Poznańskie Centrum Superkomputerowo-Sieciowe
ICHB Polska Akademia Nauk

Dokumentacja techniczna dla programistów Wieloformatowych Obiektów Multimedialnych i Interaktywnych (WOMI) na platformie epodreczniki.pl

Platformy Technologicznej
epodreczniki.pl
w wersji 3.0

Poznań, 30 listopada 2015 r.

Spis treści

Historia zmian

Wersja	Autor/autorzy	Opis zmian
1.1-1.6	Andrzej Przybyszewski, Piotr Dziubecki, Tomasz Kuczyński, Krzysztof Kurowski	Podstawowa specyfikacja i obsługa zapisu obrazów z poziomu WOMI i i kontekstu użytkownika
1.7	Andrzej Przybyszewski	Zmiany w API
1.8	Andrzej Przybyszewski	Stabilna wersja API
1.9	Andrzej Przybyszewski	Dodatkowe metody zapisu i odczytu w profilu użytkownika

Wprowadzenie

Głównym celem dokumentacji technicznej dla programistów Wieloformatowych Obiektów Multimedialnych i Interaktywnych (WOMI) jest opis interfejsów programistycznych dla oprogramowania dostarczanego i uruchamianego przez zewnętrznych programistów na platformie epodreczniki.pl.

Komponenty typu WOMI są to elementy e-podręczników składające się z plików pozwalających na zaawansowaną prezentacje tekstową i graficzną w przeglądarce internetowej. Obecnie na platformie wspierane są tylko technologie zgodne z ustalonym i potwierdzonym przez *W3C* otwartym standardem *HTML5*, w szczególności sam język *HTML5, CSS2, CSS3 i JavaScript*.

WOMI nie pozwalają osadzać elementów oprogramowania napisanych w innych niż wyżej wymienionych technologiach, w szczególności: *Adobe Flash, MS Silverligh*, które wymagają dodatkowych komponentów instalowanych w systemie.

Podejścia tworzenia WOMI

Szkielet aplikacyjny WOMI opiera się na bibliotece *require.js*. Biblioteka pozwala na bardzo modularne podejście programistyczne i "opakowanie" części kodu jako niezależne lub zależne moduły. Moduły mogą być przechowywane w osobnych plikach poprzez odpowiednie ich umieszczenie w funkcji nadrzędnej. System importowania zależności modułów zapobiega problemom związanym z brakiem możliwości załadowania danego modułu/biblioteki przed wykonaniem.

W większości specyficznych ustawień/konfiguracji programista tworzący moduł nie musi znać szczegółów biblioteki *require.js*, zaleca się jednak zapoznanie z opisem funkcjonalności rozpoznania i przetworzenia modułów require.js.

Główny moduł powinien spełniać poniższe założenia:

- nadrzędne wywołanie funkcji *define*, która rejestruje to co zwraca funkcja podana w drugim parametrze jako moduł;
- pierwszy parametr funkcji jest to lista zależnych modułów, bibliotek, które mają zostać załadowane przed wykonaniem tego modułu;
- parametry funkcji przekazywane jako drugi parametr *define*, są to kolejne uchwyty do załadowanych bibliotek (wg kolejności, podania ich w zależnościach), mogą być one użyte dalej w ciele funkcji (modułu);
- atrybuty nowo utworzonego modułu:
    - *enableMaximize*: aby WOMI uruchomiło się w kontenerze na całym ekranie (po kliknięciu w obraz zastępczy), wartość tego parametru musi być ustawiona na *true*;

- moduł (czyli ta funkcja) powinna zwracać istotne dla nas (czytnika) informacje czyli "klasę" zawierającą metodę start
    - metoda *start*: zawiera parametry:
        - *placeholder*, będzie do niego przekazany główny węzeł DOM, do którego będzie można dopisywać nowe elementy wg uznania
        - *options*, obiekt posiadający opcje, z którymi zostaje odpalony moduł
        - *width*
        - *height*
        - *methods*:
            - gdy moduł nie posiada zdefiniowanego parametru *enableMaximize* lub *false*, dostępne są metody: *openFullscreen()* i *closeFullscreen()*
            - gdy *enableMaximize == true*, zdefiniowana jest tylko jedna metoda: *closeWomi()*, która zamyka całe WOMI i wraca do stanu pierwotnego
        - *isFullscreen*: parametr mówiący o tym czy womi zostało odpalone w opcji pełnoekranowej, (zawsze *false* przy *enableMaximize*)
        - metoda *clean*: opcjonalna metoda zwalniająca zasoby itp
        - metoda *sizeChange*: opcjonalna metoda, która dostaje jako parametry: *width* i *height* kontenera nadrzędnego, może służyć do podpięcia się na zdarzenie zmiany rozmiaru (metoda zostanie wywołana przed uruchomieniem metody start, a później za każdym razem gdy dojdzie do zmiany rozmiaru kontenera)
        - należy pamiętać by zwracać prototyp funkcji (w paradygmacie obiektowym - klasę), który może być później zinstancjonowany
        - biblioteka *declare.js* (http://doug-martin.github.io/declare.js/) pomaga w tworzeniu klas, ale można to zrobić także tradycyjnymi sposobami.

