ipfon

ipfon IVR API

wersja 1.0.3

Zawartość

1. Historia zmian

L.p.	Wersja	Data	Opis
1	1.0.0	10 października 2010	wersja początkowa
2	1.0.1	05 listopada 2010	dodanie *SayTime*, poprawki edytorskie
3	1.0.2	08 listopada 2010	zmiana nazwy parametru z *Play* na *Prompt* w *GetDTMF* i *Play*, dodanie tablicy błędów, poprawki edytorskie
4	1.0.3	24 sierpnia 2011	dodanie wartości parametru *ChannelStatus*

2. Ogólny zarys działania usługi

IVR API to usługa systemu pozwalająca na wykonanie zdalnych funkcji na wirtualnej centrali za pomocą zewnętrznych aplikacji. IVR API pozwala na sterowanie obsługą połączenia przez aplikację zewnętrzną (aplikację klienta).

Po połączeniu na IVR, w zależności od ustawienia opcji „automatyczne odbieranie", połączenie jest albo odbierane po ustalonym w sekundach czasie, albo decyzja o tym, czy to połączenie ma zostać odebrane, jest przekazywana do skryptu klienckiego.

2.1. Przykładowe zastosowania

Usługa IVR API może mieć bardzo wiele zastosowań, Jej największą zaletą jest jej elastyczność i nieograniczone możliwości jakie pozostawia ona twórcy aplikacji. Przykładowo, usługa IVR API może służyć jako:

1) system obsługi konkursów - producent umieszcza kod wewnątrz opakowania swojego produktu, klient powinien przedzwonić pod podany numer i podać ten kod i uzyskać informację o wygranej,
2) telefoniczna informacja o statusie zamówienia, gdzie klient podaje przez DTMF numer zamówienia, a aplikacja zwraca jego status i odgrywa określoną zapowiedź: np. zamówienia zostało wysłane, numer listu przewozowego to 1234567,
3) weryfikacja na podstawie ID klienta i PINu osoby dzwoniącej,
4) weryfikacja po numerze osoby dzwoniącej i kierowanie połączenia do określonej osoby/działu,
5) zegarynka – przykładowa zegarynka dostępna jest pod numerem 61 622 26 12,
6) odgrywanie wybranego powitania w zależności od numeru osoby dzwoniącej,
7) potwierdzanie zamówień / numerów telefonów – klient musi zadzwonić na podany numer, a tam dostanie zadanie – podaj następujący kod, np. 1579 i aplikacja go zweryfikuje – odpowiednik systemu Captcha.

2.2. Zamawianie i uruchamianie IVR API

Zamówienia usługi IVR API dokonujemy poprzez rejestrator usług dostępny po zalogowaniu się do *ipfon24* w dziale *Zamawianie usług* wybierając link nr 3.

Po pomyślnym zakończeniu zamówienia i jego opłaceniu można przystąpić do uruchomienia. Założenia IVR API dokonujemy w ipfon24 na odpowiednim koncie billingowym, skojarzonym z zamówieniem.

