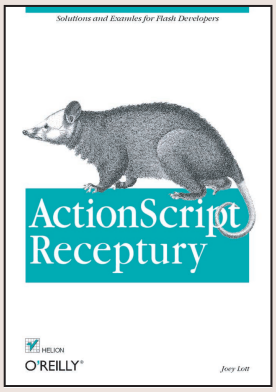
ActionScript. Receptury



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

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.