

# API игрового Web сервера

---

## Оглавление

Введение .....	3
Контактная информация .....	3
{URL_API}/gameSnapshot/{FID}.json .....	4
{URL_API}/listInvitingGames/{FID}.json .....	5
{URL_API}/listAvailableGames/{FID}.json .....	6
{URL_API}/createGame/{FID}.json .....	7
{URL_API}/removeGame/{FID}.json .....	8
{URL_API}/invitePlayer/{FID}.json .....	9
{URL_API}/removePlayer/{FID}.json .....	10
{URL_API}/startGame/{FID}.json .....	11
{URL_API}/distributionCards/{FID}.json .....	12
{URL_API}/leftGame/{FID}.json .....	13
{URL_API}/requestCard/{FID}.json .....	14
{URL_API}/gameAttack/{FID}.json .....	15
{URL_API}/gameDefend/{FID}.json .....	16
{URL_API}/callShiftUp/{FID}.json .....	17
{URL_API}/joinGame/{FID}.json .....	18
{URL_API}/defendNPC/{FID}.json .....	19
{URL_API}/attackNPC/{FID}.json .....	20
{URL_API}/ratingList/{FID}.json .....	21
{URL_API}/resetTrigger/{FID}.json .....	22
{URL_API}/signUp/{FID}.json .....	23
{URL_API}/logout/{FID}.json .....	24
{URL_API}/checkAuth/{FID}.json .....	25
{URL_API}/register/{FID}.json .....	26
{URL_API}/saveProfile/{FID}.json .....	27
{URL_API}/deleteProfile/{FID}.json .....	28
{URL_API}/costGames/{FID}.json .....	29
{URL_API}/listSavingGames/{FID}.json .....	30

{URL_API}/restoreGame/{FID}.json.....	31
{URL_API}/sendMessage/{FID}.json.....	32
{URL_API}/chatMessages/{FID}.json.....	33
Алгоритм 1. Формирования игрового снимка. ....	34
Алгоритм 2. Формирование снимка сущности игрока.....	36
Алгоритм 3. Формирование хеша игровой карты. ....	37
Алгоритм 4. Формирование снимка модели авторизации. ....	38
Алгоритм 5. Формирование снимка сообщения. ....	39
Приложение 1. Типы игры. ....	40
Приложение 2. Игровые статусы игровой модели.....	41
Приложение 3. Игровые действия. ....	42
Приложение 4. Типы игроков.....	43
Приложение 5. Действия игрока.....	44
Приложение 6. Игровые статусы игрока. ....	45
Приложение 7. Игральные карты. ....	46
Масти. ....	46
Значения карт. ....	46

## Введение.

В данном руководстве описан API игрового Web-сервера, который использует приложение «Карточная игра «Дурак» для связи с игровым сервером и управления игровым процессом.

Примеры алгоритмов будут даны с использованием псевдокода и синтаксиса PHP.

## Контактная информация.

По всем вопросам, связанным с API игрового Web-сервера можно обращаться к автору программы.

Автор программы «Карточная игра «Дурак» — Чуркин Вадим.

Адрес электронной почты: [vadim\\_churkin@mail.ru](mailto:vadim_churkin@mail.ru).

ICQ: #230-114-238

Домашняя страница: <http://wisepoint.ru>.

**{URL\_API}/gameSnapshot/{FID}.json**

Возвращает снимок заданной игры.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>userID</b> = CLIENT_ID	идентификатор пользователя

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

{

<b>msgID</b> = MSG_GAME_SNAPSHOT	(10004)
----------------------------------	---------

**frameID** = FRAME\_ID

При успехе:

<b>result</b> = 'snapshot'	игровой снимок (см. алгоритм 1)
----------------------------	---------------------------------

**errorMessage** = пустая строка

<b>errorID</b> = EVENT_NONE	(0)
-----------------------------	-----

При неудаче:

**result** = null

<b>errorMessage</b> = 'LANG_ERROR_GAME_SNAPSHOT'	сообщение об ошибке
--	---------------------

<b>errorID</b> = EVENT_SERVER_ERROR	(64)	код ошибки
-------------------------------------	------	------------

}

**{URL\_API}/listInvitingGames/{FID}.json**

Возвращает список игр, созданных заданным игроком, которые производят набор команды.