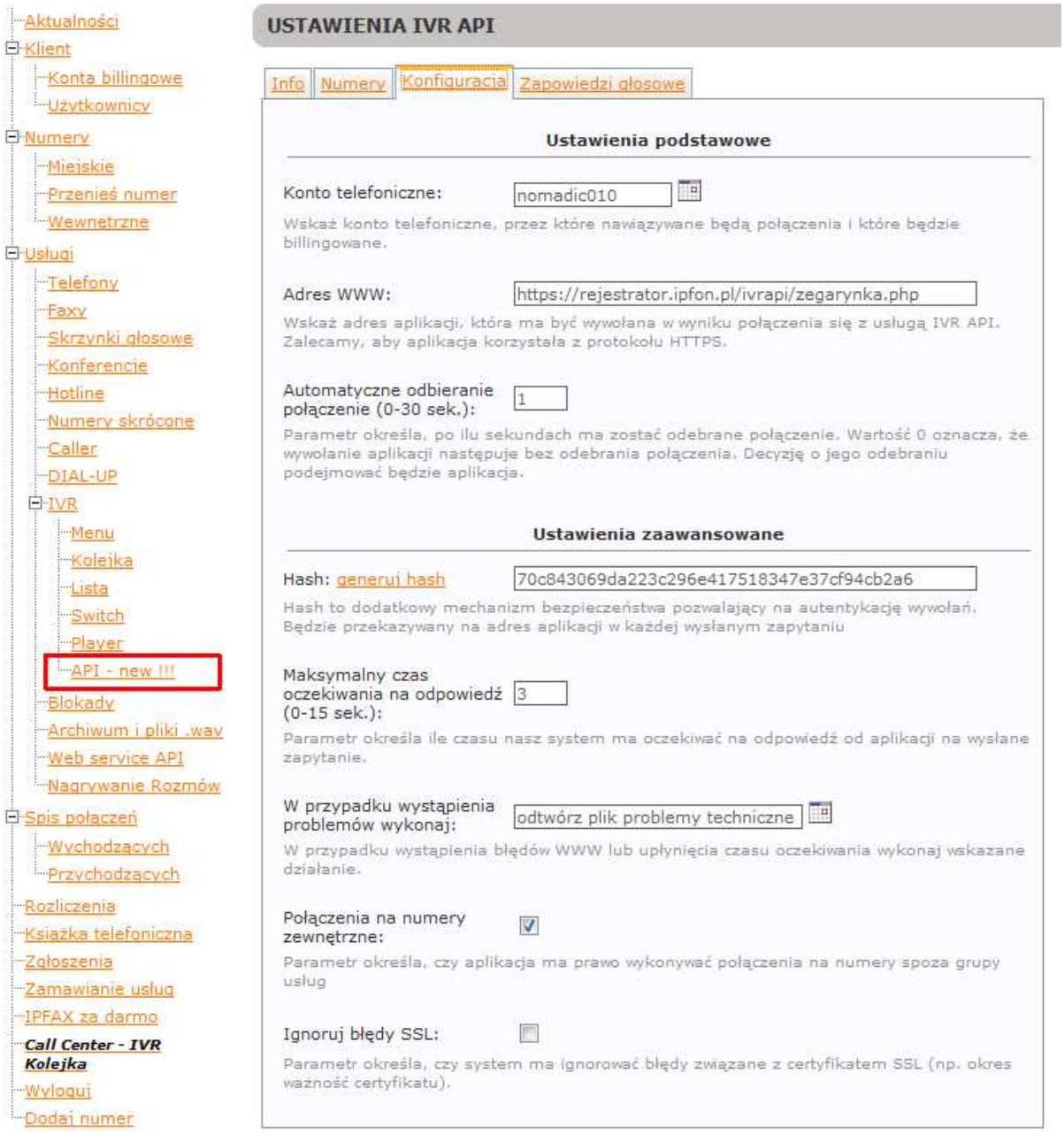
1) Aby uruchomić IVR API należy z *Menu* -> IVR w *ipfon24* wybrać opcje API . Następnie należy w zakładce ***Info*** nadać nazwę oraz wybrać odpowiednie konto billingowe, na który przypisane jest zamówienie.
2) Następnie w zakładce ***Numery*** przypisujemy numer miejski do usługi (opcja).
3) Kolejnym krokiem jest zakładka ***Konfiguracja***, w której definiujemy parametry usługi.
4) W zakładce ***Zapowiedzi głosowe*** przypisujemy poszczególnym pozycją pliki WAV z zapowiedziami.

Konfiguracja - ustawienia podstawowe

Konto telefoniczne – konto SIP, przez które nawiązywane będą połączenia na numery wskazane w ustawieniach IVR API. Ustawienie tego pola jest obowiązkowe jeśli mają być zestawiane połączenia z IVR na zewnętrzne numery telefoniczne.

Adres WWW - adres aplikacji, która ma być wywoływana w wyniku połączenia z usługą IVR API. Zalecamy aby aplikacja korzystała z protokołu HTTPS.

Automatycznie odbierać połączenie - parametr określa po ilu sekundach ma zostać odebrane połączenia. Wartość "0" (zero) oznacza, że decyzja o m, czy odebrać połączenia będzie pozostawiona aplikacja klienta.



Konfiguracja - ustawienia zaawansowane

Hash - dodatkowy mechanizm bezpieczeństwa pozwalający na autentykację wywołań. Będzie przekazywany na adres aplikacji w każdym wysłanym zapytaniu.

Maksymalny czas oczekiwania na odpowiedź - parametr określający ile czasu nasz system ma oczekiwać na odpowiedź od aplikacji na wysłane zapytanie. W przypadku wystąpienia błędów WWW lub upłynięcia czasu oczekiwania system może wykonać zdefiniowane działanie.

W przypadku problemów wywołaj połączenie z: - w przypadku wystąpienia błędów WWW, upłynięcia czasu oczekiwania definiujemy działanie jakie zostanie wykonane.

