**[JSON] Podstawy i struktura formatu JSON**

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiN31znwdtbpQ2cn989UTnFG3GcZBcdKTqaz-6eRKnfpBimh-qZDUGn3S7P_5h6chOl4AIveOTpR0NN6d077lW3O5r9rvxSIO4Q3wQDe33Wjbln-ytPeHHTdg5ac2iwbikOv21ofG1A6L0/s1600/icoJSON.png)

JSON to skrót od JavaScript Object Notation, w wolnym tłumaczeniu Notacja Obiektowa JavaScriptu. Jak sama nazwa wskazuje, format stanowi podzbiór języka JavaScript, w którym to służy on jako wewnętrzny język do opisywania obiektów. Nie ogranicza to bynajmniej pola, na którym można go stosować do samego tylko JavaScriptu, a wręcz przeciwnie - pomimo nazwy format ten jest używany powszechnie do wymiany danych we wszystkich praktycznie językach. Jest absolutnie niezależny od platformy, gdyż jego implementacja moze znaleźć się w praktycznie dowolnej aplikacji. Jest to zatem bardzo uniwersalny format, z powodzeniem wykorzystywany w celu wymiany danych pomiędzy aplikacjami.  
  
JSON jest bardzo lekkim formatem - jego składnia ograniczona jest do minimum, jest to wręcz banalnie prosty, tekstowy format, w którym poza samymi danymi stosuje się raptem kilka innych ograniczników, takich jak przecinek, czy nawiasy. Przez taki minimalizm jest on niestety trochę mniej czytelny dla człowieka, niż np. XML, ale za to nadrabia małym rozmiarem, gdyż nie posiada w sobie żadnych elementów takich jak znaczniki w XML-u. Warto znać JSON-a, gdyż jest on bardzo przydatny w wymianie danych, szczególnie sieciowych. W artykule tym omówimy podstawowe rzeczy, takie jak składnia, a w następnych artykułach zobaczymy jak pracować na danych z JSON-a w językach PHP i Javascript.

**Podstawowe zasady**

W JSON-ie obowiązuje kilka kluczowych reguł jakich należy się trzymać podczas tworzenia pliku JSON:

* każdy dokument powinien być kodowany w UTF-8
* każdy dokument JSON musi zawierać się w głównym nawiasie klamrowym
* każdy dokument musi zawierać jeden główny element, w którym zawiera się cała reszta
* wszystkie nazwy elementów/kluczy i ich wartości objęte muszą być cudzysłowami
* pary klucz-wartość oddzielamy dwukropkiem (**klucz** : **wartość**)
* elementy występujące obok siebie oddzielamy przecinkiem
* zawartość elementu obejmujemy w nawiasy klamrowe
* tablice obejmujemy w nawiasy kwadratowe

**Struktura JSON-a**

Każdy plik JSON musi zawierac się w jednym głównym nawiasie klamrowym, najprostszy, pusty dokument JSON wyglądać będzie więc tak:

1. {


5. }

Dodajmy teraz nazwę dla głównego elementu, który jest wymagany, niech to będzie np. rodzina:

1. {
3. "rodzina": {


7. }
9. }

W JSON-ie każdą definicję obiektu/elementu, która zawierać będzie więcej niż jedno pole obejmujemy w nawias klamrowy.  
Dodajmy teraz jakiś element do naszego głównego elementu, niech to będzie np. pole o nazwie nazwa.  
W JSON-ie wartości dla danych pól podajemy na zasadzie:

1. "klucz" : "wartość"

a w przypadku, gdy mamy do podania wartości kilku pól, to rodzielamy je przecinkiem:

1. {
3. "klucz1" : "wartość1",
5. "klucz2" : "wartość2",
7. "klucz3" : "wartość3"
9. }

Jak widzimy, w przypadku ilości większej, niż jedno pole całość obejmujemy w nawias klamrowy.  
Dodajmy teraz pole nazwa do naszego głównego elementu:

1. {
3. "rodzina": {
5. "nazwa" : "Kowalscy"
7. }
9. }

Powyższy zapis w XML-u wyglądałby tak:

**XML:**

1. <rodzina>
3. <nazwa>Kowalscy</nazwa>
5. </rodzina>

Teraz dodajmy kolejny element o nazwie dziadkowie:

1. {
3. "rodzina": {
5. "nazwa" : "Kowalscy",
7. "dziadkowie" : {


11. }
13. }
15. }

A w nim kolejny element o nazwie dziadek z polami:

* imie
* nazwisko
* oraz element dzieci, w którym znajdą się kolejne elementy:

1. {
3. "rodzina": {
5. "nazwa" : "Kowalscy",
7. "dziadkowie" : {
9. "dziadek" : {
11. "imie" : "Alojzy",
13. "nazwisko" : "Kowalski",
15. "dzieci" : {

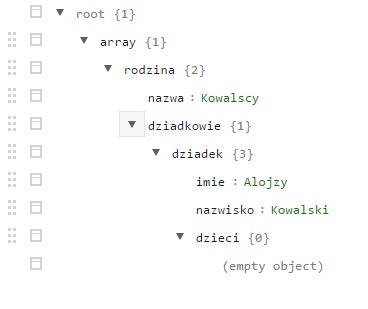

19. }
21. }
23. }
25. }
27. }

Powyższy zapis w XML-u wyglądałby tak:

