



Technologia i rozwiązania

Facebook Graph API

Tworzenie rozbudowanych rozwiązań we Flashu

Graph API — Lubię to!

- Jak wykorzystać potencjał 500 milionów użytkowników?
- Jak zarejestrować własną aplikację na Facebooku?
- Jakie możliwości kryje ActionScript 3 SDK for Facebook Platform?

Helion



Michael James Williams

[PACKT]
PUBLISHING

» Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

» 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

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2011

Facebook Graph API. Tworzenie rozbudowanych rozwiązań we Flashu

Autor: Michael James Williams

Tłumaczenie: Łukasz Schmidt

ISBN: 978-83-246-3381-4

Tytuł oryginału: [Facebook Graph API Development with Flash](#)

Format: 170×230, stron: 288



Graph API – Lubię to!

- Jak wykorzystać potencjał 500 milionów użytkowników?
- Jak zarejestrować własną aplikację na Facebooku?
- Jakie możliwości kryje ActionScript 3 SDK for Facebook Platform?

Ponad 500 milionów użytkowników, spędzających 700 000 000 000 minut miesięcznie na jednej witrynie. O czym mowa? O serwisie Facebook! Jego potencjał doceniła niejedna firma. Zastanawiasz się, jak dołączyć do tego grona? Jak zdobyć popularność, fanów i być może zarobić? Uwierz, że to nic trudnego! Facebook udostępnia bogate API, dzięki któremu bez problemu zintegrujesz się z witryną facebook.com.

W tej książce wiedza jest na wyciągnięcie ręki. W trakcie lektury nauczysz się korzystać z dostarczonych funkcji przy użyciu ActionScript 3 SDK for Facebook Platform. Dowiesz się, jak zarejestrować swoją aplikację, uwierzytelniać użytkowników oraz żądać uprawnień rozszerzonych. Ponadto zobaczysz, jak korzystać z wyszukiwarki, wykrywać błędy oraz publikować wiadomości na tym portalu społecznościowym. Książka ta jest idealną pozycją dla każdego fana Facebooka posiadającego zacięcie programistyczne. Sprawdzi się także doskonale w rękach profesjonalistów chcących stworzyć nowe narzędzia lub gry dla portalu Facebook. Polub to!

- Zalety i wady Facebooka
- Wybór hostingu WWW
- Dostęp do Graph API poprzez przeglądarkę
- Pobieranie informacji ze strony za pomocą ActionScript 3
- Rejestracja aplikacji na Facebooku
- Uwierzytelnianie za pomocą ActionScript 3
- Uzyskiwanie rozszerzonych uprawnień
- Pobieranie i stronicowanie wyników
- Wykorzystanie wyszukiwarki
- Pisanie w feedzie użytkownika
- Publikowanie sformatowanych postów
- Dodawanie i usuwanie oznaczenia „Lubię to!”
- Obsługa notatek i wydarzeń
- Tworzenie stron, grup, aplikacji i wideo
- Język FQL

Dotrzyj do milionów użytkowników serwisu Facebook!

Spis treści

O autorze	9
O recenzencie	11
Wstęp	13
Rozdział 1. Wprowadzenie	17
Dlaczego Facebook jest taki dobry?	17
Facebook jest popularny	18
Liczby	19
Facebook jest wszędzie	19
Programowanie dla Facebooka jest ciekawe	21
Hosting WWW	21
Co to jest host WWW?	21
Dlaczego potrzebny jest host WWW?	22
Jak wybrać host?	22
Przydatne oprogramowanie	23
A co z nazwami domen?	23
Jaka znajomość AS3 jest wymagana?	24
Kod źródłowy	25
Napędzany przez...	27
Wykrywanie błędów	27
Uważaj na przechowywanie w pamięci podręcznej	28
Uwaga końcowa	30
Rozdział 2. Witaj w Graphie	31
Dostęp do Graph API poprzez przeglądarkę	31
Czas zacząć działać — ładowanie strony	32
Korzystanie z Graph API za pomocą AS3	34
Czas zacząć działać — pobieranie informacji ze strony za pomocą AS3	35

Czas zacząć działać — deserializowanie obiektu JSON	38
Czas zacząć działać — wizualizowanie informacji	40
Połączenia	42
Czas zacząć działać — wyszukiwanie połączeń w przeglądarce	42
Renderowanie list	46
Czas zacząć działać — renderowanie list postów	46
Renderowanie połączeń	50
Czas zacząć działać — wyświetlanie połączeń obiektu Graph	51
Obiekty żądające	53
Czas zacząć działać — tworzenie obiektu żądającego HTTP	53
Połączenia połączeń	59
Czas zacząć działać — ładowanie zdjęć z albumu	60
Wszystko razem	65
Czas zacząć działać — przemieszczanie się w Graph	65
Podsumowanie	67
Rozdział 3. Wpuść mnie!	69
Co można zobaczyć?	69
Czas zacząć działać — rozglądanie się po kontach innych ludzi	70
Co to ma wspólnego z Graph API?	73
Tokeny dostępu są dowodem autoryzacji	74
Autoryzacja użytkownika i aplikacji	74
Czas zacząć działać — rejestracja aplikacji na Facebooku	75
ID aplikacji + zalogowany użytkownik = token dostępu	78
Czas zacząć działać — żądanie tokenu dostępu przy użyciu przeglądarki	78
Rejestrowanie URI przekierowania dla naszej aplikacji	78
Używanie tokenu dostępu	81
Ja, ja, ja	83
Zachowanie tajemnicy	83
Co dał nam Facebook?	84
Uwierzytelnianie za pomocą AS3	85
Czas zacząć działać — używanie tokenu dostępu w aplikacji Visualizer	85
To oszustwo!	90
Czas zacząć działać — uwierzytelnianie w aplikacji	90
Inne podejście	92
Czas zacząć działać — uwierzytelnianie za pomocą JavaScriptu	92
Tworzenie strony zwrotnej	94
Odbieranie tokenu dostępu	96
Co z użytkownikami, którzy nie używali wcześniej tej aplikacji?	99
Uprawnienia rozszerzone	101
Czas zacząć działać — uzyskiwanie uprawnień rozszerzonych	101
Czas zacząć działać — żądanie uprawnień rozszerzonych	102
Chcę wszystkiego i chcę tego teraz	105
Korzystanie z Adobe ActionScript 3 SDK for Facebook Platform	106
Czas zacząć działać — implementacja SDK	106
Podsumowanie	114

Rozdział 4. Dalej w Graph	117
Pobieranie większej liczby wyników ze stronicowaniem	117
Czas zacząć działać — wyświetlanie liczby obiektów na liście	118
Czas zacząć działać — żądanie większej liczby obiektów	121
Czas zacząć działać — żądanie jeszcze większej liczby obiektów	123
Stronicowanie	124
Czas zacząć działać — uzyskiwanie danych podzielonych na strony	125
Czas zacząć działać — dodawanie parametrów limit i offset do instancji GraphQL	127
Czas zacząć działać — żądanie danych na podstawie daty	129
Czas zacząć działać — dodawanie parametrów since i until do instancji GraphQL	131
Czas zacząć działać — filtrowanie według daty za pomocą UI	133
Partycjonowanie z okazji Twoich urodzin	136
Czas zacząć działać — używanie parametru ids w GraphQL URL	139
Podsumowanie	141
Rozdział 5. Wyszukaj mnie	143
Używanie funkcji Search (szukaj) na stronie głównej	143
Czas zacząć działać — badanie wyników szybkiego wyszukiwania	144
Czas zacząć działać — korzystanie z pełnego wyszukiwania	146
Wyszukiwanie za pomocą GraphQL API	149
Czas zacząć działać — wyszukiwanie bez autoryzacji	149
Czas zacząć działać — wyszukiwanie po dokonaniu autoryzacji	153
Różnice	156
Ograniczenia	156
Czas zacząć działać — implementowanie okna Search (szukaj) w projekcie Visualizer	158
Czas zacząć działać — wyszukiwanie za pomocą SDK	162
Czas zacząć działać — wyszukiwanie w Twoich aktualnościach	165
Czas zacząć działać — wyszukiwanie wśród postów na tablicy znajomego	167
Czas zacząć działać — przeszukiwanie aktualności za pomocą aplikacji Visualizer	169
Podsumowanie	174
Rozdział 6. Dodawanie do Graphu	177
Witaj, Facebooku!	177
Czas zacząć działać — publikowanie na tablicy użytkownika	178
Metody żądań	181
Co to jest metoda żądania?	181
Czas zacząć działać — używanie metody POST	182
Czas zacząć działać — wykrywanie błędów	184
Czas zacząć działać — przyznawanie potrzebnych uprawnień	186
Czas zacząć działać — publikowanie posta za pomocą SDK	188
Dalsza praca z postami na tablicy	190
Czas zacząć działać — publikowanie postów sformatowanych	191
Pisanie na innych tablicach	196
Czas zacząć działać — pisanie za pomocą aplikacji Visualizer na innej tablicy	197
Parametry actions, privacy i source	198
Parametr actions	198

Czas zacząć działać — dodawanie operacji za pomocą parametru actions	199
Parametr privacy	200
Czas zacząć działać — modyfikowanie ustawień prywatności posta	201
Parametr source	203
Usuwanie obiektów Graph	203
Czas zacząć działać — usuwanie posta	204
Czas zacząć działać — usuwanie postów za pomocą aplikacji Visualizer	204
Publikowanie innych obiektów Graph	210
Komentarze	210
Oznaczenie „Lubię to!”	211
Usuwanie oznaczeń „Lubię to!”	212
Notatki	212
Wydarzenia	213
RSVP wydarzeń	214
Albumy	215
Zdjęcia	216
Zameldowania	217
A co z...?	218
Wysyłanie wiadomości do skrzynki	218
Tworzenie stron, grup, aplikacji i wideo	218
Modyfikowanie informacji biograficznych	219
Tworzenie znajomych	219
Zapraszanie znajomych na wydarzenia	219
Podsumowanie	219
 Rozdział 7. FQL ma znaczenie!	 223
Co to jest FQL?	224
Interfejs FQL	224
Modele danych	225
Reprezentacje danych	226
Pobieranie informacji	226
Czas zacząć działać — pobieranie informacji z tabeli page	227
Co z połączeniami?	230
Zdjęcia, albumy i ich właściciele	232
Klucze podstawowe	234
Kurze łapki	235
Tabele łącza	236
Czas zacząć działać — pobieranie listy nazw znajomych użytkownika za pomocą AS3	238
Czas zacząć działać — łatwiejszy sposób	239
Czas zacząć działać — ograniczanie liczby wywołań API do jednego	239
Graph jako warstwa	240
Uprawnienia	241
Sprawdzanie istniejących uprawnień	241
Uzyskiwanie dalszych informacji	242
Ograniczenia	243
Wyszukiwania muszą wykorzystywać pole indeksowane	243
Czy to ma znaczenie praktyczne?	244

Zaawansowany FQL	244
Operatory	245
Operatory porównania	245
Operatory logiczne	245
Sortowanie	246
Stronicowanie	246
Dodatkowe funkcje	247
Wywoływanie kilku zapytań jednocześnie	248
Podsumowanie	248
 Rozdział 8. Zakończenie	 251
Umieszczanie aplikacji online	251
Na Facebooku	252
IFrame	252
Czas zacząć działać — przygotowanie IFrame aplikacji	253
Zakładki na stronie	256
Czas zacząć działać — dodawanie aplikacji do zakładki strony	256
Poza Facebookiem	258
Własna witryna	259
Portale gier Flash	260
Jako aplikacja desktopowa AIR	261
Czas zacząć działać — autoryzacja za pomocą AIR i HTTP	261
Czas zacząć działać — autoryzacja za pomocą AIR i SDK	263
Jako aplikacja AIR dla systemu Android	265
Czas zacząć działać — autoryzacja w systemie Android	265
Konfigurowanie ustawień aplikacji na Facebooku	267
Udostępnianie aplikacji	268
Edytowanie strony profilu aplikacji	269
Katalog aplikacji Facebooka	272
Uwaga na przestrzeganie zasad!	273
Co dalej?	274
Oficjalny AS3 Facebook SDK	274
Inne API dla Facebooka	274
JavaScript SDK	274
Insights API	275
Facebook Chat API	275
Internationalization API	275
Usługa Adobe Social	275
Technologie powiązane	276
PHP	276
Protokół Open Graph	276
Aktualizacje w czasie rzeczywistym	276
Zupełnie nowe i dostępne wkrótce	277
Facebook Credits	277
Użytkownicy testowi	277
Nowe wiadomości	277
Zasoby dla programistów Facebooka	277
Oficjalne zasoby Facebooka	277
Inne dobre witryny	278
Ja, ja, ja!	279

Nadążyć za Zuckerbergami	279
Radzenie sobie ze zmianami	280
Podsumowanie	280
Dodatek A. Odpowiedzi na pytania do quizów	281
Rozdział 2.	281
Rozdział 3.	281
Rozdział 4.	282
Rozdział 5.	282
Rozdział 6.	282
Rozdział 7.	282
Skorowidz	283

Witaj w Graphie

Facebook przechowuje ogromną ilość informacji o ludziach, firmach, wydarzeniach, a także albumy ze zdjęciami i wiele innych. W nim znajdziesz także informacje, w jaki sposób są one połączone: kto jest właścicielem każdego z albumów, kto jest na zdjęciu, która firma organizuje każde wydarzenie.

Przez cztery lata dostęp do wszystkich tych informacji uzyskiwano za pomocą rozbudowanego API, który w miarę dodawania do niego nowych funkcji stawał się coraz bardziej złożony. W kwietniu 2010 Facebook uruchomił **Graph API**, który znacznie uprościł dostęp do wszelkich danych.

W tym rozdziale:

- Poznasz Facebook Graph.
- Dowiesz się, czym jest Graph API i jaką strukturę nadaje wszystkim danym na Facebooku.
- Uzyskasz dostęp do publicznych danych Graphu za pomocą AS3 i Graph API.

Zaczynamy.

Dostęp do Graph API poprzez przeglądarkę

Przejdziemy od razu do rzeczy — zobaczymy, jak Graph API prezentuje informacje ze strony publicznej.