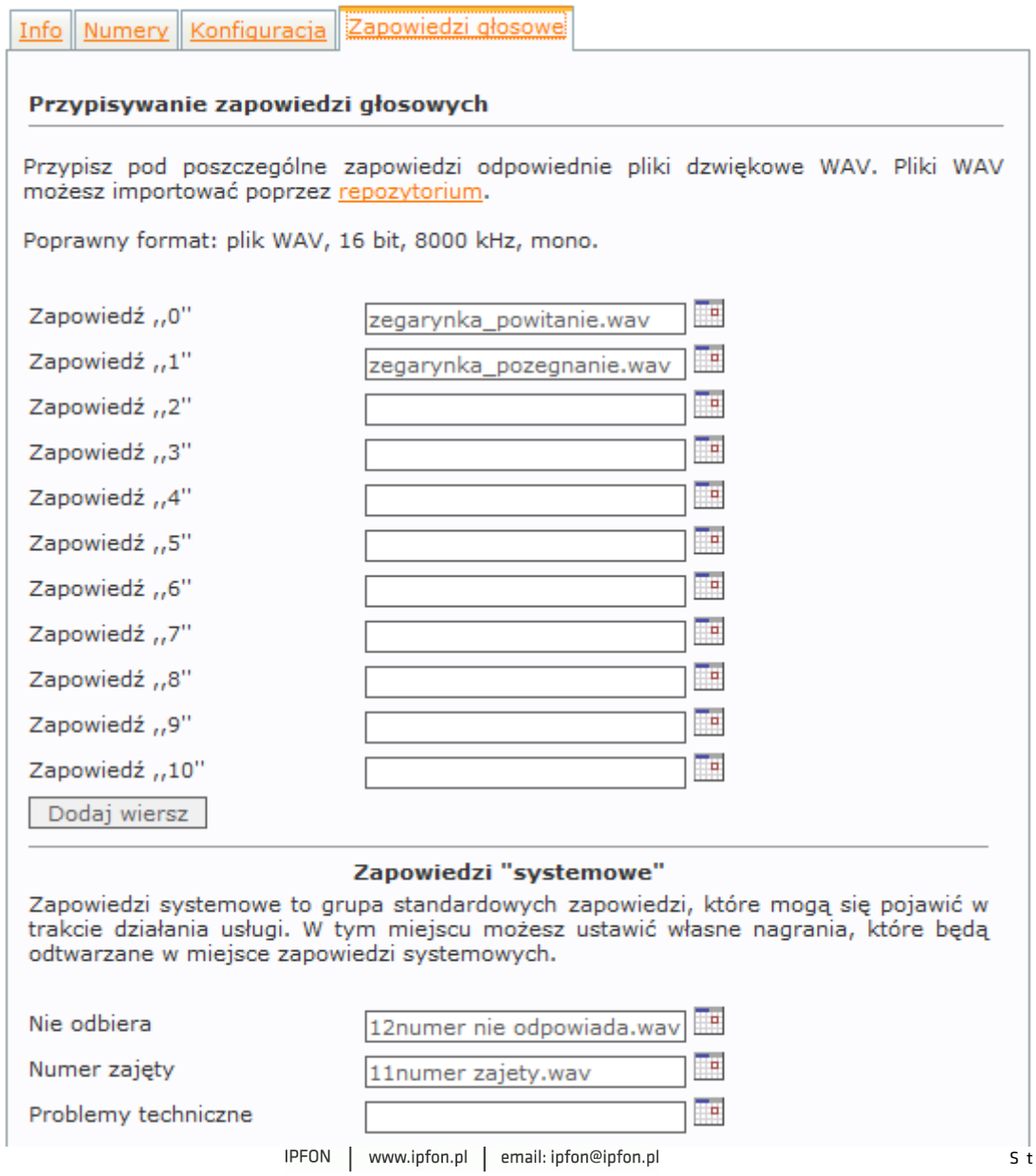
Połączenia na numery zewnętrzne - parametr określa, czy aplikacja ma prawo wykonywać połączenia na numery spoza grupy usług.

Ignoruj błędy ssl - parametr określa czy system ma ignorować błędy związane z certyfikatem SSL (np. okres ważności certyfikatu).

Zapowiedzi głosowe

Zapowiedzi od 1 do 10 - w tych polach pod poszczególne numery zapowiedzi podpinamy zaimportowane wcześniej pliki WAV.

Zapowiedzi systemowe - pozwalają na ustawienia własnych głosów dla zapowiedzi systemowych, które odgrywane są automatycznie przez system w określonej sytuacji: gdy nie odbiera , gdy numer zajęty lub w przypadku wystąpienia problemów technicznych.