**XML:**

1. <rodzina>
3. <nazwa>Kowalscy</nazwa>
5. <dziadkowie>
7. <dziadek>
9. <imie>Alojzy</imie>
11. <nazwisko>Kowalski</nazwisko>
13. <dzieci>
15. </dzieci>
17. </dziadek>
19. </dziadkowie>
21. </rodzina>

A w widoku drzewka wygląda to tak:

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgNuT9PbfWB1XPEqBts8sOUtAH1a7I6BAG5fkyN75O5pXyNMyMJWv30tZZFGQZtuPXGDHOk1vj_k8owJQGJUjxmYgoTpVJUtt584qnm9c_xfBgj98I4k_SDvbSCD6lPKBZTWvN83ZzLocM/s1600/d1.jpg)

Jak teraz dodać innego dziadka?  
Zrobienie tego w poniższy sposób będzie błędne:

1. {
3. "rodzina": {
5. "nazwa" : "Kowalscy",
7. "dziadkowie" : {
9. "dziadek" : {
11. "imie" : "Alojzy",
13. "nazwisko" : "Kowalski",
15. "dzieci" : {


19. }
21. `},
23. "dziadek" : {
25. "imie" : "Alfred",
27. "nazwisko" : "Kowalski",
29. "dzieci" : {


33. }
35. }
37. }
39. }
41. }

Otóż musimy naszych dziadków ująć w tablicę, a robimy to za pomocą nawiasów kwadratowych:

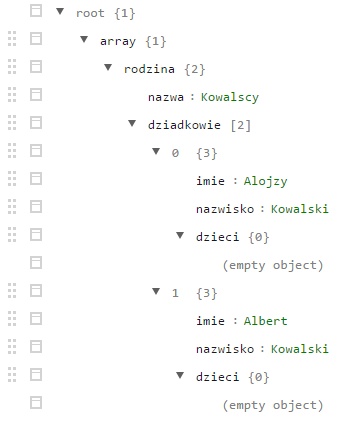
1. {
3. "rodzina": {
5. "nazwa" : "Kowalscy",
7. "dziadkowie" : [
9. {"imie" : "Alojzy", "nazwisko" : "Kowalski", "dzieci" : {} },
11. {"imie" : "Albert", "nazwisko" : "Kowalski", "dzieci" : {} }
13. ]
15. }
17. }

Powyższy zapis w XML-u wyglądałby tak:

**XML:**

1. <rodzina>
3. <nazwa>Kowalscy</nazwa>
5. <dziadkowie>
7. <dziadek>
9. <imie>Alojzy</imie>
11. <nazwisko>Kowalski</nazwisko>
13. <dzieci>
15. </dzieci>
17. </dziadek>
19. <dziadek>
21. <imie>Albert</imie>
23. <nazwisko>Kowalski</nazwisko>
25. <dzieci>
27. </dzieci>
29. </dziadek>
31. </dziadkowie>
33. </rodzina>

A w widoku drzewka prezentuje się tak:

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEj2KSArUeTwB0a1mfAQ6AAbgj5CPKKcnYTWPvYqcllDBmoS3gH8cR7Acc2zSXY_eU6GWX4sWRB7s58cGBEblLF2Py4SoHRy3SPdldSrzxj9e0xyRrVvKfxVRIcuo1kwaUxX_naZrLzP5pE/s1600/d2.jpg)

Jak więc widzimy elementy tablicowe umieszczamy w nawiasach kwadratowych, na zasadzie:

1. "tablica" : [
3. {"elementA1" : "wartośćA1", "elementA2" : "wartośćA2"},
5. {"elementB1" : "wartośćB1",  "elementB2" : "wartośćB2"},
7. ...
9. ]

Następnie dodać możemy kolejne zagnieżdżone elementy, jak ojcowie, synowie, wnukowie itd. Ale z tym już polecam poeksperymentować w ramach pracy domowej i pooglądać sobie wyniki swojej pracy na: <http://codebeautify.org/jsonviewer>

**Wykorzystanie JSON-a**

JSON jest powszechnie wykorzystywany w komunikacji sieciowej. Dane przesyłane JSON-em są najzwyczajniej w świecie przesyłane szybciej, gdyż objętość dokumentu JSON jest mniejsza, niż dokumentu z tymi samymi danymi zapisanymi w formacie XML. Jest w związku z tym również szeroko stosowany podczas przesyania danych AJAX-em. JSON jest ponadto wewnętrznie wykorzystywany przez Javascript jako jego naturalny język zapisu obiektów.  
  
Praca z JSON-em jest wygodna zarówno po stronie Javascriptu, jak i PHP, gdzie możemy dowolnie wymieniać się danymi. Ponadto konwersja pomiędzy JSON-em, a XML-em jest dziecinnie prosta, tak samo jak serializacja i deserializacja obiektów i tablic robiona przy pomocy formatu JSON-a. W następnym artykule poznamy metody służące do pracy z tym formatem, zarówno w PHP, jak i w Javascripcie.