Każdy użytkownik Facebooka ma własny profil osobisty, który możesz zobaczyć po zalogowaniu się na Facebook i kliknięciu łącza *Profile (Profil)* na pasku nawigacji na górze witryny. Profile publiczne wyglądają podobnie, ale są zaprojektowane jako sposób zaistnienia na Facebooku dla przedsiębiorstw, zespołów muzycznych, produktów, organizacji i osób publicznych.

Oznacza to, że wielu ludzi ma zarówno profil osobisty, jak i publiczny. Przykładowo Mark Zuckerberg — twórca Facebooka — ma profil osobisty pod adresem <http://www.facebook.com/zuck> oraz profil publiczny pod adresem <http://www.facebook.com/markzuckerberg>. W ten sposób może używać profilu osobistego do kontaktów ze znajomymi i rodziną, a publicznego — dla fanów i innych zwolenników.

Istnieje jeszcze jeden rodzaj strony: **strona społecznościowa**. Strony tego rodzaju są bardzo podobne do profili osobistych, jednak nie dotyczą osób, tylko innych bytów, takich jak interesujące zagadnienia, doświadczenia i szczytne sprawy. Takie strony automatycznie pobierają informacje z Wikipedii na dany temat, jeśli to potrzebne, oraz przekazują na żywo wszelkie posty na tablicy na dany temat.

Wszystko to może wydawać się nieco zagmatwane, jednak nie powinieneś się przejmować — kiedy zaczniesz korzystać z różnego rodzaju stron, wszystko stanie się jasne.

Czas zacząć działać — ładowanie strony

W przeglądarce przejdź do adresu <http://www.facebook.com/PacktPub>, aby załadować stronę Packt Publishing na Facebooku. Zobaczysz listę najnowszych postów na tablicy, zakładkę *Info* oraz albumy ze zdjęciami (zawierają głównie okładki książek), a także zdjęcie profilowe oraz listę fanów i łączy (patrz rysunek na następnej stronie).

Tak właśnie użytkownicy witryny widzą zawarte w niej informacje. W jaki sposób „zobaczyć” je nasz kod? Możemy sprawdzić, jak Graph API odtwarza zawartość strony Packt Publishing; wystarczy wpisać w przeglądarce adres <https://graph.facebook.com/PacktPub>. Adres ten to **Graph URL** — zauważ, że to ten sam adres, co adres samej strony, ale z bezpiecznym połączeniem https oraz poddomeną *graph* zamiast WWW.

Zobaczysz informacje zaprezentowane w następujący sposób:

```
{
  "id": "204603129458",
  "name": "Packt Publishing",
  "picture": "http://profile.ak.fbcdn.net/hprofile-ak-snc4/hs302.ash1/2327
↳4_204603129458_7460_s.jpg",
  "link": "http://www.facebook.com/PacktPub",
  "category": "Products_other",
  "username": "PacktPub",
  "company_overview": "Packt is a modern, IT focused book publisher,
↳specializing in producing cutting-edge books for communities of developers,
↳administrators, and newbies alike.\n\nPackt published its first book,
↳Mastering phpMyAdmin for MySQL Management in April 2004.",
  "fan_count": 412
}
```

33

Co się zdarzyło?

Pobrałeś reprezentację strony Packt Publishing w Graph API do swojej przeglądarki.

Graph API został zaprojektowany tak, aby łatwo było go poznać — jest sam dla siebie dokumentacją — udało się osiągnąć ten efekt. Jasne jest, że powyższe dane to lista pól i ich wartości.

Jednym polem, którego znaczenie może nie być jasne, jest `id`. To numer, którego Facebook używa wewnętrznie do wskazywania strony. Oznacza to, że strona ma dwa numery ID: numeryczny przypisywany automatycznie przez Facebook oraz alfanumeryczny wybierany przez właściciela strony. Te dwa ID są dla siebie odpowiednikami: jeśli wpiszesz w przeglądarce <https://graph.facebook.com/204603129458>, zobaczysz dokładnie te same dane, co dla <https://graph.facebook.com/PacktPub>.

Do przećwiczenia — oglądanie innych obiektów

Oczywiście, strona Packt Publishing nie jest jedyną stroną, którą możesz obejrzeć za pomocą Graph API w przeglądarce. Wyszukaj inne strony na Facebooku, następnie za pomocą formatu <https://graph.facebook.com/id> obejrzyj ich reprezentacje w Graph API. Czy zawierają więcej informacji, czy mniej?

Następnie przejdź do innych obiektów Facebooka: profili osobistych, wydarzeń, grup. Dla profili osobistych ID może być alfanumeryczny (jeśli osoba zarejestrowała własną nazwę użytkownika Facebooka pod adresem <http://www.facebook.com/username/>), ale najczęściej ID będzie numeryczny — przydzielony automatycznie przez Facebook w chwili rejestracji użytkownika.

Dla niektórych typów obiektów (np. albumów zdjęć) wartości ID nie będzie można łatwo odczytać z adresu URL na witrynie Facebooka. Takimi przypadkami zajmiemy się dalej w tym rozdziale.

Czasem otrzymasz informację o błędzie, taką jak ta:

```
{
  "error": {
    "type": "OAuthAccessTokenException",
    "message": "An access token is required to request this resource."
  }
}
```

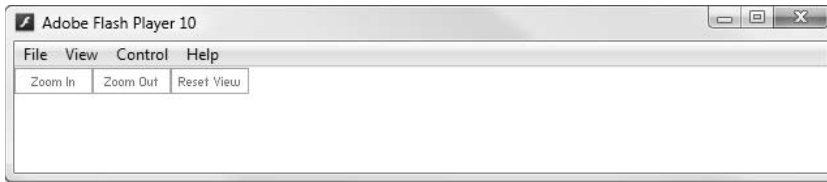
Rozwiązaniem tego problemu zajmiemy się w dalszej części książki.

Korzystanie z Graph API za pomocą AS3

Już wiesz, jak łatwo odczytywać dane Facebooka w przeglądarce. Teraz zobaczysz, jak je pobierać za pomocą AS3.

Czas zacząć działać — pobieranie informacji ze strony za pomocą AS3

Przygotuj projekt za pomocą plików startowych rozdziału 2., tak jak wyjaśniono w rozdziale 1. Sprawdź, czy projekt kompiluje się bez błędów (może pojawić się kilka ostrzeżeń, zależnie od ustawień IDE). Powinieneś zobaczyć SWF o wymiarach 640 na 480 pikseli, w kolorze białym i z trzema przyciskami w lewym górnym rogu: *Zoom In* (powiększ), *Zoom out* (zmniejsz) oraz *Reset View* (zeruj widok).



Ten projekt jest podstawą bogatej aplikacji internetowej (RIA), za pomocą której będzie można przeglądać wszystkie informacje na Facebooku przy użyciu Graph API. Cały kod UI jest już gotowy, potrzeba tylko danych Graph do wyrenderowania informacji. Naszym zadaniem będzie napisanie kodu, który pobierze te dane i przekaże do rendererów.

Nie będę zagłębiał się w szczegóły całego projektu i wyjaśniał, co robi każda klasa, ponieważ tematem tej książki jest wykorzystanie Facebooka w połączeniu z Flashem, a nie budowanie aplikacji RIA. W tej chwili powinieneś wiedzieć, jak działa pojedyncza instancja z pakietu `controllers`. Obiekt klasy `CustomGraphContainerController` jest tworzony w chwili inicjalizacji projektu i odpowiada za przepływ danych do i z Facebooka. W tym celu dziedziczy kilka przydatnych metod po klasie `controllers.GCController`; skorzystamy z nich później.

Za pomocą swojego IDE otwórz klasę `CustomGraphContainerController`. Znajduje się ona w pliku `\src\controllers\CustomGraphContainerController.as`, a jej zawartość jest taka jak w listingu niżej:

```
package controllers
{
    import ui.GraphControlContainer;
    public class CustomGraphContainerController extends GCController
    {
        public function CustomGraphContainerController(a_graphControlContainer:
            ↳GraphControlContainer)
        {
            super(a_graphControlContainer);
        }
    }
}
```

Na początku przy użyciu Graph URL pobierzemy reprezentację strony Packt Publishing w Graph API, tak samo jak robiliśmy to w przeglądarce. W tym celu możemy skorzystać z klasy `URLLoader`.

Klasy `URLLoader` i `URLRequest` używane są razem do ładowania danych z adresu URL. Dane te mogą być tekstem, danymi binarnymi lub zmiennymi zakodowanymi w URL. Ładowanie rozpoczyna się od przekazania obiektu `URLRequest`, którego właściwość `url` zawiera żądany URL, do metody `load()` obiektu `URLLoader`.

Po zakończeniu ładowania żądanych danych `URLLoader` wywołuje zdarzenie `COMPLETE`. Dane można odczytać z właściwości `data` tego zdarzenia.

Zmodyfikuj `CustomGraphContainerController.as` w następujący sposób (nowe linie wyróżnione są pogrubieniem):

```
package controllers
{
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import ui.GraphControlContainer;

    public class CustomGraphContainerController extends GCController
    {

        public function CustomGraphContainerController(a_graphControlContainer:
        ↳GraphControlContainer)
        {
            super(a_graphControlContainer);

            var loader:URLLoader = new ULLoader();
            var request:URLRequest = new URLRequest();
            //wskaż, który Graph URL załadować
            request.url = "https://graph.facebook.com/PacktPub";
            loader.addEventListener(Event.COMPLETE, onGraphDataLoadComplete);
            //rozpocznij właściwy proces ładowania
            loader.load(request);
        }

        private function onGraphDataLoadComplete(a_event:Event):void
        {
            var loader:URLLoader = a_event.target as ULLoader;
            //pobierz dane, które zostały załadowane, i wyświetl je
            var graphData:String = loader.data;
            trace(graphData);
        }
    }
}
```

Tutaj tylko ładujemy informacje z adresu `https://graph.facebook.com/PacktPub` i wypisujemy je w oknie wyjścia.

Przetestuj projekt i obejrzyj zawartość okna wyjścia. Powinieneś zobaczyć następujące dane:

```
{ "id": "204603129458", "name": "Packt Publishing", "picture": "http://profile.ak.fbcdn.net/hprofile-ak-snc4/hs302.ash1/23274_204603129458_7460_s.jpg",
  "link": "http://www.facebook.com/PacktPub", "category": "Products_other",
  "username": "PacktPub", "company_overview": "Packt is a modern, IT focused book publisher, specializing in producing cutting-edge books for communities of developers, administrators, and newbies alike.\n\nPackt published its first book, Mastering phpMyAdmin for MySQL Management in April 2004.", "fan_count": 412 }
```

Jeżeli otrzymujesz błąd, sprawdź, czy Twój kod odpowiada kodowi zamieszczonemu powyżej. Jeśli w oknie wyjścia nie ma nic, upewnij się, że jesteś połączony z internetem. Jeśli ciągle nie nie widać, być może ustawienia zabezpieczeń nie zezwalają na dostęp do internetu poprzez Flash, sprawdź taką możliwość.

Co się zdarzyło?

Podziały wierszy i tabulatory pomiędzy wartościami zostały usunięte, a niektóre znaki ukryte, dzięki czemu tekst trudno odczytać, jednak możesz przekonać się, że to te same dane, które uzyskaliśmy, wpisując w przeglądarce <https://graph.facebook.com/PacktPub>. Nie ma tu żadnych niespodzianek — właśnie tak działa URLLoader.

Dane w takiej postaci nie są szczególnie przydatne. Aby coś z nimi zrobić, będziemy musieli skonwertować je na obiekt, z którym będziemy mogli współpracować bezpośrednio w AS3.

Format wykorzystany w Graph API nosi nazwę JSON (wymawiane „dzejson”, skrót od ang. *JavaScript Object Notation* — obiekt w notacji JavaScript).

JSON jest zrozumiałym dla człowieka formatem danych opartym na tekście. Pozwala na prezentowanie obiektów jako par klucz-wartość w następujący sposób:

```
{
  "klucz1": "wartość1",
  "klucz2": "wartość2",
  "klucz3": "wartość3"
}
```

Wartości mogą być ciągami znaków (w cudzysłowie), liczbami, wartościami Boolean lub null (bez cudzysłowów).

Obiekty JSON mogą także zawierać tablice oznaczone nawiasami kwadratowymi:

```
{
  "klucz1": "wartość1",
  "tablica": [
    "Pierwszy element w tablicy",
    "Drugi element w tablicy",
    "Trzeci element w tablicy"
  ]
}
```

Mogą nawet zawierać inne obiekty JSON oznaczone zagnieżdżonymi nawiasami klamrowymi:

```
{
  "klucz1": "wartość1",
  "podObiekt":
  {
    "podKlucz1": "podWartość1",
    "podKlucz2": "podWartość2",
  }
}
```

Takie podobiekty mogą zawierać inne obiekty lub tablice, a tablice mogą zawierać kolejne obiekty i tablice.

Zauważ, że przypomina to składnię AS3 służącą do deklarowania obiektu:

```
var obiektAS3:Object = {
  klucz1:"wartość1",
  klucz2:"wartość2",
  podObiekt:{
    podKlucz1:"podwartość1"
  },
  mojaTablica:[1, 2, 3]
}
```

Więcej informacji znajdziesz na <http://www.json.org>.

W odróżnieniu od XML, AS3 nie ma wbudowanych funkcji obsługujących obiekty JSON, istnieje jednak oficjalnie wspierana biblioteka, która to robi.

Czas zacząć działać — deserializowanie obiektu JSON

Biblioteka Adobe `as3corelib` zawiera zestaw klas narzędziowych służących do serializowania i deserializowania JSON. Jest dostępna pod adresem <http://github.com/mikechambers/as3corelib>, ale nie musisz jej pobierać, ponieważ znajduje się także w katalogu `\src\` projektu (składa się ze wszystkich klas w pakiecie `com.adobe.*`).