3. Algorytm działania usługi

1) Połączenie trafia do IVR API albo poprzez połączenie na bezpośredni numer przypięty do usługi albo poprzez inną usługę IVR.
2) Po odczekaniu zadaną ilość sekund, ustawianą w panelu klienta połączenie jest odbierane (w przypadku wartości 0 ten krok jest pomijany).
3) Wywołanie skryptu pod wskazanym adresem, który dalej będzie sterował połączeniem.
4) Jeśli skrypt kliencki chce się rozłączyć, to skocz do punktu 7.
5) Wykonaj akcję podaną przez skrypt kliencki.
6) Skocz do punktu 3
7) Rozłącz się.

Komunikacja ze skryptem klienckim wygląda w następujący sposób. Dane są przesyłane przez metodę POST protokołu HTTP. Parametry opisujące zapytanie znajdują się w polu „request" i są one opakowane do formatu JSON.

Parametry są tablicą asocjacyjną, której klucze mogą przyjmować następujące wartości:
FromNumber, ToNumber, Session, Event, ChannelStatus, UniqueCallId, oraz **Error** (który jest obsługiwany w specjalny sposób)

Poszczególne pola zostaną opisane dokładniej w dalszej części tego dokumentu.

Po wysłaniu powyższego zapytania, program IVR API oczekuje, aby skrypt kliencki przekazał akcje do wykonania, w treści strony. Akcje te powinny zostać opakowane w formacie JSON.

Odpowiedź powinna być tablicą asocjacyjną, która ma następujące pola: **Action, Session**

W przypadku wystąpienia błędu, w zapytaniu może pojawić się pole opisujące błąd – **Error**.

3.1. Przykładowe zapytania w IVR API

Mamy ustawione automatyczne odbieranie po 1 sekundzie. Wdzwania się osoba na IVR API, połączenie jest odbierane i IVR łączy się do strony klienckiej, wysyła żądanie, które po zdekodowaniu przez JSON wygląda mniej więcej tak:

```
array (
   'FromNumber' =>  '123456789',
   'ToNumber' =>  '123456780',
   'ChannelStatus' =>  'Connected',
   'UniqueCallId' =>  'f95bccb5b1140554030b9d4186855cdc12e6e1c0',
   'Hash' =>  '4fa0ff6d065ba50c13208224cc19b23f32e6e1c0',
   'Event' =>
        array(
                 'EventName' =>  'Connected',
   ),
   'Session' =>  NULL,
)
```

Zdarzenie to informuje o tym, ze połączenie zostało odebrane; podaje numer źródłowy, numer docelowy itp.

Przykładowa odpowiedź skryptu klienckiego, po zdekodowaniu JSON, może wyglądać tak:

```
array (
   'Action' =>
        array (
                  'Type' =>  'CallExternalNumber',
                  'Destination' =>  '125555555',
                  'Options' =>  'PlayCallStatus,Timeout=20',
   ),
   'Session' =>
        array (
   ),
)
```

Po otrzymaniu takiej odpowiedzi IVR zainicjuje nowe połączenie na numer 125555555. Po zakończeniu wybranego połączenia IVR API rozłączy połączenie i wyśle takie żądanie do skryptu klienckiego:

```
array (
   'FromNumber' =>  '123456789',
   'ToNumber' =>  '123456780',
   'ChannelStatus' =>  'Hangup',
   'UniqueCallId' =>  'f95bccb5b1140554030b9d4186855cdc12e6e1c0',
   'Hash' =>  '4fa0ff6d065ba50c13208224cc19b23f32e6e1c0',
   'Event' =>
        array(
                  'EventName' =>  'Hangup',
                  'EventData' =>  '',
   )),
   'Session' =>
        array (
   ),
)
```

Na powyższe zapytanie skrypt kliencki nie odpowiada.

4. Opis parametrów przesyłanych przy wywołaniu skryptu klienckiego

FromNumber - numer źródłowy,

ToNumber - numer docelowy,

Session – parametr, który został przekazany podczas ostatniej odpowiedzi skryptu klienckiego. Pole to można użyć do przechowywania wartości parametrów pomiędzy wywołaniami; może to być tekst, liczba, tablica asocjacyjna itp.

Event - tablica asocjacyjna opisująca zdarzenie, które zaszło.

ChannelStatus – status połączenia. Parametr może przyjmować wartości: *Ringing*, *Connected* lub *Hangup*.

UniqueCallId – unikalny identyfikator połączenia,

Hash – unikatowy skrót generowany w panelu klienta.

Tablica ***Event*** posiada 2 parametry. Parametr *EventName* określa rodzaj zdarzenia, jaki zaszedł. W parametrze *EventData* znajdują się dodatkowe informacje o zdarzeniu, które zaszło.

