody updateDisplay() klipu filmowego clock_mc - raz na sekundę.
clockInterval = setInterval(clock_mc, "updateDisplay", 1000);
```

Zobacz także

Dodanie wyświetlacza cyfrowego do powyższego zegara za pomocą technik opisanych w recepturach 10.6 i 10.3 pozostawiamy Czytelnikom jako ćwiczenie. Zobacz także recepturę 1.7.