Dostępne biblioteki programistyczne

Do wykorzystania w "imporcie" w module:

- *'jquery'* - biblioteka *jquery*
- *'jqueryui'* - dodatek do biblioteki *jquery* - mechanizmy interfejsu użytkownika
- *'declare'* - biblioteka *declarejs*
- *'underscore'* - biblioteka *underscore.js*
- *'backbone'* - biblioteka *backbone.js*

- *'domReady'* - plugin *require.js* do wywoływania zdarzeń po załadowaniu DOM
- *'text'* - plugin *require.js* do importowania tekstu
- biblioteki *createjs*: zgodnie z oczekiwaniami partnerów udostępniamy spakowane wersje bibliotek, są to spakowane źródła *createjs* i *movieclip* w odpowiednich wersjach wg: http://code.createjs.com/
- poniżej przedstawiono listę mapowań *requirejs* dla tych bibliotek, należy używać konkretnego *movieclip* tylko z konkretną biblioteką *createjs* (jeśli potrzebne):
    - 'epo.createjs.2013.02.12' oraz 'epo.createjs.movieclip.0.6.0'
    - 'epo.createjs.2013.05.14' oraz 'epo.createjs.movieclip.0.6.1'
    - 'epo.createjs.2013.09.25' oraz 'epo.createjs.movieclip.0.7.0'
    - 'epo.createjs.2013.12.12' oraz 'epo.createjs.movieclip.0.7.1'

Wspólne pliki

WOMI może składać się z innych modułów, które są importowane w głównym module. Jednak dodatkowe moduły często mogą być ponownie użyte w nowych WOMI, dlatego zalecane by zaimplementować je w formie "biblioteki". W pierwszej kolejności zaleca się implementacje wspólnych modułów w jednym WOMI, a następnie bezpośredni kontakt z partnerem technologicznym w celu sprawdzenia, zatwierdzenia i wgrania wytworzonych bibliotek na serwer statyczny.

WOMI jako HTML w ramce

Istnieje możliwość stworzenia WOMI w oparciu o plik HTML, który zostanie umieszczony na stronie jako ramka *iframe*. W efekcie, takie podejście pozwala programiście na większą dowolność w tworzeniu zaawansowanych animacji oraz interakcji z użytkownikiem. WOMI tego typu powinno się składać z pliku głównego HTML, a w razie potrzeby dodatkowych plików: JS, CSS i wykorzystywanych plików graficznych.

Istnieje możliwość wydzielenia wspólnych plików dla programisty. W celu wykorzystania własnych bibliotek jako osobnych plików HTML należy przygotowaną strukturę wraz z zawartością oraz odpowiednim identyfikatorem nazwy przesłać do partnera technologicznego w celu przeprowadzenia podstawowego audytu.

Schemat dostępu dla partnerów merytorycznych oraz ich podwykonawców wygląda następująco:

```
/global/libraries/<nazwa>/<ścieżka>
```

np. dla pliku, który dla twórcy jest pod ścieżką relatywną: *js/script.js*

i nazwą partnera: *partner1*

```
<script src="/global/libraries/partner1/js/script.js"></script>
```

Interfejs programistyczny (API) dla WOMI

W ogólności interfejs programistyczny (API) ma na celu dostarczyć funkcjonalność pozwalającą na pełną integrację WOMI z platformą epodreczniki.pl. API zostało zaprojektowane ze szczególnym uwzględnieniem dla kontekstu w jakim zostało uruchomione WOMI.