Możliwe wartości parametru *EventName*:

GetDTMF – zwraca w elemencie *EventData* klawisze jakie zostały wciśnięte przez użytkownika, bądź wartość ,,timeout'', jeśli użytkownik nie wcisnął żadnych klawiszy w określonym czasie,

Play – informuje o tym, że wybrany utwór z repozytorium został odegrany,

Connected – informuje o nawiązaniu nowego połączenia,

Answered – połączenie zostało odebrane,

CallStatus – zwraca kod wyjścia ostatniej próby połączenia w elemencie *EventData*. Możliwe stany:

Answer – połączenie zostało odebrane,

Busy – wybrany numer jest zajęty,

Noanswer – wybrany numer nie odebrał,

Cancel – osoba dzwoniąca rozłączyła się.

Error – wystąpił błąd.

W przypadku wystąpienia błędu wykonywana jest określona akcja – rozłączenie / odegranie komunikatu / połączenie z obiektem. Następnie program łączy się do skryptu klienckiego i podaje w tablicy asocjacyjnej o kluczu *Error* dwa parametry:

- **ErrorNo** – unikatowy identyfikator błędu, który wystąpił,
- **ErrorMsg** – komunikat tekstowy opisujący błąd, zawierający dodatkowo wszystkie kroki wywołania skryptu.

Po wysłaniu takiego komunikatu skrypt nie oczekuje na otrzymanie jakiejkolwiek odpowiedzi

Kod błędu	Opis
1000	Zbyt wiele odpytań do aplikacji klienta
1001	Aplikacja kliencka odesłała zbyt dużo danych
1002	Aplikacja kliencka nie odpowiedziała w wyznaczonym czasie
1003	Problem z certyfikatem SSL
1004	Błąd podczas pobierania adresu IP dla domeny adresu aplikacji klienta
1005	Błąd podczas łączenia z aplikacji klienta
1006	Niepoprawna odpowiedź aplikacji klienta
2001	Plik do odegrania nie istnieje
2002	Zły format znacznika czasu
2003	Złe parametry dla wybranej akcji
3001	Brak skonfigurowanego konta SIP dla połączeń wychodzących
9999	Nieznany błąd

5. Opis parametrów zwracanych przez skrypt kliencki

Action – tablica asocjacyjna zawierająca akcję do wykonania. Powinna ona zawierać klucz o nazwie *Type*, którego wartość określa typ akcji do wykonania. Lista możliwych do użycia wartości tego pola to: *GetDTMF*, *Wait*,*Play*, *Answer*,*DontAnswer*, *Hangup CallExternalNumber*, *CallLocalNumber*, *CallByObjectLogin*, *SayNumber*, *SayDigits*, *SayTime*.

Poniżej znajduje się lista możliwych do wykonania akcji, oraz parametry, jakie mogą zostać dla nich przekazane.

5.1. GetDTMF

GetDTMF – pobiera wciskane na klawiaturze telefonu klawisze. Opcjonalnie można wybrać jedną z dostępnych zapowiedzi głosowych. Wczytywanie klawiszy kończy się przed czasem, jeśli wciśniemy przycisk '#'.

parametr	opis
Prompt	określa numer zapowiedź, która ma być odgrywana. Zapowiedź, ustawiana jest w panelu klienta.
Timeout	ile milisekund czekać na naciśnięcie kolejnego klawisza
MaxDigits	maksymalna ilość odczytywanych klawiszy

Przykład:

```
$response['Action'] = array('Type' => 'GetDTMF', 'Prompt' => '1', 'Timeout' => '1000', 'MaxDigits' => '10')
```

5.2. Wait

Wait – oczekuje zadaną ilość sekund

parametr	opis
WaitTime	Ilość sekund w przedziale 0-60

Przykład:

```
$response['Action'] = array('Type' => 'Wait', 'WaitTime' => '20');
```

5.3. Play

Play – odgrywa zadany plik dźwiękowy

parametr	opis
Prompt	Określa numer zapowiedzi, która ma zostać odegrana. Nagrania ustawia się w Panelu Klienta

Przykład:

```
$response['Action'] = array('Type' => 'Play', 'Prompt' => '1');
```

5.4. Answer

Answer – odbiera połączenie

Przykład:

```
$response['Action'] = array('Type' => 'Answer');
```

5.5. DontAnswer

DontAnswer – czeka aż osoba dzwoniąca rozłączy się, bądź minie 45 sekund, kiedy to numer dzwoniący usłyszy sygnał zajętości

Przykład:

```
$response['Action'] = array('Type' => 'DontAnswer');
```

5.6. Hangup

Hangup – rozłącz się

Przykład:

```
$response['Action'] = array('Type' => 'Hangup');
```

5.7. CallExternalNumber

CallExternalNumber – dzwoni na wybrany dowolny numer zewnętrzny. Połączenie zestawione będzie poprzez wskazane w panelu klienta domyślne konto SIP.