1. W `CustomGraphContainerController.as` zaimportuj klasę JSON:
2. Zmodyfikuj funkcję `onGraphDataLoadComplete()` tak, aby deserializowała ciąg znaków JSON na obiekt zamiast prostego wyświetlania tego ciągu:

```
private function onGraphDataLoadComplete(a_event:Event):void
{
  var loader:URLLoader = a_event.target as URLLoader;
```

```
//pobierz dane, które zostały załadowane, i wyświetl je
var graphData:String = loader.data;
var decodedJSON:Object = JSON.decode(graphData);
}
```

3. Wyświetl właściwość name nowego obiektu, aby sprawdzić działanie kodu:

```
private function onGraphDataLoadComplete(a_event:Event):void
{
    var loader:URLLoader = a_event.target as URLLoader;
    //pobierz dane, które zostały załadowane, i wyświetl je
    var graphData:String = loader.data;
    var deserialisedJSON:Object = JSON.decode(graphData);
    trace("name:", decodedJSON.name);
}
```

4. Skompiluj i uruchom SWF. Rezultat powinien być następujący:

name: Packt Publishing

Co się zdarzyło?

Przekazaliśmy ciąg znaków do metody JSON.decode():

```
{
    "id":"204603129458",
    "name":"Packt Publishing",
    "picture":"http://profile.ak.fbcdn.net/hprofile-ak-snc4/hs302.ash1/23274_204603129458_7460_s.jpg",
    "link":"http://www.facebook.com/PacktPub",
    "category":"Products_other","username":"PacktPub",
    "company_overview":"Packt is a modern, IT focused book publisher,specializing
    ↪in producing cutting-edge books for communities of developers, administrators,
    ↪and newbies alike.\n\nPackt published its first book, Mastering phpMyAdmin
    ↪for MySQL Management in April 2004.",
    "fan_count":412
}
```

W efekcie łańcuch został zamieniony na obiekt własny AS3, tak jakbyśmy wpisali:

```
var graphObject:Object = {};
graphObject.id = "204603129458";
graphObject.name = "Packt Publishing";
graphObject.picture = "http://profile.ak.fbcdn.net/hprofile-ak-snc4/hs302.ash1/23274_204603129458_7460_s.jpg";
graphObject.link = "http://www.facebook.com/PacktPub";
graphObject.category = "Products_other";
graphObject.username = "PacktPub";
graphObject.company_overview = "Packt is a modern, IT focused book publisher,
specializing in producing cutting-edge books for communities of developers,
administrators, and newbies alike.\n\nPackt published its first book, Mastering
phpMyAdmin for MySQL Management in April 2004."
graphObject.fan_count = 412;
```

(Zauważ, że w odróżnieniu od „surowego” ciągu znaków, którego użyliśmy wcześniej, znaki ukośnika w URL nie zostały oznaczone znakami ucieczki).

Oznacza to, że możemy łatwo uzyskać dostęp do wszelkich informacji, które posiada Facebook na temat tej strony, możemy nawet iterować poprzez poszczególne elementy danych.

Czas zacząć działać — wizualizowanie informacji

Dość instrukcji `trace()`! Czas wyświetlić coś we właściwym pliku SWF.

`CustomGraphContainerController` dziedziczy metodę `renderGraphObject()`, która zrobi to dla nas. Wystarczy przekazać jej argument typu `graph.GraphObject`.

`GraphObject.as` jest prostą klasą, otwórz plik i obejrzyj ją:

```
package graph
{
    import graph.controls.GraphObjectRenderer;
    public dynamic class GraphObject extends BaseGraphItem
    {
        public var rendererObject:GraphObjectRenderer;
        public var graphObjectListRenderers:Array = [];

        public function GraphObject()
        {
        }
    }
}
```

Nie powinieneś zajmować się tym kodem. Musisz tylko wiedzieć, że jest oznaczony jako `dynamic`, co znaczy, że możesz tworzyć nowe właściwości w czasie wykonywania bez nadawania im nazw wcześniej. Możemy więc zrobić tak:

```
var graphObject:GraphObject = new GraphObject();
graphObject.favoriteColor = "red";
```

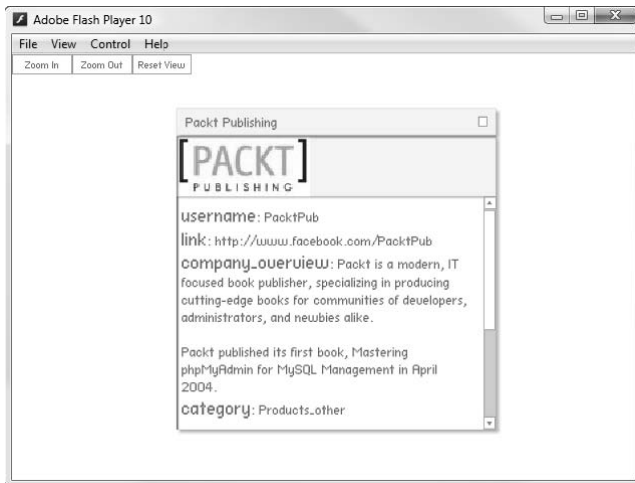
Kiedy `GraphObject` zostanie przekazany do metody `CustomGraphContainerController.renderGraphObject()`, każda z właściwości `GraphObject` zostanie wyrenderowana na liście — automatycznie, każda właściwość wraz z tymi dwiema, które już zostały zdefiniowane wewnątrz klasy.

Wewnątrz funkcji `CustomGraphContainerController.onGraphDataLoadComplete()` wykonaj następujące czynności.

1. Utwórz nową instancję GraphObject.
2. Skopiuj wszystkie właściwości decodedJSON do nowego GraphObject.
3. Przekaż GraphObject do renderGraphObject().
4. Kod, który to wszystko wykonuje, jest następujący:

```
private function onGraphDataLoadComplete(a_event:Event):void
{
    var loader:URLLoader = a_event.target as URLLoader;
    //pobierz dane, które zostały załadowane, i wyświetl je
    var graphData:String = loader.data;
    var decodedJSON:Object = JSON.decode(graphData);
    var graphObject:GraphObject = new GraphObject();
    //skopiuj wszystkie właściwości z decodedJSON do graphObject
    for (var key:String in decodedJSON)
    {
        graphObject[key] = decodedJSON[key];
    }
    this.renderGraphObject(graphObject);
}
```

5. Skompiluj i przetestuj. Otrzymany SWF pokazany został na rysunku poniżej.



Kliknij przycisk *Zoom In* (powiększ) kilka razy, aby renderer stał się większy i lepiej widoczny, tak jak na rysunku powyżej. Twój renderer może wyświetlać pola w innej kolejności; Facebook zwraca pola w przypadkowej kolejności.

Co się zdarzyło?

Okno, które pojawiło się na scenie, nazywam rendererem, a konkretnie **rendererem obiektu Graph**. Może być ono przeciągane poprzez chwycenie za pasek tytułowy, jego zawartość można przewijać, a całe okno zamknąć, klikając przycisk w prawym górnym rogu.

Udało Ci się pobrać dane z Graph API Facebooka i wyświetlić je w SWF. Plik SWF jest elastyczny — wystarczy, że zmienisz wartość `request.url`, tak aby wskazywała na Graph URL innego obiektu Facebooka, i zobaczysz obiekt ten wyświetlony w rendererze.

Większość danych z `GraphObject` została wyświetlona w polu tekstowym wewnątrz okna w prostym formacie "klucz: wartość". Pole `name` strony jest wyświetlone na pasku tytułowym okna, a jeśli strona ma pole `picture` (w JSON widzimy, że PacktPub ma), obraz jest pobierany i wyświetlany wewnątrz renderera za pomocą obiektu `Loader`.

Podobnie do `URLLoader`, klasa `flash.display.Loader` pobiera obiekt, który wskazuje `URLRequest`, oraz wywołuje zdarzenie `COMPLETE`, kiedy pobieranie zostaje zakończone. W odróżnieniu od `URLLoader`, `Loader` służy do pobierania obrazów i plików SWF, a zdarzenie jest wywoływane przez jeden z jego podobiektów — `contentLoaderInfo`. `Loader` jest rozszerzeniem `DisplayObject` i przyjmuje wygląd obrazu, kiedy skończy go pobierać.

Model zabezpieczeń Flasha zapobiega dostępowi do danych obrazów z plików SWF znajdujących się w innej domenie niż obraz, chyba że plik reguł międzydomenowych umieszczony w domenie obrazu na to zezwala. Na szczęście, plik tego rodzaju na Facebooku jest tolerancyjny i pozwala na dostęp z każdej domeny.

Jest to graficzna reprezentacja obiektu strony z Graph API.

Połączenia

Być może myślisz: „To bardzo ładnie, ale przecież okno nie pokazuje wszystkich informacji związanych ze stroną, czyż nie? Gdzie są posty z tablicy i zdjęcia?”.

Czas zacząć działać — wyszukiwanie połączeń w przeglądarce

Facebook traktuje wszystkie posty z tablic, zdjęcia, wideo, a nawet informacje o statusie jako odrębne obiekty w Graph API; nie umieszcza ich razem w obiekcie strony. Oto na przykład obiekt reprezentujący pojedynczy post autorstwa Packt Publishing:

```
{
  "id": "204603129458_127056137323572",
  "from": {
    "name": "Packt Publishing",
    "category": "Products_other",
    "id": "204603129458"
  },
}
```

```

"message": "The Amazon SimpleDB Developer Guide has been published! Get your
↳copy now! http://bit.ly/b1FQUG",
"picture":
"http://external.ak.fbcdn.net/safe_image.php?d=c4a7887cb52dd8f93e439aaec13c034
↳b&w=130&h=130&url=https%3A%2F%2Fwww.packtpub.com%2Fsites%2Fdefault%2Ffiles%
↳2Fimagecache%2Fproductview%2F7344EN_MockupCover%2520Template.jpg",
"link": "http://bit.ly/b1FQUG",
"name": "Amazon SimpleDB Developer Guide | Packt Publishing Technical & IT Book
↳Store",
"caption": "bit.ly",
"description": "Gain in-depth understanding of Amazon SimpleDB with PHP, Java,
↳and Python examples, and run optimized database-backed applications on
Amazon\\'s Web Services cloud",
"icon": "http://static.ak.fbcdn.net/rsrc.php/zB010/hash/9yv171tw.gif",
"type": "link",
"created_time": "2010-06-04T12:39:44+0000",
"updated_time": "2010-06-04T12:39:44+0000",
"likes": 1
}

```

Ten obiekt już wygaś i nie jest dostępny przy użyciu Graph API, jednak — jak być może zgadłeś — był dostępny pod adresem https://graph.facebook.com/204603129458_127056137323572. Jest w tym samym formacie, co obiekt strony, chociaż ma kilka innych pól, a więc nasz renderer obiektów Graph mógłby wyrenderować go poprawnie.

Oczywiście, takie rozwiązanie jest bezużyteczne, chyba że będziemy znali ID każdego z postów przypisanych Packt Publishing. Nie wiadomo, skąd uzyskać takie dane. A może jednak wiadomo?

Powiedziałem wcześniej, że Graph API został zaprojektowany tak, aby być dokumentacją dla samego siebie. Możemy zażądać dodatkowych metainformacji o dowolnym obiekcie Graph poprzez dodanie flagi *metadata=1* na końcu dowolnego Graph URL. Otwórz <https://graph.facebook.com/PacktPub?metadata=1> w przeglądarce. W JSON pojawi się nowa właściwość *type*.

```
"type": "page"
```

To jest przydatne, bo — jak powiedziałem — posty i strony (oraz wszystkie obiekty Graph) przyjmują ten sam format, a ta właściwość umożliwia ich rozpoznawanie.

W tej chwili jednak bardziej interesuje nas nowy obiekt *metadata*. Zawiera on jeden obiekt *connections* oraz jedną tablicę *fields*. Przyjrzyjmy się najpierw tablicy *fields*:

```

"fields": [
{
  "name": "id",
  "description": "The page's ID"
},
{
  "name": "name",
  "description": "The page's name"
}
]

```

```

    },
    {
      "name": "picture",
      "description": "The pages profile picture"
    },
    {
      "name": "category",
      "description": "The page's category"
    },
    {
      "name": "fan_count",
      "description": "\\* The number of fans the page has"
    }
  ]

```

Jest to lista wyjaśniająca, co reprezentuje każde z pól w ciele obiektu Graph. W czasie pisania tej książki została dodana nowa funkcja, a więc lista może być pełniejsza, kiedy Ty ją zobaczysz.

Oto obiekt connections:

```

"connections": {
  "feed": "https://graph.facebook.com/packtpub/feed",
  "posts": "https://graph.facebook.com/packtpub/posts",
  "tagged": "https://graph.facebook.com/packtpub/tagged",
  "statuses": "https://graph.facebook.com/packtpub/statuses",
  "links": "https://graph.facebook.com/packtpub/links",
  "notes": "https://graph.facebook.com/packtpub/notes",
  "photos": "https://graph.facebook.com/packtpub/photos",
  "albums": "https://graph.facebook.com/packtpub/albums",
  "events": "https://graph.facebook.com/packtpub/events",
  "videos": "https://graph.facebook.com/packtpub/videos"
}

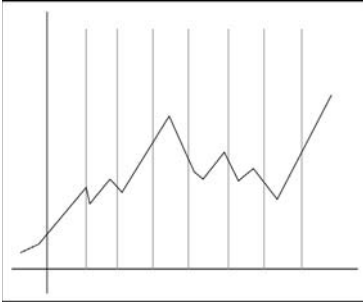
```

Obejrzyj w przeglądarce jeden z adresów URL z poprzedniej listy: <http://graph.facebook.com/packtpub/posts>. Zwraca on strukturę JSON zawierającą tablicę data oraz obiekt paging. Tablica data zawiera kilka obiektów post, obiektem paging zajmiemy się w dalszej części książki.

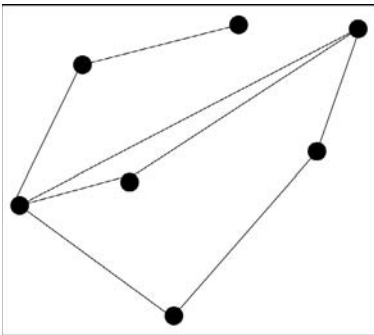
Co się zdarzyło?

Parametr metadata=1 nakazuje interfejsowi Graph API wyświetlenie wszystkich metadanych o bieżącym obiekcie, w tym przypadku są to między innymi typ obiektu, tablica opisów właściwości obiektu oraz wszystkie adresy URL, które zawierają listy obiektów powiązanych z tą stroną.