Wymienione poniżej sposoby tworzenia WOMI i integracji przez API mogą być rozszerzane o dodatkowe funkcjonalności wspierane na platformie epodreczniki.pl.

API dla modularnego WOMI

Opis 'reader.api'

'reader.api' jest biblioteką do zaimportowania w module *requirejs*.

Aby zainicjować nowy obiekt API w kontekście danego modułu, należy wykonać *var readerApi = new api(require*); a następnie można używać poniższych metod (relatywna ścieżka zaczyna sie od ./):

metoda	parametry	opis
getFullPath	path	metoda przyjmuje relatywną ścieżkę względem aktualnego modułu (pliku js) z katalogu WOMI, i zwraca pełną ścieżkę
loadCss	path	metoda ładuje plik CSS z relatywnej ścieżki
setUserVar	varName, value	ustawienie zmiennej wykorzystywanej przez aplikację (w kontekście użytkownika, kolekcji, modułu i womi) (zapisuje do bazy danych)#aktualnie to co local
getUserVar	varName, callback	pobranie zmiennej ustawionej powyższą metodą (zapisuje do bazy danych)#aktualnie to co localwartość jest zwracana w funkcji wywołania zwrotnego
setLocalUserVar	varName,	ustawienie zmiennej wykorzystywanej przez aplikację (w kontekście użytkownika, kolekcji, modułu i womi)

metoda	parametry	opis
	value	(zapisuje do localStarage)
getLocalUserVar	varName	pobranie zmiennej ustawionej powyższą metodą (zapisuje do localStorage)
getContext	callback	zwraca obiekt z parametrami: • variant: wariant e-podręcznika • isTeacher: czy użytkownik jest nauczycielem
getAudioUrl	id, callback	zwraca link do strumienia WOMI audio, dla WOMI o 'id'
getVideoUrl	id, callback	zwraca link do strumienia WOMI video, dla WOMI o 'id'
setUserAnswer	value	zapisanie womi jako postępu wykonania (statystyki związane z liczba udanych/nieudanych podejść do zadania), wartości: correct, incorrect
saveImageFile	filename, fileData, descriptor, callback	filename to nazwa pliku, fileData to plik w postaci url base64 odczytany np przez FileReader.readAsDataURL, descriptor to deskryptor pliku, który chcemy zaktualizować (null jeśli tworzymy), w callback obiekt odpowiedzi m.in. z deskryptorem
getFileUrl	descriptor, callback	podając deskryptor otrzymuje się url do pliku zapisanego powyższą metodą
sendMail	data	data to obiekt zawierający: subject i body (temat i treść emaila)
getUserInfo	callback	zwraca w callbacku obiekt, pole w authenticated przyjmuje wartość true gdy użytkownik zalogowany, false gdy niezalogowany, pole username przechowyje nazwę użytkownika

Metody do ustawiania zmiennych jak na razie są jedynie makietami i nie zapisują ich w faktycznym kontekście w bazie danych. Warto zaznaczyć, iż można wykorzystywać tą funkcjonalność do testowania bo są zapisywane w zmiennych *javascript*.

Przykład:

```
define(['require', 'reader.api'], function(require, api){
    var readerApi = new api(require);
});
```

API dla WOMI w ramce

ABy używać API w kontekście okna w *iframe*, należy do pliku HTML dodać następujące skrypt:

```
<script src="/global/libraries/epo/frame_script.js"></script>
<script src="/global/libraries/jquery/2.1.0/jquery.min.js"></script>
<script src="/global/libraries/declare/declare.js"></script>
<script src="/global/libraries/epo/api/withoutRequirejs.js"></script>
<script src="/global/libraries/epo/api/ReaderApi.js"></script>
```

Powyższe skrypty pozwalają na uruchomienie API w osobnym pliku HTM w ramce.

API komunikuje się po Message API: https://developer.mozilla.org/en-US/docs/Web/API/Window.postMessage, jednakże jest ono "przykryte" odpowiednio spójnym interfejsem, co pozwala na pełną kompatybilność komunikacji *window<->parent* z przeglądarkami internetowymi wspierającymi HTML5.

W sytuacji kiedy w WOMI korzystamy z *requirejs* wystarczy odpowiednio skonfigurować sobie *jquery*, *declare* i podlinkować ścieżkę do *ReaderApi.js* (jest to moduł *requirejs*).