parametr	opis
Destination	Numer, na który ma zostać wykonane połączenie. Uwaga: opcja „pozwól na połączenia na numery zewnętrzne" musi być włączona w panelu, aby ta akcja zadziałała
Options	Opcje oddzielone przecinkami. Dostępne opcje to: **ContinueAfterCompletition** – po zakończeniu połączenia kontynuuj odpytywanie skryptu o dalsze akcje. **PlayCallStatus** - odegraj komunikat informujący osobę podłączoną do IVR o statusie połączenia, np. ,,wybrany numer jest zajęty'', ,,wybrany numer nie odpowiada''. **RingTones** – generuj sygnał wybierania numeru, **Timeout** – wartość od 1 – 30; ile sekund czekać na połączenie.

Przykład:

```
$response['Action'] = array('Type' => 'CallExternalNumber', 'Destination' => '501234567', 'Options' => 'PlayCallStatus,Timeout=30');
```

5.8. CallLocalNumber

CallLocalNumber – dzwoni na wybrany numer ale ograniczony do numerów z grupy numerów klienta.

parametr	opis
Destination	Numer, na który ma zostać wykonane połączenie
Options	Tak jak w przypadku akcji *CallExternalNumber*

Przykład:

```
$response['Action'] = array('Type' => 'CallLocalNumber', 'Destination' => '616666666", 'Options' => 'PlayCallStatus,Timeout=30');
```

5.9. CallByObjectLogin

CallByObjectLogin – dzwoni na wybrany obiekt, wykorzystują login obiektu w ramach grupy klientów.

parametr	opis
Destination	Nazwa obiektu na który ma zostać wykonane połączenie
Options	Tak jak w przypadku akcji *CallExternalNumber*

Przykład:

```
$response['Action'] = array('Type' => 'CallByObjectLogin', 'Destination' => 'test001', 'Options' => 'PlayCallStatus,Timeout=30');
```

5.10. SayNumber

SayNumber – syntezator wymawia podana liczbę (np. 55 jako pięćdziesiąt pięć).

parametr	opis
Number	Numer, który ma zostać wypowiedziany

Przykład:

```
$response['Action'] = array('Type' => 'SayNumber', 'Number' => '555');
```

5.11. SayDigits

SayDigits – syntezator wymawia podane cyfry (np. 55 jako pięć pięć).

parametr	opis
Digits	Cyfry, które mają zostać wypowiedziane

Przykład:

```
$response['Action'] = array('Type' => 'SayDigits', 'Digits' => '555');
```

5.12. SayDate

SayDate – syntezator wymawia datę i/lub godzinę (np. piątek, 5 listopada 2010 rok, 13:14:24).

parametr	opis
Format	format określający, w jaki sposób przeczytać żądaną datę. Poszczególne litery określają dany człon daty, jak np. dzień tygodnia, rok, godzina itp. Ten parametr musi przyjmować jedną, lub więcej liter, które są interpretowane następująco: A - dzień tygodnia, jak np. ,,wtorek'', B - nazwa miesiąca, jak np. ,,listopad'', m - numer miesiąca, jak np. ,,jedenasty'', e - dzień miesiąca, jak np. ,,drugi'' Y - rok, jak np. ,,dwa tysiące dziesiąty'', H - godzina, jak np. ,,piętnasta'', M - minuta, jak np. ,,trzy'', S - sekunda, jak np. ,,i 5 sekund''
Timestamp	Czas POSIX http://pl.wikipedia.org/wiki/Czas_uniksowy),

Przykład:

```
$response['Action'] = array('Type' => 'SayDate', 'Format' => 'AeBYHMS', 'Timestamp'
=> '1288962864');

Odczyta datę: piątek, 5 listopada 2010 rok, 13:14:24
```