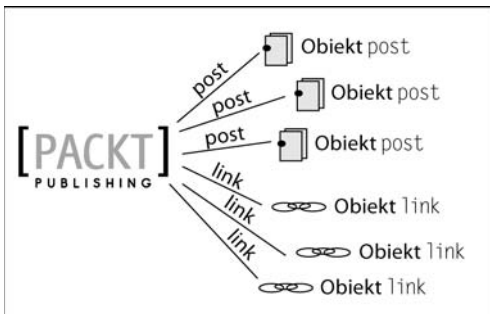
Od takiego układu pochodzi nazwa Graph API. W zwykłym znaczeniu angielskie słowo *graph* oznacza wykres, taki jak pokazany na kolejnym rysunku.



Jednak w matematyce „graf” to zbiór wierzchołków połączonych krawędziami, tak jak na rysunku poniżej.

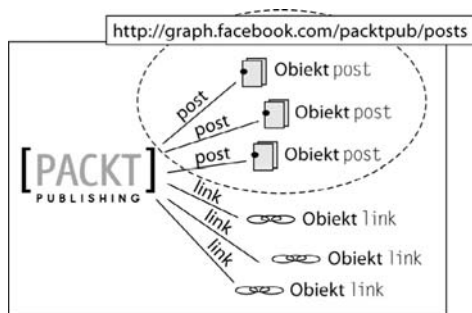


Graph API to reprezentacja danych przechowywanych na Facebooku, ułożonych tak jak na kolejnym rysunku.



Na diagramie z rysunku powyżej każdy obiekt jest wierzchołkiem, a krawędzie reprezentują różne rodzaje połączeń.

Przywołanie <http://graph.facebook.com/packtpub/posts> zwraca wszystkie wierzchołki złączone z PacktPub za pomocą połączenia typu posts, czyli wszystkie obiekty post, które zostały umieszczone na tablicy wydawnictwa Packt.



Do ćwiczenia — badanie połączeń

Poznałeś parametr metadata, możesz zatem za pomocą przeglądarki zbadać różne rodzaje połączeń i zobaczyć, jakiego rodzaju obiekty można znaleźć.

Renderowanie list

Co się stanie, jeśli spróbujesz załadować <https://graph.facebook.com/packtpub/posts> za pomocą tego samego kodu, którego użyliśmy do załadowania obiektu strony Packt Publishing?

W panelu wyjścia otrzymasz następujący komunikat:

Graph Object was null!

Nie udało się. Sposób, w jaki Graph API określa strukturę JSON, w tym przypadku jest zupełnie różny od struktury JSON dla strony, posta lub każdego innego obiektu Graph. To samo będzie dotyczyć JSON dla adresów URL innych połączeń. Strukturę tę nazywamy **listą Graph**.

Czas zacząć działać — renderowanie list postów

Jako że właściwość `data` listy Graph jest tablicą obiektów Graph, możemy przejść przez nią w pętli i utworzyć renderer obiektu Graph dla każdego elementu. Możesz spróbować to zrobić, ja jednak preferuję inne rozwiązanie.

Utworzyłem drugi renderer: tym razem **renderer listy Graph**. Utworzyłem także klasę `graph.GraphList`. `CustomGraphContainerController` dziedziczącą metodę `renderGraphList()`. Przyjmuje ona jako parametr obiekt typu `graph.GraphList` i tworzy nowy renderer listy Graph służący do wyświetlania jego zawartości. Potrzebujemy więc listy Graph — otrzymamy ją od Graph API — którą przekształcimy w instancję klasy `GraphList`. Klasa `GraphList` jest nieco bardziej skomplikowana niż `GraphObject`; posiada metodę `addToList()`, do której możemy przekazać dowolną instancję `GraphObject`, aby dodać ją do listy.

Ciągle będziemy przechodzić w pętli przez tablicę data, ale zamiast renderować każdy GraphObject oddzielnie, dodamy go do GraphList i ją wyrenderujemy.

Zmodyfikuj URL, którego potrzebuje CustomGraphContainerController, tak aby ładowana była lista postów:

```
public function
CustomGraphContainerController(a_graphControlContainer:GraphControlContainer)
{
    super(a_graphControlContainer);
    var loader:URLLoader = new URLLoader();
    var request:URLRequest = new URLRequest();
    // wskaż, który Graph URL załadować
    request.url = "https://graph.facebook.com/PacktPub/posts";
    loader.addEventListener(Event.COMPLETE, onGraphDataLoadComplete);
    // rozpocznij właściwy proces ładowania
    loader.load(request);
}
```

Po załadowaniu trzeba zweryfikować, czy zwrócony element to obiekt Graph, czy lista Graph. Możemy to zrobić, sprawdzając właściwość data: jeżeli właściwość ta istnieje, możemy przyjąć, że element jest listą.

```
private function onGraphDataLoadComplete(a_event:Event):void
{
    var loader:URLLoader = a_event.target as URLLoader;
    //pobierz dane, które zostały załadowane, i wyświetl je
    var graphData:String = loader.data;
    var decodedJSON:Object = JSON.decode(graphData);
    if (decodedJSON.data)
    {
        //ma właściwość data, a więc przyjmujemy, że jest to lista Graph
    }
    else
    {
        //nie ma właściwości data, a więc przyjmujemy, że jest to obiekt Graph
        var graphObject:GraphObject = new GraphObject();
        //skopiuj wszystkie właściwości z rozkodowanej struktury JSON do graphObject
        for (var key:String in decodedJSON)
        {
            graphObject[key] = decodedJSON[key];
        }
        this.renderGraphObject(graphObject);
    }
}
```

Wewnątrz bloku if tworzymy na początku instancję GraphList:

```
if (decodedJSON.data)
{
    //ma właściwość data, a więc przyjmujemy, że jest to lista Graph
```

```

    var graphList:GraphList = new GraphList();
}

```

(Będziesz musiał dodać instrukcję `import graph.GraphList`).

Pamiętaj, że `decodedJSON.data` jest tablicą obiektów; przechodzimy przez nią w pętli i tworzymy `GraphObject` dla każdego elementu.

```

if (decodedJSON.data)
{
    //ma właściwość data, a więc przyjmujemy, że jest to lista Graph
    var graphList:GraphList = new GraphList();
    var childGraphObject:GraphObject;
    for each (var childObject:Object in decodedJSON.data)
    {
        childGraphObject = new GraphObject();
        for (var childKey:String in childObject)
        {
            childGraphObject[childKey] = childObject[childKey];
        }
    }
}

```

To w zasadzie to samo, co robiliśmy z `decodedJSON` przy ładowaniu pojedynczego obiektu `Graph`.

A druga właściwość wewnątrz listy `Graph`, czyli obiekt `paging`? Powinniśmy dodać także ją:

```

if (decodedJSON.data)
{
    //ma właściwość data, a więc przyjmujemy, że jest to lista Graph
    var graphList:GraphList = new GraphList();
    var childGraphObject:GraphObject;
    for each (var childObject:Object in decodedJSON.data)
    {
        childGraphObject = new GraphObject();
        for (var childKey:String in childObject)
        {
            childGraphObject[childKey] = childObject[childKey];
        }
        graphList.addToList(childGraphObject);
    }
    graphList.paging = decodedJSON.paging;
}

```

Na zakończenie przekazujemy instancję `GraphList` do metody `renderGraphList()`:

```

if (decodedJSON.data)
{
    //ma właściwość data, a więc przyjmujemy, że jest to lista Graph
    var graphList:GraphList = new GraphList();

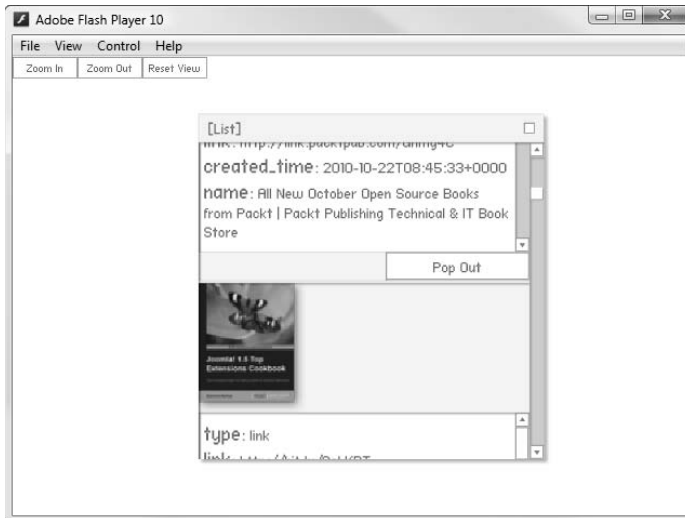
```

```

var childGraphObject:GraphObject;
for each (var childObject:Object in decodedJSON.data)
{
    childGraphObject = new GraphObject();
    for (var childKey:String in childObject)
    {
        childGraphObject[childKey] = childObject[childKey];
    }
    graphList.addToList(childGraphObject);
}
graphList.paging = decodedJSON.paging;
this.renderGraphList(graphList);
}

```

Skompiluj i przetestuj SWF. Rezultat pokazany został na rysunku poniżej.



Jest to okno (można je przewijać) zawierające wszystkie obiekty Graph z list.

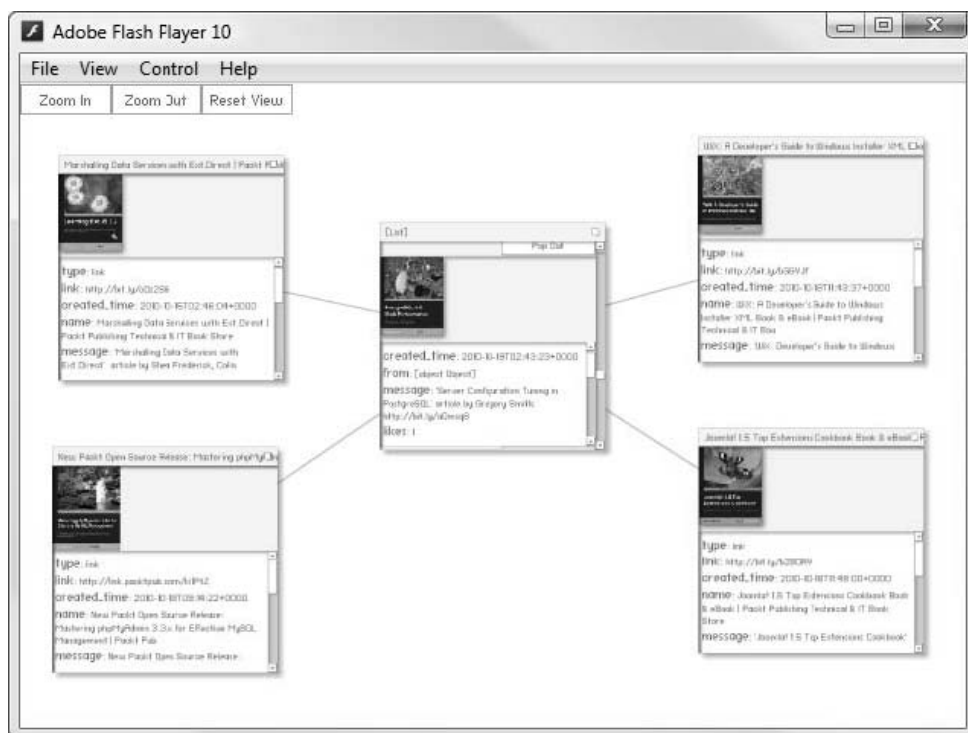
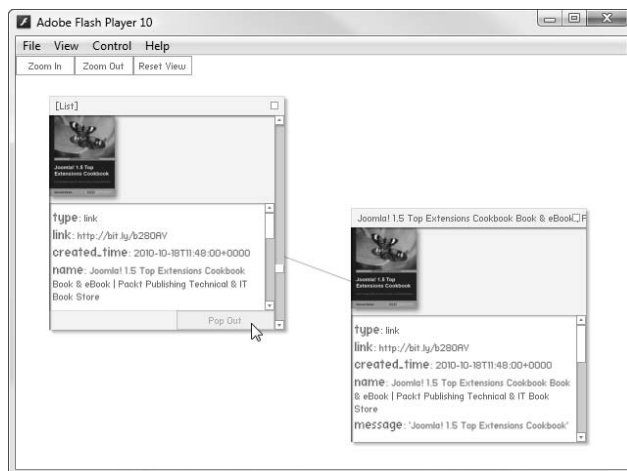
Co stanie się, kiedy klikniesz przycisk *Pop Out* (wyskocz) poniżej obiektu Graph? (Patrz pierwszy rysunek na następnej stronie).

Co się zdarzyło?

Obiekt Graph zostaje wyświetlony we własnym rendererze obiektu Graph z szarą linią łączącą go z odpowiednią pozycją na liście. Dzięki temu można obejrzeć kilka pozycji listy jednocześnie (patrz drugi rysunek na następnej stronie).

(Możesz przeciągać poszczególne renderery lub przeciągać tło, aby przesunąć wszystko naraz).

Staje się jasne, że lista Graph jest tylko kolekcją obiektów Graph.



Renderowanie połączeń

Pokazałem połączenia biegnące od list Graph do obiektów Graph, kolejnym krokiem będzie ukazanie połączeń od obiektów Graph do list Graph.

Czas zacząć działać — wyświetlanie połączeń obiektu Graph

Renderer obiektu Graph może pokazać listę wszystkich połączeń obiektu, jeśli lista ta jest dołączona do obiektu Graph.

Musisz tylko z Graph API wydobyć tę listę razem z żądanym obiektem Graph. Nasz kod tworzy instancję `GraphObject` ze strukturą JSON, kopiując wszystkie właściwości JSON do `GraphObject`, a więc te metadane zostaną dołączone. Musisz tylko dodać flagę `metadata=1` na końcu Graph URL, którego żądamy, a kod zrobi całą resztę.

Możesz to zrobić, zmieniając kod żądania, tak jak we fragmencie poniżej:

```
public function CustomGraphContainerController(a_graphControlContainer:
↳GraphControlContainer)
{
    super(a_graphControlContainer);

    var loader:URLLoader = new URLLoader();
    var request:URLRequest = new URLRequest();
    //wskaż, który Graph URL załadować
    request.url = "https://graph.facebook.com/PacktPub?metadata=1";
    loader.addEventListener(Event.COMPLETE, onGraphDataLoadComplete);
    //rozpocznij właściwy proces ładowania
    loader.load(request);
}
```

Istnieje jednak bardziej elegancki sposób zrobienia tego samego — za pomocą klasy `URLVariables`.