Zalecane jest załączanie *frame_script.js* niezależnie od tego co wykonywane jest w aplikacji.

metoda	parametry	opis
getFullPath	path	metoda przyjmuje relatywną ścieżkę względem aktualnego modułu (pliku js) z katalogu WOMI, i zwraca pełną ścieżkę (WAŻNE **działa tylko gdy API zostanie załączone przez requirejs)**

metoda	parametry	opis
setUserVar	varName, value	ustawienie zmiennej wykorzystywanej przez aplikację (w kontekście użytkownika, kolekcji, modułu i womi) (zapisuje do bazy danych) #aktualnie to co local
getUserVar	varName, callback	pobranie zmiennej ustawionej powyższą metodą (zapisuje do bazy danych) #aktualnie to co local wartość jest zwracana w funkcji wywołania zwrotnego
setLocalUserVar	varName, value	ustawienie zmiennej wykorzystywanej przez aplikację (w kontekście użytkownika, kolekcji, modułu i womi) (zapisuje do localStarage)
getLocalUserVar	varName, callback	pobranie zmiennej ustawionej powyższą metodą (zapisuje do localStorage), wartość jest zwracana w funkcji wywołania zwrotnego
getAudioUrl	id, callback	zwraca link do strumienia WOMI audio dla WOMI o 'id', wartość jest zwracana w funkcji wywołania zwrotnego
getVideoUrl	id, callback	zwraca link do strumienia WOMI video dla WOMI o 'id', wartość jest zwracana w funkcji wywołania zwrotnego
getPosition	callback	zwraca różne wartości położenia kontenera z WOMI i wielkości okna w podręczniku (wykorzystywane dla układów kaflowych)
setUserAnswer	value	zapisanie womi jako postępu wykonania (statystyki związane z liczba udanych/nieudanych podejść do zadania), wartości: correct, incorrect
saveImageFile	filename, fileData, descriptor, callback	filename to nazwa pliku, fileData to plik w postaci url base64 odczytany np przez FileReader.readAsDataURL, descriptor to deskryptor pliku, który chcemy zaktualizować (null jeśli tworzymy), w callback obiekt odpowiedzi m.in. z deskryptorem
sendMail	data	data to obiekt zawierający: subject i body (temat i treść

metoda	parametry	opis
		emaila)
getFileUrl	descriptor, callback	podając deskryptor otrzymuje się url w callbacku do pliku zapisanego powyższą metodą
getUserInfo	callback	zwraca w callbacku obiekt, pole w authenticated przyjmuje wartość true gdy użytkownik zalogowany, false gdy niezalogowany, pole username przechowyje nazwę użytkownika

Przykład - nie używamy require.js w pliku HTML:

```
$(document).ready(function(){
    var api = new epoReaderApi({});
    api.getUserVar('233049f3094', function(val){
        console.log(val);
    });
    api.setUserVar('233049f3094', 'random value');
});
```

API do pobrania osadzalnego WOMI

Dla WOMI w ramce, można zapytać o URL do osadzenia WOMI w *iframe*. W tym celu należy dodać do skryptów plik:

```
/global/libraries/epo/api/EmbedApi.js
```

a następnie wywołać metodę *getEmbedUrl(type, womiId, womiVersion, callback)*, gdzie type toodpowiednio : *Audio* lub *Video*

Przykład:

```
epoEmbedApi.getEmbedUrl('Audio', 86016, 1, function(data){
    console.log(data);
});
```

W odpowiedzi wywołanej metody dostaniemy obiekt JS z właściwością URL.

Użycie Reader API w WOMI typu "Baw się i ucz" / "Pomyśl i działaj"

W WOMI tego typu można używać analogicznie API dla WOMI w ramce, czyli *ReaderApi* i *EmbedApi*.

Używanie API do zapisu/odczytu danych

API do zapisu danych powinno być wywoływane w kluczowych momentach wykonania aplikacji. Odczyt danych powinien nastąpić na początku uruchamiania aplikacji. WAŻNE: nie zaleca się wywoływania w sposób nieograniczony i/lub niekontrolowany zapisu i odczytu danych/zmiennych podczas wykonywania aplikacji.