6. Przykłady

6.1. Zegarynka

Pod numerem 61 622 26 12 dostępna jest zegarynka, która podaje aktualny dzień tygodni, godziny i minuty.

```
<?
ini_set('log_errors', 'error_log');
ini_set('error_log', '/tmp/php-error.log');
ini_set('log_errors', 'On');
ini_set('display_errors', 'Off');
ini_set('error_reporting', E_ALL);


// stany aplikacji
define('STATE_CONNECTED', 1);
define('STATE_PLAYED_POWITANIE', 2);
define('STATE_PLAYED_TIME', 3);

// zapowiedzi
define('POWITANIE', 0);
define('POZEGNANIE', 1);

// adres, na ktory beda wysylane powiadomienia
define('ADMINISTRATOR_EMAIL', 'admin@admin.pl');

// sciezka do pliku logu
define('LOG_FILENAME', '/tmp/app-log.txt');

function _log($text)
{
        $mail_text = date('Y-m-d H:i:s ') . $text . "\n";
        $f = fopen(LOG_FILENAME, "a+");
        fwrite($f, $mail_text);
        fclose($f);
}

_log("Polaczony");

if (get_magic_quotes_gpc())
        $_POST["request"] = stripslashes($_POST["request"]);

_log("Zawartosc zapytania: " . $_POST["request"]);

$request = (array) json_decode($_POST["request"]);

$session = (array) $request["Session"];

_log(var_export($request, TRUE));

// w przypadku, gdy otrzymujemy komunikat o bledzie
// wysylamy na e-mail powiadomienie i konczymy prace
if (array_key_exists("Error", $request)) {
        mail(ADMINISTRATOR_EMAIL, "Blad aplikacji", var_export($request, TRUE));
}
```

```
_log("Zawartosc sesji na poczatku: " . var_export($session, TRUE));

$response = array();

if (!$session["state"])
    $session["state"] = STATE_CONNECTED;

_log("Stan: " . $session["state"]);

$event = (array) $request["Event"];

_log("Event: " . var_export($event, TRUE));

switch ($session["state"])
{
    case STATE_CONNECTED:
        $response["Action"] = array("Type" => "Play", "Prompt" => POWITANIE);
        $session["state"] = STATE_PLAYED_POWITANIE;
        break;

    case STATE_PLAYED_POWITANIE:
        $response["Action"]   =   array('Type'   =>   'SayDate',   'Format'   =>   'AHM',
'Timestamp' => time());
        $session["state"] = STATE_PLAYED_TIME;
        break;

    case STATE_PLAYED_TIME:
        $response["Action"] = array("Type" => "Play", "Prompt" => POZEGNANIE);
        $session["state"] = STATE_GOODBYE;
        break;

    default:
        $response["Action"] = array('Type' => 'Hangup');
        _log("Nieznany stan?: " . $session["state"]);
        break;
}

_log("session: " . var_export($session, TRUE));

$response["Session"] = $session;

_log("response: " . var_export($response, TRUE));

echo json_encode($response);

?>
```

6.2. Weryfikacja numeru A (osoby dzwoniącej)

W tym przykładzie pokazujemy skrypt, który przekierowuje połączenie na numer komórkowy 501234567 tylko, gdy połączenie przychodzi z numeru 6161234567. W pozostałych przypadkach połączenie nie jest odbierane albo jest automatycznie rozłączane.

```
<?

// włączamy logowanie błędów do pliku; należy odpowiednio samemu ustawić poprawną ścieżkę
ini_set('log_errors', 'error_log');
ini_set('error_log', '/tmp/plik-z-logami-php');
ini_set('log_errors', 'On');
ini_set('display_errors', 'Off');
ini_set('error_reporting', E_ALL);


// adres e-mail, na który będziemy wysyłać komunikaty błędów
define('ADMINISTRATOR_EMAIL', 'adres@email');


function _error($text) {
        header('HTTP/1.0 500 Blad\n');

        echo $text;
}

// jeśli PHP ma włączone magic_quotes, to musimy wyłuskać oryginalną wartość parametru $_POST["request"]
if (get_magic_quotes_gpc())
        $_POST["request"] = stripslashes($_POST["request"]);

$request = (array) json_decode($_POST["request"]);


// jeśli nie otrzymaliśmy poprawnego zapytania - kończymy działanie
if (!$request) {
        die("Blad - request jest pusty");
}

if ($request["Error"]) {
    mail(ADMINISTRATOR_EMAIL, '', var_export($request, TRUE));
    exit(0);
}

$response = array();

if (!array_key_exists("Event", $request)) {
        _error("Blad - brak zdarzenia" . var_export($request, TRUE));
}


if (preg_match('/^(0|\+48|)6161234567$/', $request["FromNumber"])) {
    $response["Action"] = array("Type" => "CallExternalNumber", "Destination" => "501234567", "Options" => "PlayCallStatus,Timeout=10");

} else {
```

```
        if ($request["ChannelStatus"] == "Ringing")
                $response["Action"] = array("Type" => "DontAnswer");

        else
                $response["Action"] = array("Type" => "Hangup");
}

// przechowywanie danych sesji pomiędzy wywołaniami
$response["Session"] = (array) $request["Session"];

echo json_encode($response);


?>
```