W `CustomGraphContainerController.as` dodaj linię importującą tę klasę:

```
import flash.net.URLVariables;
```

Następnie zmodyfikuj konstruktor, tak jak poniżej:

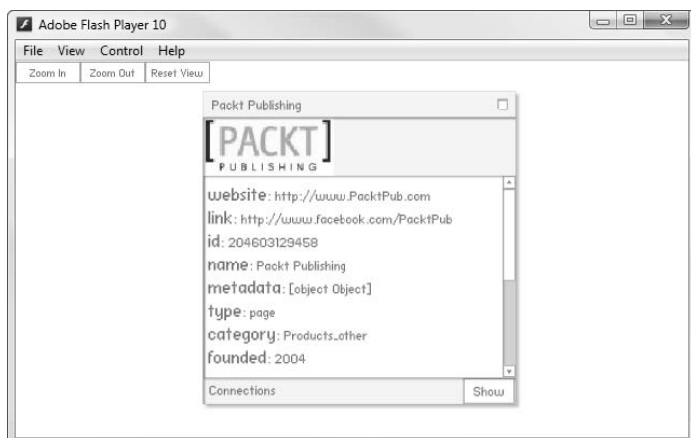
```
public function CustomGraphContainerController(a_graphControlContainer:
↳GraphControlContainer)
{
    super(a_graphControlContainer);

    var loader:URLLoader = new URLLoader();
    var request:URLRequest = new URLRequest();
    var variables:URLVariables = new URLVariables();
    //wskaż, który Graph URL załadować
    request.url = "https://graph.facebook.com/PacktPub";
    variables.metadata = 1;
    request.data = variables;
    loader.addEventListener(Event.COMPLETE, onGraphDataLoadComplete);
}
```

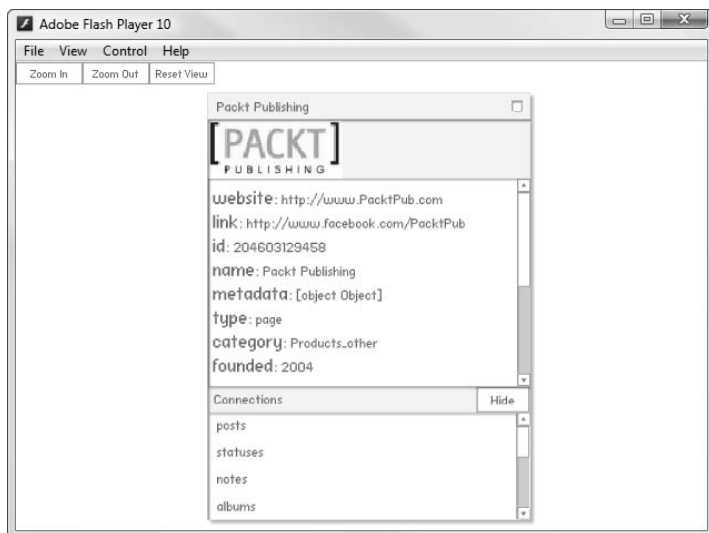
```
//rozpocznij właściwy proces ładowania
loader.load(request);
}
```

Jak najprawdopodobniej domyśliłeś się, ustawienie `variables.metadata = 1` jest dokładnie tym samym, co dodanie `?metadata=1` na końcu adresu URL. Rozwiązanie wymaga kilku dodatkowych linii, ale znacząco ułatwia ustawianie różnych parametrów oraz pozwala oddzielić je od URL.

Skompiluj SWF; powinieneś uzyskać następujący efekt.



Czy zauważyłeś nowy przycisk *Connections* (połączenia) w dolnej części renderera? Kliknij przycisk *Show* (pokaż).



Co się zdarzyło?

Możesz teraz zobaczyć wszystkie połączenia obiektu Graph w jego rendererze. Oczywiście, nie jest to szczególnie interesujące, chyba że postanowisz zobaczyć, co jest na drugim końcu połączenia!

Obiekty żądające

Najlepiej byłoby, gdyby po kliknięciu przez użytkownika połączenia na liście przewijanej powstał i został wyświetlony nowy renderer listy Graph dla tego połączenia.

Aby to osiągnąć, dodamy do listy detektor zdarzenia `MouseEvent.CLICK` i użyjemy go do wywołania nowego żądania `URLLoader` dla klikniętego połączenia.

Nas szczęście, kod UI został już dołączony do projektu; musimy tylko z niego skorzystać. W tym celu użyjemy czegoś, co nazywam **obiektem żądającym** (ang. *requestor*).

Czas zacząć działać — tworzenie obiektu żądającego HTTP

Celem naszego działania jest przesunięcie całego kodu odnoszącego się do `URLLoader` z `CustomGraphContainerController` do oddzielnej klasy `HTTPRequestor`. Później zastąpimy konstruktor `CustomGraphContainerController` następującym:

```
public function CustomGraphContainerController(a_graphControlContainer:
↳GraphControlContainer)
{
    super(a_graphControlContainer);
    _requestor = new HTTPRequestor();
    _requestor.request(new GraphRequest("PacktPub"));
}
```

Po co? No cóż, poza utworzeniem bardziej eleganckiego kodu, istnieją jeszcze dwie istotne korzyści.

1. O wiele łatwiej będzie zażądać kilku obiektów Graph lub list Graph; nie będzie trzeba tworzyć kilku instancji `URLLoader`.
2. W następnym rozdziale zobaczysz, jak używać oficjalnego Adobe ActionScript 3 SDK for Facebook Platform do pobierania informacji z Graph API. Jeśli kod żądania zostanie wydzielony w jednej klasie, będziesz musiał zmienić tylko jedną linię, aby skorzystać z SDK zamiast HTTP:

```
public function CustomGraphContainerController(a_graphControlContainer:
↳GraphControlContainer)
{
```

```

    super(a_graphControlContainer);
    _requestor = new SDKRequestor();
    _requestor.request(new GraphRequest("PacktPub"));
}

```

GraphRequest jest prostą klasą. Jej konstruktor umożliwia użycie dwóch parametrów do wskazania, co chcesz pobrać z Graph API:

- objectID, nazwa dowolnego obiektu Graph,
- connectionID, nazwa dowolnego połączenia z tym obiektem Graph.

Aby zażądać strony Packt Publishing, użyjemy GraphRequest:

```
newGraphRequest("PacktPub");
```

Gdy zażądamy listy postów ze strony Packt Publishing, użyjemy następującego kodu:

```
newGraphRequest("PacktPub", "posts");
```

Klasa została już napisana i znajduje się w `\src\graph\apis\http\HTTPRequestor.as`. Obejrzyj ją! Istnieje w niej kilka zmian w porównaniu z kodem napisanym dla CustomGraphContainer ➔ Controller.as, ale wszystkie zostały wyjaśnione w komentarzach:

```

package graph.apis.http
{
    import events.DialogEvent;
    import events.RequestEvent;
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.HTTPStatusEvent;
    import flash.events.IEventDispatcher;
    import flash.events.IOErrorEvent;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.net.URLVariables;
    import flash.utils.Dictionary;
    import graph.apis.base.IRequestor;
    import graph.BaseGraphItem;
    import graph.GraphList;
    import graph.GraphObject;
    import graph.GraphRequest;
    import com.adobe.serialization.json.JSON;

    //klasa musi wywoływać zdarzenia (przeglądaj kod, zobaczysz dlaczego)
    public class HTTPRequestor extends EventDispatcher implements IRequestor
    {
        //służy do ustalania, który GraphRequest utworzył określony
        //URLLoader
        private var _requests:Dictionary = new Dictionary();

        public function HTTPRequestor(target:IEventDispatcher = null)

```

```

{
    //jest potrzebne, ponieważ klasa rozszerza EventDispatcher
    super(target);
}

public function request(a_request:GraphRequest):void
{
    var loader:URLLoader = new URLLoader();
    var urlRequest:URLRequest = new URLRequest();
    var variables:URLVariables = new URLVariables();

    //tworzymy URL z parametrów GraphRequest
    urlRequest.url = "https://graph.facebook.com/" + a_request.objectID;
    if (a_request.connectionID)
    {
        urlRequest.url += "/" + a_request.connectionID;
    }
    variables.metadata = 1;
    urlRequest.data = variables;

    //służy do ustalania, który GraphRequest utworzył określony
    //URLLoader w czasie dalszego działania
    _requests[loader] = a_request;
    loader.addEventListener(Event.COMPLETE, onGraphDataLoadComplete);
    loader.load(urlRequest);
}

private function onGraphDataLoadComplete(a_event:Event):void
{
    var loader:URLLoader = a_event.target as URLLoader;
    var graphData:String = loader.data;
    var decodedJSON:Object = JSON.decode(graphData);
    //znajdujemy oryginalny GraphRequest użyty do utworzenia URLLoader
    var originalRequest:GraphRequest = _requests[loader] as
    GraphRequest;
    if (decodedJSON.data)
    {
        var graphList:GraphList = new GraphList();
        var childGraphObject:GraphObject;
        for each (var childObject:Object in decodedJSON.data)
        {
            childGraphObject = new GraphObject();
            for (var childKey:String in childObject)
            {
                childGraphObject[childKey] = childObject[childKey];
            }
            graphList.addToList(childGraphObject);
        }
        graphList.paging = decodedJSON.paging;
        //używamy właściwości oryginalnego GraphRequest do dodania
    }
}

```

```

        //kilku danych do samej graphList
        graphList.ownerID = originalRequest.objectID;
        graphList.connectionType = originalRequest.connectionID;

        //ta klasa nie ma metody renderGraphList(),
        //a więc wywołujemy zdarzenie, które CustomGraphContainerController
        //będzie wykrywał i wywoła własną metodę renderGraphList()
        dispatchEvent(new RequestEvent(RequestEvent.REQUEST_COMPLETED,
        ↳graphList));
    }
else
{
    var graphObject:GraphObject = new GraphObject();
    for (var key:String in decodedJSON)
    {
        graphObject[key] = decodedJSON[key];
    }
    //ta klasa nie ma metody renderGraphList(),
    //a więc wywołujemy zdarzenie, które CustomGraphContainerController
    //będzie wykrywał i wywoła własną metodę renderGraphList()
    dispatchEvent(new RequestEvent(RequestEvent.REQUEST_COMPLETED,
    ↳graphObject));
}
}
}
}

```

Nie musisz zmieniać istniejącego kodu ani nawet rozumieć go, wyjątek stanowi kod żądania HTTP, który został napisany wcześniej. Pamiętaj, że jego celem jest enkapsulacja żądań do Graph API.

Teraz wróć do CustomGraphContainerController.as i usuń cały kod związany z żądaniem:

```

package controllers
{
    import ui.GraphControlContainer;
    public class CustomGraphContainerController extends GCController
    {
        public function CustomGraphContainerController(a_graphControlContainer:
        ↳GraphControlContainer)
        {
            super(a_graphControlContainer);
        }
    }
}

```

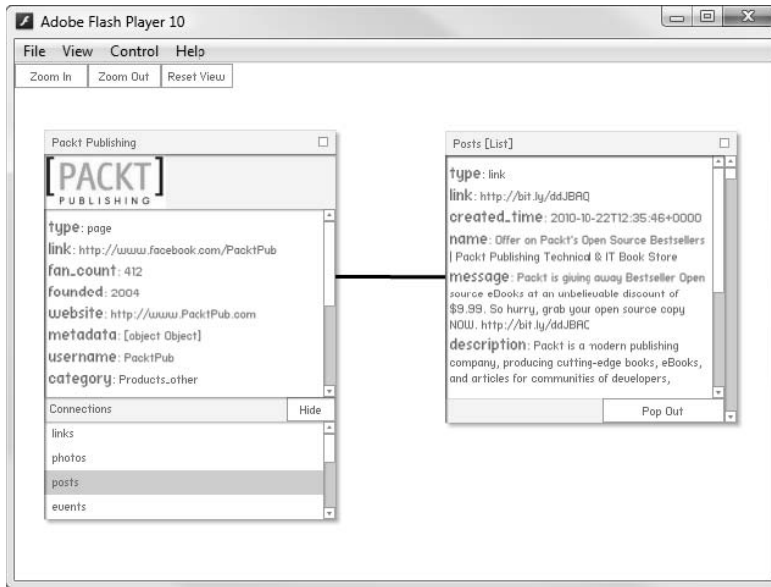
CustomGraphContainerController dziedziczy zmienną typu protected o nazwie _requestor po klasie IRequestor, a także metodę służącą do dodawania potrzebnych detektorów zdarzeń, pozostaje więc wprowadzenie tylko następujących zmian:

```

package controllers
{
    import graph.apis.http.HTTPRequestor;
    import graph.GraphRequest;
    import ui.GraphControlContainer;
    public class CustomGraphContainerController extends GCController
    {
        public function CustomGraphContainerController(a_graphControlContainer:
            ↳GraphControlContainer)
        {
            super(a_graphControlContainer);
            _requestor = new HTTPRequestor();
            addEventListenersToRequestor();
            _requestor.request(new GraphRequest("PacktPub"));
        }
    }
}

```

Skompiluj i uruchom SWE, rozwiń pole *Connections* (połączenia) i kliknij *posts* (posty).



Doskonale! Pojawił się renderer listy Graph, czarna linia łącząca listę ze stroną wskazuje, że istnieje pomiędzy nimi połączenie. Co z innymi połączeniami? Spróbuj kliknąć *statuses* (informacje o statusach).

Error #2044: Unhandled ioError:.text=Error #2032: Stream Error.
URL:https://graph.facebook.com/204603129458/statuses?metadata=1

Niedobrze.

Co się zdarzyło?

Jeśli wpiszesz kłopotliwy URL w przeglądarce (<https://graph.facebook.com/packtpub/statuses>), zobaczysz następujący komunikat:

```
{
  "error": {
    "type": "OAuthAccessTokenException", message: "An access token is
    required to request this resource."
  }
}
```

Błąd wystąpił, ponieważ nie byłeś zalogowany na Facebooku ze swojego SWF. W następnym rozdziale piszę, jak rozwiązać ten problem.