Ustawienia WOMI

WOMI powinno zostać opisane dwoma plikami:

- *manifest.json* - plik zawierający definicję silnika dla WOMI
- *metadata.json* - plik zawierający metadane opisujące WOMI

Szablon pliku manifest.json

```
{
    "engine": "custom_womi",
    "mainFile": "womi.js",
    "version": "1.0",
    "parameters": {
        "object": {
            "heightRatio": 0.54
```

```
        }
    },
    "womiIds": [
        "id_womi_1",
        "id_womi_2"
    ]
}
```

Pole *engine* służy do podania silnika przetwarzania dla danego WOMI, dla wcześniej wymienionych przypadków będą to:

- *custom_womi* - dla WOMI jako moduł *requirejs*
- *framed_html* - dla WOMI, które ma być osadzone w ramce jako HTML

Pole *parameters* musi wystąpić, a w nim obiekt zawierający pole *object*, z kolei w nim pole *heightRatio*, które ustawia proporcje WOMI, według których będzie skalowane w widoku prezentacyjnym.

W przypadku zagnieżdżania WOMI w WOMI listę takich obiektów należy zdefiniować poprzez *womiIds.*

Każde WOMI musi mieć podaną proporcję**.** *Height ratio* w tym wypadku to stosunek wysokości do szerokości dla danego WOMI, pozwala to skalować WOMI z zachowaniem jego proporcji.

Przykład dla WOMI w ramce html

```
{
    "engine": "framed_html",
    "mainFile": "index.html",
    "version": "1.0",
    "parameters": {
        "object": {
            "heightRatio": 0.54
        }
    }
}
```

Plik HTML powinien zawierać dołączony skrypt:

```
<script src="/global/libraries/epo/frame_script.js"></script>
```

Przygotowanie WOMI krok po kroku

1. Zalogować się do Edytora Zasobów. Przejść do opcji utworzenia nowego WOMI
2. Stworzyć nowe puste WOMI, czyli strukturę plików, które mogą być już wypełnione kodem.
3. W nowo utworzonym WOMI należy załadować z dysku pliki w sekcji uaktualnienia zasobu.
4. Po udanym załadowaniu możemy edytować pliki WOMI.
5. Po wybraniu "podglądu" możemy na żywo obserwować wprowadzane zmiany.
6. Gdy WOMI jest już kompletne można je pobrać w odpowiednim formacie do importu w Repozytorium Treści na platformie epodreczniki.pl.

Dodatkowe kroki

1. Jeżeli potrzebujemy zaimplementować globalną bibliotekę dla swoich WOMI, należy tworzyć katalogi i pliki w jednym WOMI.
2. Po tym jak biblioteka będzie gotowa, należy postępować zgodnie z opisem przedstawionym w tym dokumencie.
3. Po otrzymaniu informacji zwrotnych od partnera technologicznego można tworzyć WOMI z własnymi bibliotekami.

UWAGA

Wsparcie dla starego sposobu wykorzystywania API (związanego z lokalnym serwerem) nie jest zalecane i nie będzie wspierane.

DODATEK

Istnieje kilka dodatkowych typów WOMI wspierany przez platformę epodreczniki.pl:

Lista dostępnych silników WOMI:

- *edge_animation*: dla animacji ze środowiska *Adobe Edge*
- *createjs_animation*: animacje *CreateJS*

- *ge_animation*: animacje Grupy Edukacyjnej
- *custom_womi*: szablonowe generyczne WOMI, opisywane wyżej
- *custom_logic_exercise_womi*: podobne do *custom_womi*, pozwala tworzyć WOMI, które nie mają rozmiaru, mogą też ładować same z siebie inne WOMI i tworzyć fragmenty treści, w większości API opisano pod adresem: http://fury.man.poznan.pl/~jaftowicz/dokumentacja/EPO.api.PlaceholderApi.html
- *ace_editor*: silnik dla edytora *Ace*
- *svg_editor*: silnik dla edytora *SVG Edit*
- *geogebra*: WOMI typu *geogebra*
- *swiffy*: WOMI typu *swiffy*

custom_logic_exercise_womi

WOMI tego typu nie może być używane bez wcześniejszej konsultacji z PCSS. Partner powinien podać powód czemu chce robić WOMI, w którym może być tekst i inne WOMI i dlaczego tego nie robi tworząc podręcznik.

Użycie *placeholder.api* jest powiązane z *custom_logic_exercise_womi* jest analogiczne do *reader.api*:

```
define(['require', 'jquery', 'declare', 'placeholder.api'], function (require, $, declare, papi) {
    return declare({
        instance: {
            start: function (placeholder) {
                var pa = new papi($(placeholder), require);
                //... more code here
            }
        }
    });
});
```