6.3. Autoryzacja osoby dzwoniącej na podstawie ID i PIN

Aplikacja w tym przykładzie pozwala na połączenie z dalszym obiektem IVR, jeśli użytkownik poprawnie poda kod swój unikatowy identyfikator i kod PIN.

```
<?


ini_set('log_errors', 'error_log');
ini_set('error_log', '/tmp/php-error.log');
ini_set('log_errors', 'On');
ini_set('display_errors', 'Off');
ini_set('error_reporting', E_ALL);


// stany aplikacji
define('STATE_CONNECTED', 1);
define('STATE_ENTER_AGENT_ID', 2);
define('STATE_ENTERED_AGENT_ID', 3);
define('STATE_ENTER_PIN', 4);
define('STATE_ENTERED_PIN', 5);
define('STATE_END', 6);

// zapowiedzi
define('PROMPT_ENTER_AGENT_ID', 0);
define('PROMPT_ENTER_AGENT_PIN', 1);
define('PROMPT_OK', 2);

// adres, na ktory beda wysylane powiadomienia
define('ADMINISTRATOR_EMAIL', 'adres@email');

// sciezka do pliku logu
define('LOG_FILENAME', '/tmp/app-log.txt');

function _log($text) {

    $mail_text = date('Y-m-d H:i:s ') . $text . "\n";
    $f = fopen(LOG_FILENAME, "a+");
    fwrite($f, $mail_text);
```

```
    fclose($f);
}

// ID agenta => PIN
$agents = array("1234" => "1234",
                "7777" => "1111");


_log("Polaczony");.

if (get_magic_quotes_gpc()).
    $_POST["request"] = stripslashes($_POST["request"]);


_log("Zawartosc zapytania: " . $_POST["request"]);

$request = (array) json_decode($_POST["request"]);

$session = (array) $request["Session"];

_log(var_export($request, TRUE));

//  w  przypadku,  gdy  otrzymujemy  komunikat  o  bledzie  -  wysylamy  na  e-mail
powiadomienie i konczymy prace
if (array_key_exists("Error", $request)) {
        mail(ADMINISTRATOR_EMAIL, "Blad aplikacji", var_export($request, TRUE));
}

_log("Zawartosc sesji na poczatku: " . var_export($session, TRUE));

$response = array();

if (!$session["state"])
    $session["state"] = STATE_CONNECTED;

_log("Stan: " . $session["state"]);


$event = (array) $request["Event"];

_log("Event: " . var_export($event, TRUE));

switch ($session["state"]) {

    case STATE_CONNECTED:

        $session["state"] = STATE_ENTER_AGENT_ID;
        $response["Action"] = array("Type" => "Wait", "WaitTime" => "0");

        break;

    case STATE_ENTER_AGENT_ID:
```

```
        $response["Action"]    =    array("Type"    =>    "GetDTMF",    "Prompt"    =>
PROMPT_ENTER_AGENT_ID);
        $session["state"] = STATE_ENTERED_AGENT_ID;

        break;

    case STATE_ENTERED_AGENT_ID:

        if ($event["EventName"] == "GetDTMF") {
            if ($event["EventData"] <> "timeout") {

                $session["state"] = STATE_ENTER_PIN;
                $session["agent_id"] = $event["EventData"];

            } else.
                $session["state"] = STATE_ENTER_AGENT_ID;

            $response["Action"] = array("Type" => "Wait", "WaitTime" => "0");
.............
        }

        break;

    case STATE_ENTER_PIN:

        $response["Action"]    =    array("Type"    =>    "GetDTMF",    "Prompt"    =>
PROMPT_ENTER_AGENT_PIN);
        $session["state"] = STATE_ENTERED_PIN;
        break;

    case STATE_ENTERED_PIN:

        if ($event["EventName"] == "GetDTMF") {

            $response["Action"] = array("Type" => "Wait", "WaitTime" => "0");

            if ($event["EventData"] <> "timeout") {

                $session["agent_pin"] = $event["EventData"];
                $session["state"] = STATE_END;

            } else.
                $session["state"] = STATE_ENTER_PIN;

.............
        }
        break;

    case STATE_END:

        $agent_id = $session["agent_id"];
        $agent_pin = $session["agent_pin"];

        _log("Koniec $agent_id, $agent_pin");
```

```
        $response["Action"] = array("Type" => "Wait", "WaitTime" => "0");

        if (array_key_exists($agent_id, $agents) && $agents[$agent_id] == $agent_pin)
            $response["Action"] = array("Type" => "GetDTMF", "Prompt" => PROMPT_OK);
        else.
            $session["state"] = STATE_ENTER_AGENT_ID;
        break;

    default:
        _log("Nieznany stan?: " . $session["state"]);
        break;

}


_log("session: " . var_export($session, TRUE));

$response["Session"] = $session;

_log("response: " . var_export($response, TRUE));

echo json_encode($response);


?>
```