Na razie możesz obejść błąd, dodając detektor zdarzenia `IO_ERROR` do `URLLoader`. W `HTTPRequestor.as` zmodyfikuj metodę `request()`:

```
public function request(a_request:GraphRequest):void
{
    var loader:URLLoader = new URLLoader();
    var urlRequest:URLRequest = new URLRequest();
    var variables:URLVariables = new URLVariables();

    //tworzymy URL z parametrów GraphRequest
    urlRequest.url = "https://graph.facebook.com/" + a_request.objectID;
    if (a_request.connectionID)
    {
        urlRequest.url += "/" + a_request.connectionID;
    }
    variables.metadata = 1;
    urlRequest.data = variables;

    //służy do ustalania, który GraphRequest utworzył określony
    //URLLoader w czasie dalszego działania
    _requests[loader] = a_request;
    loader.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
    loader.load(urlRequest);
}
```

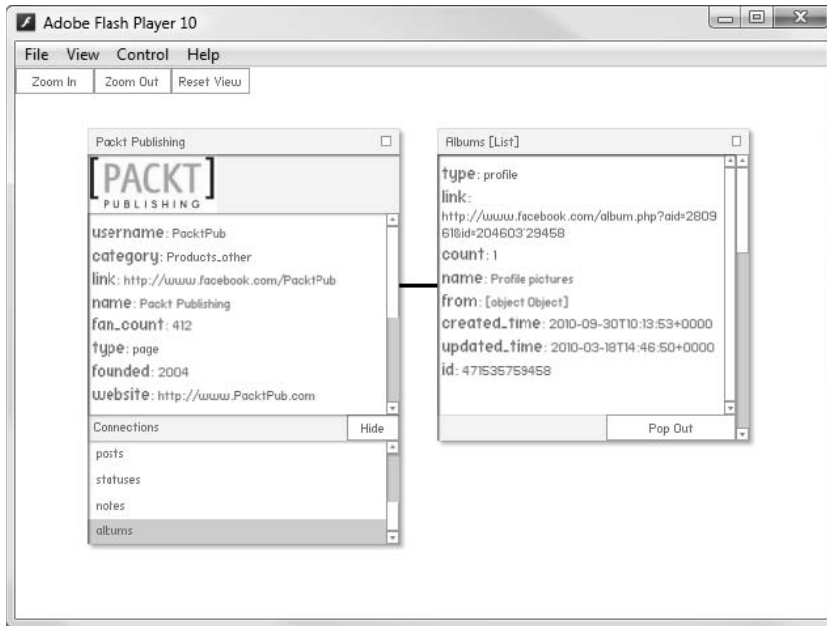
Musisz zaimportować klasę `flash.events.IOErrorEvent`. Następnie w tej samej klasie utwórz prostą funkcję obsługującą zdarzenie, aby wyświetlić zawartość błędu:

```
private function onIOError(a_event:IOErrorEvent):void
{
    trace(a_event.text);
}
```

W ten sposób zobaczysz informację o błędzie w oknie wyjścia, ale błąd nie spowoduje zawieszenia SWF. Uwaga: blok `try catch` nie będzie działał dla błędów tego rodzaju.

Połączenia połączeń

Przyjrzyj się rendererowi listy Graph utworzonemu po kliknięciu połączenia *albums*.



Czy czegoś brakuje?

Brakuje zdjęć! Po wczytaniu strony Packt Publishing na Facebooku możemy zobaczyć dużo zdjęć, ale tutaj nie ma dla nich adresów URL. Sprawdź to poprzez załadowanie listy Graph w przeglądarce; nawet z flagą `?metadata=1` nic nie wskazuje, gdzie mogą być zdjęcia:

```
{
  "data": [
    {
      "id": "471535759458",
      "from": {
        "name": "Packt Publishing",
        "category": "Products_other",
        "id": "204603129458"
      },
      "name": "Profile pictures",
      "link": "http://www.facebook.com/album.php?aid=280961&id=204603129458",
      "count": 1,
      "type": "profile",
      "created_time": "2010-09-30T10:13:53+0000",
      "updated_time": "2010-03-18T14:46:50+0000"
    }
  ],
}
```

```

{
  "id": "307932939458",
  "from": {
    "name": "Packt Publishing",
    "category": "Products_other",
    "id": "204603129458"
  },
  "name": "Books",
  "description": "Packt Books",
  "link": "http://www.facebook.com/album.php?aid=180619&id=204603129458",
  "count": 32,
  "type": "normal",
  "created_time": "2010-02-04T12:32:17+0000",
  "updated_time": "2010-03-18T16:08:42+0000"
},
"paging": {
  "previous": "https://graph.facebook.com/204603129458/albums?metadata=1&limit=25&since=2010-09-30T10%3A13%3A53%2B0000",
  "next": "https://graph.facebook.com/204603129458/albums?metadata=1&limit=25&until=2010-02-04T12%3A32%3A16%2B0000"
}
}

```

Czas zacząć działać — ładowanie zdjęć z albumu

Zobaczyłeś jednak, że każdy obiekt wewnątrz tablicy data jest obiektem Graph. Przyjrzyj się albumowi Packt Books (ma ID 307932939458) poprzez wpisanie w przeglądarce <https://graph.facebook.com/307932939458?metadata=1>:

```

{
  "id": "307932939458",
  "from": {
    "name": "Packt Publishing",
    "category": "Products_other",
    "id": "204603129458"
  },
  "name": "Books",
  "description": "Packt Books",
  "link": "http://www.facebook.com/album.php?aid=180619&id=204603129458",
  "count": 32,
  "type": "album",
  "created_time": "2010-02-04T12:32:17+0000",
  "updated_time": "2010-03-18T16:08:42+0000",
  "metadata": {

```

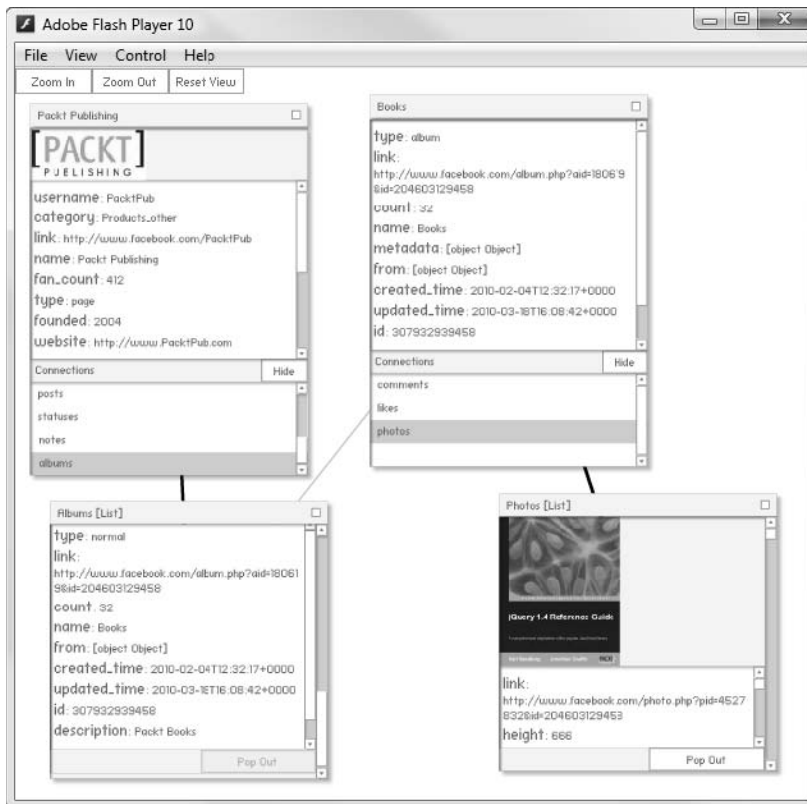
```

"connections": {
  "photos": "https://graph.facebook.com/307932939458/photos",
  "likes": "https://graph.facebook.com/307932939458/likes",
  "comments": "https://graph.facebook.com/307932939458/comments"
},
}
}

```

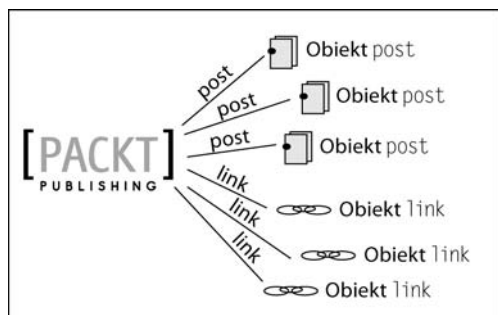
Tym razem metadane podały potrzebne informacje. Zdjęcia połączone są z obiektem Graph album za pomocą połączenia photos.

Uruchom SWF i ponownie załaduj połączenie albums. W rendererze przejdź do obiektu Graph o nazwie Books i kliknij *Pop Out*. Następnie rozwiń pole *Connections* renderera Books i kliknij *photos*.

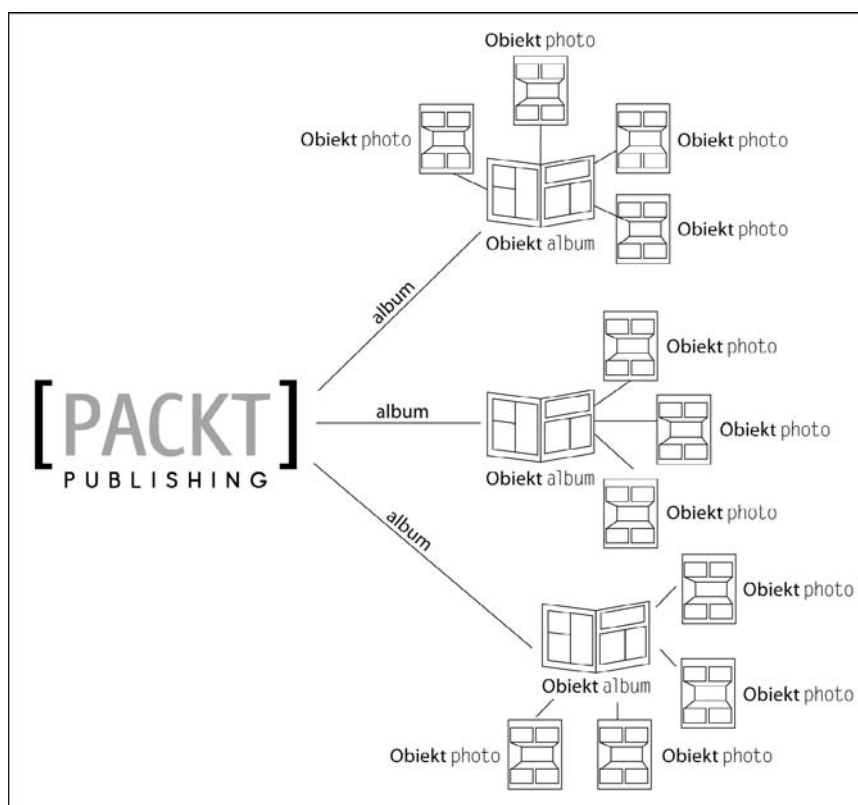


Co się zdarzyło?

Kiedy braliśmy pod uwagę wyłącznie połączenie posts, nasz graf był bardzo prosty: pomiędzy stroną i każdym elementem z nią związanym istniało tylko jedno połączenie.

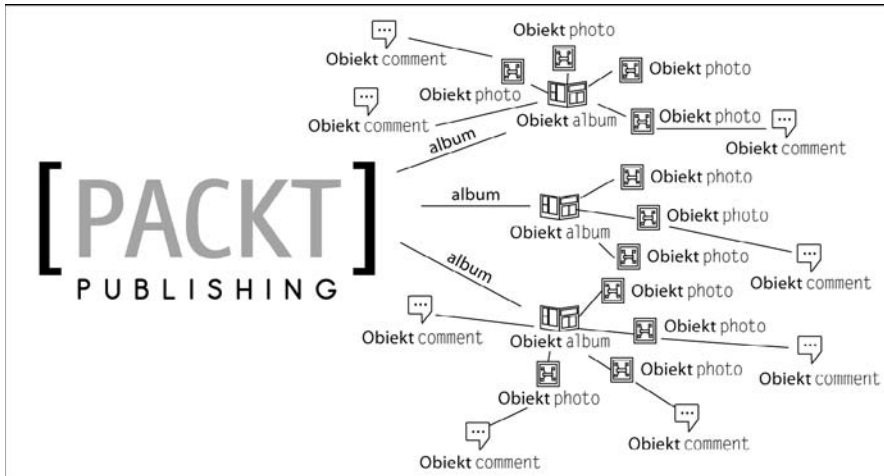


Teraz, kiedy dodaliśmy połączenie albums, wszystko stało się bardziej skomplikowane.



Musimy przemieszczać się po dwóch poziomach połączeń, aby przejść od strony do obiektów, których szukamy.

To nie koniec połączeń. Zarówno albums, jak i photos mogą łączyć się z comments.



Poza tym, każdy komentarz ma właściwość `from`, która łączy go z użytkownikiem będącym jego autorem. Użytkownik może być także „oznaczony” jako pojawiający się na zdjęciu, co łączy zdjęcie z użytkownikiem.

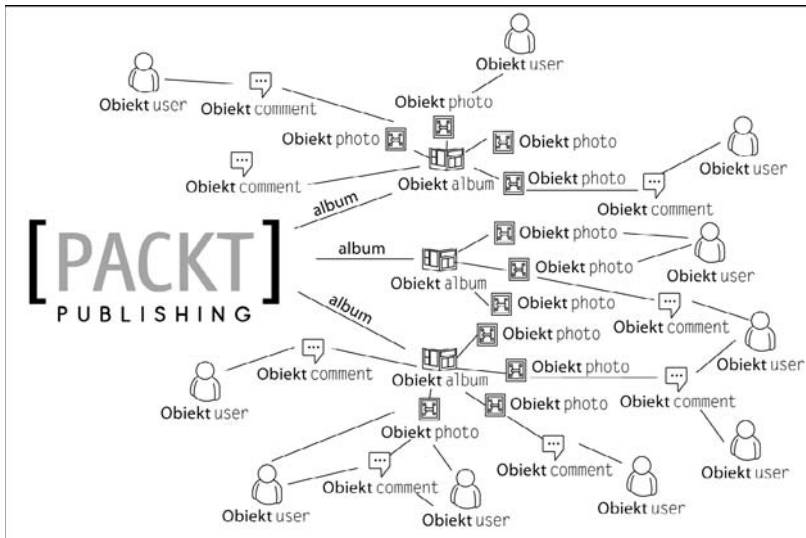
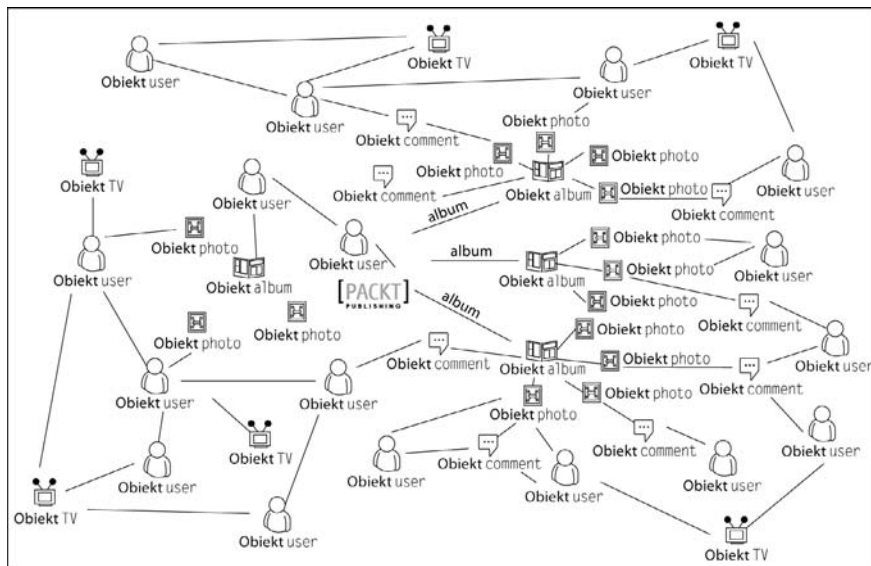


Diagram sprawia wrażenie coraz bardziej skomplikowanego (podobieństwo do matematycznego grafu jest już łatwo dostrzegalne). Oczywiście teraz, kiedy mamy do czynienia z wieloma ludźmi, liczba połączeń staje się ogromna. Użytkownicy mogą być połączeni z dowolnym innym obiektem poprzez bycie znajomym innego użytkownika, oznaczenie na zdjęciu, wideo lub notatce, poprzez zamieszczanie komentarza, łączy lub innego elementu lub kliknięcie *Lubię to!* dla dowolnego innego elementu na Facebooku.



Jeśli rozpoczniesz od określonej strony i przejdziesz przez wszystkie obiekty, z którymi jest połączona, a następnie przez połączone z tymi obiektami kolejne i tak dalej, będziesz mógł przejść przez ogromną liczbę wierzchołków bez konieczności zaczynania na nowej stronie.

Siłą Graphu jest elastyczność. Każdy rodzaj obiektu Graph ma taką samą strukturę jak każdy inny, z wyjątkiem list Graph, które mogą zawierać tablice obiektów Graph. Właśnie dlatego nasz renderer obiektów Graph może łatwo wyświetlać dowolny rodzaj obiektu Graph.

Na pewno zauważyłeś, że nie tylko komentarze mają właściwość `from`? Mają ją także albumy, pojedyncze zdjęcia i niemal wszystkie rodzaje obiektów, które nie są stroną lub użytkownikiem. Oznacza to, że możesz zacząć od dowolnego obiektu, znaleźć jego twórcę i przesuwać się coraz dalej na zewnątrz po grafie z tego punktu.

Model Graphu ma określone skutki dotyczące prywatności. Przypuśćmy, że otrzymaliśmy dostęp do informacji o stronie, dzięki czemu będziemy mieli dostęp do informacji o dowolnym obiekcie na tej stronie. W takim przypadku będziemy mogli przejść z tej strony:

- do albumu umieszczonego na tej stronie i dalej,
- do zdjęcia w tym albumie i dalej,
- do użytkownika oznaczonego na tym zdjęciu i dalej,
- do listy postów na tablicy tego użytkownika i dalej,
- do komentarza do jednego z tych postów umieszczonego przez znajomego pierwszego użytkownika i dalej,
- do użytkownika, który zamieścił ten komentarz, i dalej,
- do programu TV, który ten użytkownik lubi, i dalej,
- do łącza umieszczonego na stronie tego programu TV i dalej,
- do użytkownika, który umieścił to łącze.

I tak dalej. Nic dziwnego, że Facebook korzysta z bardzo szczegółowego zestawu reguł określającego zarówno to, do czego może mieć dostęp użytkownik, jak i to, do czego może mieć dostęp działająca w jego imieniu aplikacja. Reguły te objaśnię w kolejnym rozdziale.

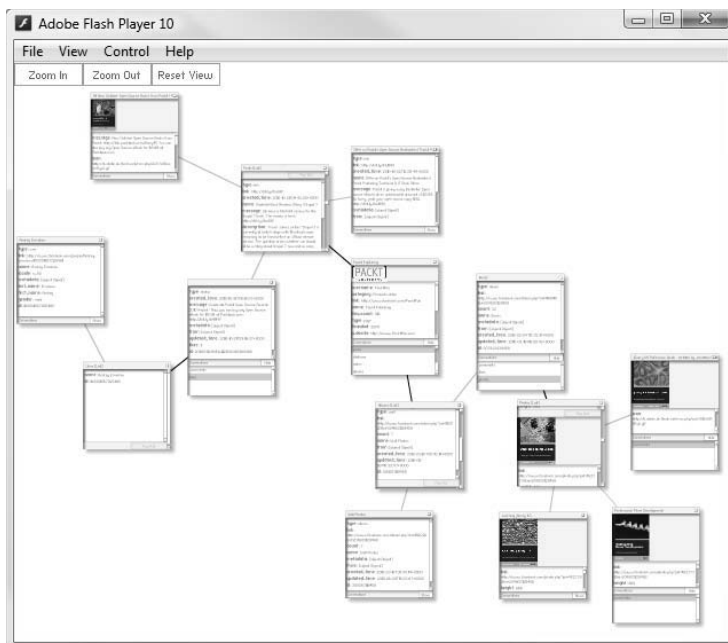
Wszystko razem

Na zakończenie zobaczymy, jak daleko możemy przejść po grafie, zaczynając od strony Packt Publishing.

Czas zacząć działać — przemieszczanie się w Graph

Ustaw aplikację Visualizer tak, aby żądała strony PacktPub. Następnie skompiluj i uruchom SWE, użyj pola *Connections* (połączenia) i przycisków *Pop Out* (wyskoż) do obejrzenia Graphu — zobacz, jak daleko możesz dojść. Nie zapomnij, że możesz przeciągać renderery i zmniejszać widok, aby zmieścić ich więcej w oknie odtwarzacza Flash Player! Pamiętaj, że czarne linie oznaczają połączenia, a szare, że obiekt należy do listy.

Na rysunku niżej pokazano, jak może wyglądać aplikacja po kilku kliknięciach.



Można już dostrzec podobieństwo do rozbudowanych diagramów grafu pokazanych wcześniej w tym rozdziale.

Co się zdarzyło?

Napisałeś kod dla RIA, który pozwala na badanie całego publicznie dostępnego Graphu. Rozpocząć możesz w dowolnym punkcie. Innymi słowy, utworzyłeś we Flashu szperacza dla Facebooka.

Do ćwiczenia — badanie innych obszarów

Nie musisz zaczynać podróży od strony PacktPub. Spróbuj zmienić początkową instancję Graph ↪Request, tak aby zażądała strony Facebooka — Facebook — lub publicznego profilu Marka Zuckerberga — markzuckerberg — lub innej strony znanego zespołu, firmy lub osoby.

Uświadom sobie, że nie jesteś ograniczony do pojedynczego GraphRequest, możesz utworzyć ich tyle, ile będziesz chciał. Spróbuj utworzyć kilka jednocześnie i zobacz, czy ich efekty będą się pokrywać!

Przyjrzyj się w oknie wyjścia informacjom mówiącym, że nie dało się uzyskać obiektu lub listy Graph. Czy przyczyna jest zawsze taka sama?

Quiz

1. Co oznacza parametr `?metadata=1` użyty w Graph URL?
 - a. Sprawia, że metadane są widoczne.
 - b. Sprawia, że metadane są niewidoczne.
2. Ile poziomów możesz przejść w Graphie, gdy zaczniesz od dowolnej strony?
 - a. Jeden.
 - b. Dwa.
 - c. Dziesięć.
 - d. Nieskończenie wiele.
3. To prawda czy fałsz: jeśli JSON zwrócony z Graph URL zawiera obiekt `data`, to zawsze możemy przyjąć, że jest to lista Graph?
 - a. Prawda.
 - b. Fałsz.
4. To prawda czy fałsz: jeśli JSON zwrócony z Graph URL nie zawiera obiektu `data`, to zawsze możemy przyjąć, że jest to obiekt Graph?
 - a. Prawda.
 - b. Fałsz.

Podsumowanie

W tym rozdziale nauczyłeś się wiele o Graph API; dowiedziałeś się nie tylko tego, czym jest, ale także jak używać go z AS3.

Oto najważniejsze informacje do zapamiętania.

- Graph API nosi taką nazwę, ponieważ prezentuje wszystkie dane Facebooka w postaci wielkiego grafu z obiektami i połączeniami.
- Graph API zawiera dwa rodzaje elementów: obiekty Graph i listy Graph.
- Obiekty Graph mają dwa ID: numeryczny wskazany przez Facebook oraz (jest to tylko możliwość) alfanumeryczny wskazany przez właściciela obiektu.
- Obiekty Graph mają połączenia; połączenia prowadzą do list Graph, listy Graph zawierają obiekty Graph.
- Format Graph URL służący do uzyskiwania obiektów Graph jest następujący:
https://graph.facebook.com/graph_object_id.
- Format Graph URL służący do uzyskiwania list Graph jest następujący:
https://graph.facebook.com/graph_object_id/connection_id.
- Adresy Graph URL zwracają dane w formacie JSON. Jest to format tekstowy wykorzystujący pary klucz-wartość do reprezentowania obiektów zawierających właściwości, tablice i inne obiekty.
- Czasem Graph URL zwracają komunikaty o błędach, one także używają formatu JSON.
- Możemy użyć parametru `?metadata=1` w Graph URL do spowodowania, aby zwrócił dodatkowe informacje o elemencie, np. listę połączeń wychodzących z obiektu Graph.
- Dla obiektów Graph, które są częścią list Graph, metadane nie są zwracane.
- Reprezentacja w JSON obiektu Graph może zostać poddana deserializacji na obiekt AS3, którego możemy użyć w kodzie za pomocą biblioteki `as3corelib`.

Mówiłem także o tym, dlaczego Graph API jest tak elastyczny — używa tej samej struktury dla każdego rodzaju obiektu w bazie danych Facebooka.

Jednak co zrobić z obiektami, które zwróciły błąd autoryzacji, kiedy próbowaliśmy uzyskać informacje na ich temat? Zajmiemy się nimi w kolejnym rozdziale.

Skorowidz

A

About, 268, 271
access_token, 84
accessToken, 182
actions, 191, 198
ActionScript, 93
 API, 260
Add to my Page, 257
Adobe
 Social, 275
 ActionScript 3 SDK for
 Facebook Platform, 113
 SDK dla platformy
 Facebook, 27
adres URL, 79
Advanced, 267, 268
agent użytkownika, 79
 porządek działania, 83
AIR, 207, 261, 264
 autoryzacja, 261
 dla systemu Android, 265
 autoryzacja, 265
aktualizacje w czasie
 rzeczywistym, 276
aktualności, 166
album, 215
 dodanie zdjęcia, 216
albums, 61, 62
alert(), 28
all.js, 113
ALL_FRIENDS, 202
ampersand, 80
Android autoryzacja, 265

API, *Patrz* Graph API
Key, 76
klucz, 76, 114
URL, 227
aplikacja
 autoryzacja, 74
 desktopowa, 84
 identyfikator, 76
 klucz sekretny, 76
 rejestracja, 75
 uwierzytelnianie
 AS3, 85
 JavaScript, 92–99
 w aplikacji, 90
 AppData, 278
 Application
 ID, 76
 Profile, 270
 Secret, 76
 Array.join(), 103
 AS3, 34
 escape(), 131
 Facebook SDK, 106
 URLVariables, 142
 as3corelib, 27, 38
 ASP.NET, 276
 attribution, 196
 autoryzacja, 69, 74
 aplikacji, 74
 Graph URL, 115
 użytkownika, 74
 w systemie Android, 265

za pomocą
 AIR i HTTP, 261
 AIR i SDK, 264

B

Basic directory information, 73
baza danych, 224
błąd, 278
 system śledzenia, 278
 wykrywanie, 27, 184
bogata aplikacja internetowa,
 Patrz RIA
Bookmark URL, 254, 268
browse(), 216

C

Canvas
 Page, 253
 Type, 253
 URL, 253
caption, 191, 192
center, 152, 176
checkin, 152
checkins, 217
comments, 62, 210
COMPLETE, 42
connectionID, 54
Connections, 52
contentLoaderInfo, 42
coordinates, 218
currying, 111
CUSTOM, 202

D

dane

model

Graphu, 226

obiektowy, 225

relacyjnej bazy danych, 226

reprezentacja, 226

uwierzytelniające, 74

data, 44

Date, 85

De MonsterDebugger, 28

Deauthorize Callback, 267

Delete, 204

DELETE, 207

deleteGraphObject(), 205

description, 191, 192, 215

deserializacja, 67

JSON, 38

detektor zdarzenia, 58, 184, 261

Developer, 76, 253

Developers, 268

DialogEvent, 27, 183

DisplayObject, 42, 261

distance, 152, 176

Distractor, 274

domena nazwa, 23

dostęp do danych bitowych

obrazu, 195

dynamic, 40

E

Edit URL, 256

edytor HTML, 23

edytuj adres URL, 256

end_time, 214

enkapsulacja żądań, 56

event, 152

EVERYONE, 202

expires_in, 84

ExtendedPermissions.as, 104

ExternalInterface, 93, 97, 115

F

Facebook, 17

Chat API, 275

Credits, 277

Indie Games, 278

Integration, 256, 268

Integrations, 254

konto, 21

Messages, 20

Places, 20, 217

Query Language, *Patrz* FQL

rejestracja, 21

zasady, 273

Facebook.api(), 109, 188

Facebook.login(), 113

Facebook.postData(), 190

FarmVille, 21

FBJSBridge.js, 113

FBML, 252, 255

feed, 120, 137, 178

FileReference, 216

filtrowanie

wyników wyszukiwania, 176

list Graph za pomocą UI,

133

Flash, 42

Tracer for Firebug, 28

flash.display.Loader, 42

FQL, 224, 248

ograniczenia, 243

friends_interests, 102

FRIENDS_OF_FRIENDS,

202

from, 63, 64

G

główny ID tablicy, 234

gra Flash integracja z

Facebookiem, 260

Graffiti, 270, 271

Graph

API, 21, 30, 31, 34, 35, 44,

45, 67

elastyczność, 67

lista, 46, 49, 67

liczba obiektów, 118

połączeń obiektu, 51

stronicowanie, *Patrz*

stronicowanie

żądanie większej liczby

obiektów, 121

obiekt, 67

URL, 32, 67, 115

dostęp do profilu, 73

GraphRequest, 54

group, 152

grupa przynależność, 70

H

hash, 261

hiperłącze, 200

host www, 22

hosting www, 22

HTMLLoader, 261

HTTP GET, 181

HTTPStatusEvent, 184

I

icon, 196

Icon, 268

ID, 34, 67

aplikacji, 114

tablicy główny, 234

żądanie kilku

jednocześnie, 138

identyfikator aplikacji, 76

ids, 139

IFrame, 252

Size, 254

IMDB, 258

IN, 239

incognito, 70

indeks, 243

informacja o statusie, 190

informacje profilowe, 73

initComplete(), 107

InPrivate, 71

Inside

Facebook, 278

Social Games, 278

Virtual Goods, 278

InsideNetwork, 278

Insights, 275

intelligentne

sortowanie, 145

wyszukiwanie, 144

Interests, 276

Internationalization API, 275

IO_ERROR, 58

J

JavaScript
 Object Notation, *Patrz* JSON
 JavaScript SDK, 274
 JSON, 37, 67, 198, 228
 deserializowanie, 38
 serializowanie, 38
 JSON.encode(), 218

K

kanwa, 253
 karta prywatna, 71
 katalog aplikacji Facebooka, 272
 klient FTP, 23
 klucz
 API, 76, 114
 obcy, 234
 podstawowy tabeli, 234
 sekretny, 76, 114
 kod
 UI, 35
 żądania, 53
 HTTP, 56
 kodowanie URL, 131
 kolumna, 225
 komentarz, 210
 konto, 21
 przeglądanie, 70
 kredyty Facebooka, 277

L

latitude, 218
 length, 120
 licznik, 268
 limit, 121, 125, 127, 141
 LIMIT, 247
 link, 191
 linkName, 192
 linkURL, 192
 lista
 Graph, 46, 49, 67
 liczba obiektów, 118
 renderer, 46
 stronicowanie,
 Patrz stronicowanie
 żądanie większej liczby
 obiektów, 121

połączeń obiektu Graph, 51
 uprawnień rozszerzonych
 użytkownika, 241
 LiveDocs, 24
 load(), 36
 Loader, 42
 locale, 157, 176
 loginComplete, 109
 Logo, 268
 longitude, 218
 lower(string), 247
 Lubię to!, 211
 usuwanie, 212

Ł

ładowanie zdjęć z albumu, 61

M

me(), 247
 message, 180, 182, 191, 211,
 213, 216, 218
 metainformacja, 43
 metoda żądania, 181
 MinimalComps, 27
 Mochi
 Media, 260
 Social Platform, 260

N

name, 191, 214, 215
 nazwa
 domeny, 23
 tabeli, 256
 NETWORKS_FRIENDS, 202
 NO_FRIENDS, 202
 notacja krótkiej łapki, 235
 notatki, 212
 now(), 247

O

obiekt
 Graph, 67
 lista właściwości, 210
 publikowanie, 210

złożony, 140, 141, 142
 żądający, 53
 objectID, 54
 odmowa dostępu, 114
 offline_access, 105
 offset, 125, 127, 141
 OFFSET, 247
 ograniczenie dostępu, 71, 73
 Graph API, 73
 okno prywatne, 71
 onDeleteComplete(), 206
 onRedirect(), 262
 Open Graph, 276
 operacja przypisana do
 aplikacji, 200
 operator w FQL
 logiczny, 245
 porównania, 245
 ORDER BY, 246
 ownerID, 197

P

Packt Publishing, 32
 Page, 252
 Tabs, 256
 PageData, 278
 paging, 44, 142
 pamięć podręczna
 czyszczenie, 29
 pełne wyszukiwanie, 146
 perms, 113
 photos, 61, 62
 PHP, 276
 picture, 191
 pictureURL, 192
 place, 218
 plik
 regul
 międzydomenowych, 42
 SWF, 28, 42
 pobieranie
 informacji, 226
 z tablicy, 226
 ze strony, 35
 list ID znajomych
 w FQL, 238

podstawowe informacje
 profilowe, 73
 podwęzeł, 140
 podzapytanie, 240
 pole, 225
 indeksowane, 244
 połączenie, 67
 odzworowanie w FQL, 230
 Pop Out, 49
 post, 45
 na tablicy, 137
 niesformatowany, 196
 sformatowany, 191
 umieszczanie na dowolnej
 tablicy, 196
 usuwanie
 aplikacją, 204
 poprzez interfejs
 Facebooka, 204
 POST, 181, 182, 197
 posts, 45
 Preview my Profile, 73
 privacy, 191, 198, 200
 Privacy Settings, 89
 profil, 21, 32, 70
 aplikacji, 270
 osobisty, 32
 publiczny, 32
 protokół OAuth 2.0, 83
 proxy PHP, 195
 prywatność posta, 201
 ALL_FRIENDS, 202
 CUSTOM, 202
 EVERYONE, 202
 FRIENDS_OF_FRIENDS,
 202
 NETWORKS_FRIENDS,
 202
 przeglądanie prywatne, 71
 przemieszczanie się
 w Graph, 65
 przyznawanie uprawnień
 rozszerzonych, 186
 publikowanie
 na innych tablicach, 196
 aplikacja, 197
 na tablicy użytkownika, 178,
 181, 178–88

wymagane parametry
 i uprawnienia, 220
 za pomocą
 Facebook.api(), 188
 Facebook.postData(), 190
 HTTP, 210, 219
 SDK, 188, 220
 Publish, 178, 210, 216
 publish(), 179, 197, 210, 216
 publish_checkins, 218
 publish_stream, 186, 197, 209,
 211, 213, 215, 217
 PublishingCapabilities, 178
 PublishObject, 180, 197

R

ramka, 252
 rand(), 247
 rejestracja, 21
 Remove, 204
 renderer, 35, 41, 51
 listy Graph, 46
 obiektu Graph,
 Patrz renderer
 reprezentacja danych, 226
 requestor, 53
 Reset View, 35
 rest parameter, 103
 RIA, 18
 Rozmiar IFrame, 254
 rozwijanie funkcji, 111
 RVSP, 214
 rzutowanie, 161

S

Sandbox Mode, 268
 scope, 101
 SDK, 107, 274
 SDKRequestor, 110
 SDKRequestor(), 107
 Search, 143
 SELF, 202
 serializowanie, 38
 serwer www, 21
 setAccessToken(), 96, 97
 Sharing on Facebook, 73

sieć
 przynależność, 70
 regionalna, 70
 społecznościowa, 17
 since, 131, 142
 Site URL, 79
 Social, 275
 SOME_FRIENDS, 203
 sortowanie, 246
 inteligentne, 145
 SOS max, 28
 source, 191, 198, 203, 216
 SQL, 224
 StageWebView, 265
 start_time, 214
 status, 190
 statuses, 57
 strip_tags(), 247
 strlen(), 247
 strona, 141
 kanwy, 253
 społecznościowa, 32, 230
 z opakowaniem, 113
 zakładka, 256
 zwrotna, 94
 stronicowanie, 124, 141
 na podstawie
 daty, 133, 138, 247
 liczb, 247
 za pomocą
 limit i offset, 128
 since i until, 131
 URL, 130
 strpos(), 247
 subject, 213
 substr(), 247
 SWF, 22, 115
 SWFObject, 24
 system śledzenia błędów, 278

T

Tab
 Name, 256
 URL, 256
 tabela, 224
 łącza, 237

tablica
 publikowanie wpisów, 178,
 181, 178–88
 znajomego, 21
 thread, 277
 token dostępu, 74, 78, 79, 81,
 96, 114, 261, 263
 odbieranie, 96
 permanentny, 105
 wygaśnięcie, 84
 informacje, 84
 tryb przeglądania
 prywatnego, 70
 Tutorial, 271
 twórca aplikacji, 76
 typ kanwy, 253
 type, 149, 176
 wartości, 152

U

udostępnianie
 aplikacji, 268
 na Facebooku, 73
 UI, 35, 133
 filtrowanie list Graph, 133
 umieszczanie aplikacji online,
 252
 until, 129, 142
 upper(string), 247
 uprawnienia, 69, 71, 114, 241
 grupy, 71
 informacje, 242
 rozszerzone, 101, 105, 115
 przyznawanie, 186
 użytkownika, 102, 241
 znajomych, 102
 żądanie, 102, 113
 użytkownika, 71
 URI, 80
 URL
 kanwy, 253
 karty, 256
 zakładki, 254
 URLLoader, 36, 53, 216
 URLRequest, 36, 42
 URLVariables, 51
 user, 153
 user_checkins, 218

user_interests, 102
 user_photos, 215, 217
 ustawienia prywatności, 71, 89
 usuwanie
 oznaczenia Lubię to!, 212,
 220
 przez interfejs Facebooka,
 204
 w AIR, 220
 za pomocą
 aplikacji, 204
 HTTP, 220
 SDK, 220
 uwierzytelnianie, 114, 259
 aplikacji, *Patrz* aplikacja
 uwierzytelnianie
 użytkownika metodą
 Facebook.login(), 108
 użytkownik testowy, 277

V

Visualizer, 27
 Vizzy Flash Tracer, 28

W

WHERE, 239, 244
 wiersz, 225
 wizualizowanie informacji, 40
 wydarzenia, 213
 wykonanie zapytania, 227
 wykrywanie błędów, 27, 184
 wysyłanie za pomocą
 HTTP, 220
 SKD, 220
 wyszukiwanie, 158
 bez tokenu dostępu, 176
 filtrowanie wyników, 176
 inteligentne, 144
 na podstawie zapytania, 162
 ograniczenia, 156
 opcje, 146, 147
 pełne, 146, 156
 Posts by everyone, 156
 Posts by friends, 146, 156
 po dokonaniu autoryzacji,
 154, 176
 szablonowe, 145

w Twoich aktualnościach,
 165, 166, 176
 wśród postów znajomego,
 165, 167, 168, 177
 wymagające uprawnień
 rozszerzonych, 176
 za pomocą
 aplikacji, 170
 Graph API, 149
 Graph URL, 153, 156,
 165, 176
 SDK, 162
 wywoływanie
 kilku zapytań
 jednocześnie, 248

X

XML, 228

Z

zabezpieczenia, 69,
Patrz ograniczenie dostępu
 zakładka, 256
 Zakładki strony, 256
 zameldowania, 217
 zapytanie, 227
 zdjęcie, 216
 ładowanie do albumu, 61
 profilowe, 115
 znacznik, 255
 Zoom
 In, 35, 41
 Out, 35
 Zuckerberg Mark, 32
 zwrotna strona, 94

Ż

żądający obiekt, 53
 żądanie, 53, 54
 kilku ID jednocześnie, 138
 metoda, 181
 uprawnień rozszerzonych,
 102
 za pomocą SDK, 113

Facebook Graph API

Tworzenie rozbudowanych rozwiązań we Flashu

Ponad 500 milionów użytkowników, spędzających 700 000 000 000 minut miesięcznie na jednej stronie. O czym mowa? O serwisie Facebook! Jego potencjał doceniła niejedna firma. Zastanawiasz się, jak dołączyć do tego grona? Jak zdobyć popularność, fanów i być może zarobić? Uwierz, że to nic trudnego! Facebook udostępnia bogate API, dzięki któremu bez problemu zintegrujesz się z witryną facebook.com.

W tej książce wiedza jest na wyciągnięcie ręki. W trakcie lektury nauczysz się korzystać z dostarczonych funkcji przy użyciu ActionScript 3 SDK for Facebook Platform. Dowiesz się, jak zarejestrować swoją aplikację, uwierzytelnić użytkowników oraz żądać uprawnień rozszerzonych. Ponadto zobaczysz, jak korzystać z wyszukiwarki, wykrywać błędy oraz publikować wiadomości na tym portalu społecznościowym. Książka ta jest idealną pozycją dla każdego fana Facebooka posiadającego zacięcie programistyczne. Sprawdzi się także doskonale w rękach profesjonalistów chcących stworzyć nowe narzędzia lub gry dla portalu Facebook. Polub to!

Dotrzyj do milionów użytkowników serwisu Facebook!

- Zalety i wady Facebooka
- Wybór hostingu WWW
- Dostęp do Graph API przez przeglądarkę
- Pobieranie informacji ze strony za pomocą ActionScript 3
- Rejestracja aplikacji na Facebooku
- Uwierzytelnianie za pomocą ActionScript 3
- Uzyskiwanie rozszerzonych uprawnień
- Pobieranie i stonowanie wyników
- Wykorzystanie wyszukiwarki
- Pisanie w feedzie użytkownika
- Publikowanie sformatowanych postów
- Dodawanie i usuwanie oznaczenia „Lubię to!”

helion.pl
księgarnia
internetowa

W katalogu: 7178



Księgarnia Internetowa
<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/news>

Helion SA
ul. Włodzimieża 5c, 44-100 Gliwice
tel.: 32 230 18 45
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po **WIECEJ**



800 KORZYSCA

ISBN 978-83-246-3381-4



Cena: 47,00 zł

Informatyka w najlepszym wydaniu