Аргументы:

{POST}

**hid** = CLIENT\_HID

HID пользователя

**ownerID** = CLIENT\_ID

идентификатор пользователя

[GET]

**{FID}** = FRAME\_ID

идентификатор фрейма

Возвращает в форматеJSON:

{

**msgID** = MSG\_LIST\_OWNER\_INVITING\_GAMES (10017)

**frameID** = FRAME\_ID

При удаче:

**result** = 'snapshot1|+|snapshot2|+|...snapshotN', где *snapshot* – это игровой снимок (см. алгоритм 1)

**errorMessage** = пустая строка

**errorID** = EVENT\_NONE (0)

При неудаче:

**result** = null

**errorMessage** = 'LANG\_ERROR\_LIST\_INVITING\_GAMES ' сообщение об ошибке

**errorID** = EVENT\_SERVER\_ERROR (64) код ошибки

}

**{URL\_API}/listAvailableGames/{FID}.json**

Возвращает список игр, в которых может принять участие текущий пользователь.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>userID</b> = CLIENT_ID	идентификатор пользователя

[GET]

**{FID}** = FRAME\_ID                      идентификатор фрейма

Возвращает в формате JSON:

```
{
    msgID = MSG_LIST_AVAILABLE_GAMES          (10015)
    frameID = FRAME_ID
```

При успехе:

**result** = 'snapshot1|+|snapshot2|+|...snapshotN', где *snapshot* – это игровой снимок (см. алгоритм 1)

**errorMessage** = пустая строка

**errorID = EVENT\_NONE** (0)

При неудаче:

```
result = null
```

**errorMessage** = 'LANG\_ERROR\_LIST\_AVAILABLE\_GAMES'      сообщение об ошибке

<b>errorID = EVENT_SERVER_ERROR</b>	(64)	код ошибки
-------------------------------------	------	------------

}

**{URL\_API}/createGame/{FID}.json**

Создаёт новую игру заданного типа.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameType</b> = GAME_TYPE_ID	идентификатор типа игры

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

{

<b>msgID</b> = MSG_CREATED_GAME	(10006)
---------------------------------	---------

**frameID** = FRAME\_ID

При успехе:

<b>result</b> = <i>'snapshot'</i>	игровой снимок (см. алгоритм 1)
-----------------------------------	---------------------------------

**errorMessage** = пустая строка

<b>errorID</b> = EVENT_NONE	(0)
-----------------------------	-----

При неудаче:

**result** = *null*

<b>errorMessage</b> = <i>'LANG_ERROR_CREATE_GAME'</i>	сообщение об ошибке
---	---------------------

<b>errorID</b> = EVENT_SERVER_ERROR	(64)	код ошибки
-------------------------------------	------	------------

}

## **{URL\_API}/removeGame/{FID}.json**

Удаляет заданную игру, если её владельцем является текущий пользователь.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_REMOVED_GAME      (10007)  
  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка
```

```
errorID = EVENT_NONE            (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_ERROR_REMOVE_GAME'    сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)        код ошибки  
  
}
```



## **{URL\_API}/invitePlayer/{FID}.json**

Меняет статус заданного игрока из «хочет принять участие в игре» в статус «принят в игру» для заданной игры, если её владельцем является текущий пользователь.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>userID</b> = USER_ID	идентификатор игрока

[GET]

<b>{FID}</b> = FRAME_FID	идентификатор фрейма
--------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_INVITED_PLAYER      (10008)  
  
  frameID = FRAME_ID
```

При успехе:

**result** = 'snapshot1|+|snapshot2|+|...snapshotN', где *snapshot* – это игровой снимок (см. алгоритм 1)

**errorMessage** = пустая строка

<b>errorID</b> = EVENT_NONE	(0)
-----------------------------	-----

При неудаче:

**result** = null

<b>errorMessage</b> = 'LANG_ERROR_INVITE_PLAYER'	сообщение об ошибке
--	---------------------

<b>errorID</b> = EVENT_SERVER_ERROR	(64)	код ошибки
-------------------------------------	------	------------

```
}
```

## **{URL\_API}/removePlayer/{FID}.json**

Меняет статус заданного игрока из «принят в игру» в статус «хочет принять участие в игре» для заданной игры, если её владельцем является текущий пользователь.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>userID</b> = USER_ID	идентификатор игрока

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_REMOVED_PLAYER      (10009)  
  
  frameID = FRAME_ID
```

При успехе:

**result** = 'snapshot1|+|snapshot2|+|...snapshotN', где *snapshot* – это игровой снимок (см. алгоритм 1)

**errorMessage** = пустая строка

<b>errorID</b> = EVENT_NONE	(0)
-----------------------------	-----

При неудаче:

**result** = null

<b>errorMessage</b> = 'LANG_ERROR_REMOVE_PLAYER'	сообщение об ошибке
--	---------------------

<b>errorID</b> = EVENT_SERVER_ERROR	(64)	код ошибки
-------------------------------------	------	------------

```
}
```

**{URL\_API}/startGame/{FID}.json**

Запускает заданную игру на игру, если её владельцем является текущий пользователь.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры

[GET]

<b>{FID}</b> = FRAME_ID	фрейм игры
-------------------------	------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_STARTED_GAME      (10016)  
  
  frameID = FRAME_ID
```

При успехе:

<b>result</b> = <i>'snapshot'</i>	игровой снимок (см. алгоритм 1)
<b>errorMessage</b> = пустая строка	
<b>errorID</b> = EVENT_NONE	(0)

При неудаче:

<b>result</b> = <i>null</i>	
<b>errorMessage</b> = <i>'LANG_ERROR_START_GAME'</i>	сообщение об ошибке
<b>errorID</b> = EVENT_SERVER_ERROR	(64) код ошибки

```
}
```

## **{URL\_API}/distributionCards/{FID}.json**

Устанавливает статус заданной игры в «раздачу карт» для заданной игры, если её владельцем является текущий пользователь.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_NONE           (10002)  
  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка  
  
errorID = EVENT_NONE        (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_ERROR_REQUEST_DEAL_CARDS'    сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)              код ошибки  
  
}
```

**{URL\_API}/leftGame/{FID}.json**

Текущий пользователь покидает заданную игру.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>userID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры, если равно 0, то игрок покидает все игры

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  msgID = MSG_LEAVED_GAME | MSG_LEAVED_ALL_GAMES      (10010|10011)  
  frameID = FRAME_ID
```

При успехе:

**result** = 'snapshot1|+|snapshot2|+|...snapshotN', где *snapshot* – это игровой снимок (см. алгоритм 1)

**errorMessage** = пустая строка

**errorID** = EVENT\_NONE (0)

При неудаче:

**result** = null

**errorMessage** = 'LANG\_ERROR\_LEAVE\_GAME'

или 'LANG\_ERROR\_LEAVE\_ALL\_GAMES' сообщение об ошибке

**errorID** = EVENT\_SERVER\_ERROR (64) код ошибки

```
}
```

**{URL\_API}/requestCard/{FID}.json**

Запрашивает карту из колоды для заданной игры для заданного игрока текущим пользователем.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>userID</b> = USER_ID	идентификатор игрока

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

{

<b>msgID</b> = MSG_NONE	(10002)
-------------------------	---------

<b>frameID</b> = FRAME_ID
---------------------------

при успехе:

**result** = TRUE

**errorMessage** = пустая строка

<b>errorID</b> = EVENT_NONE	(0)
-----------------------------	-----

При неудаче:

**result** = FALSE

<b>errorMessage</b> = 'LANG_ERROR_REQUEST_CARD'	сообщение об ошибке
---	---------------------

<b>errorID</b> = EVENT_SERVER_ERROR	(64)	код ошибки
-------------------------------------	------	------------

}

**{URL\_API}/gameAttack/{FID}.json**

Атака заданной картой в заданной игре заданным игроком, который должен быть текущим пользователем.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>userID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>card</b> = CARD_HASH	хеш карты (см. алгоритм 3), 0 – пропустить ход

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_NONE           (10002)  
  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка  
  
errorID = EVENT_NONE        (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_ERROR_GAME_ATTACK'    сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)      код ошибки  
  
}
```

**{URL\_API}/gameDefend/{FID}.json**

Отбой заданной картой в заданной игре для заданного игрока, который является текущим пользователем.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>userID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>card</b> = CARD_HASH	хеш карты (см. алгоритм 3), 0 – взять карты

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_NONE           (10002)  
  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка  
  
errorID = EVENT_NONE        (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_ERROR_GAME_STRIKEOFF'    сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)          код ошибки  
  
}
```



## {URL\_API}/callShiftUp/{FID}.json

Перевод заданной картой в заданной игре для заданного игрока, который должен быть текущим пользователем.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>userID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>card</b> = CARD_HASH	хеш карты (см. алгоритм 3), 0 – не переводить ход

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  msgID = MSG_NONE          (10002)  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка  
  
errorID = EVENT_NONE      (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_ERROR_GAME_SHIFTUP'    сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)      код ошибки  
  
}
```

## **{URL\_API}/joinGame/{FID}.json**

Текущий пользователь просится в заданную игру. Его статус становится «желает принять участие в игре».

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>userID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  msgID = MSG_JOINED_TO_GAME      (10013)  
  frameID = FRAME_ID
```

При успехе:

**result** = 'snapshot1|+|snapshot2|+|...snapshotN', где *snapshot* – это игровой снимок (см. алгоритм 1)

**errorMessage** = пустая строка

<b>errorID</b> = EVENT_NONE	(0)
-----------------------------	-----

При неудаче:

**result** = *null*

<b>errorMessage</b> = LANG_ERROR_JOIN_TO_GAME	сообщение об ошибке
---	---------------------

<b>errorID</b> = EVENT_SERVER_ERROR	(64)	код ошибки
-------------------------------------	------	------------

```
}
```

**{URL\_API}/defendNPC/{FID}.json**

Производит защиту заданным игроком NPC в заданной игре, если её владельцем является текущий пользователь.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>userID</b> = USER_ID	идентификатор игрока

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  msgID = MSG_NONE          (10002)  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка  
  
errorID = EVENT_NONE       (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_ERROR_REQUEST_DEFEND_NPC'    сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)              код ошибки  
  
}
```

## **{URL\_API}/attackNPC/{FID}.json**

Производит атаку заданным игроком NPC в заданной игре, если её владельцем является текущий пользователь.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры
<b>userID</b> = USER_ID	идентификатор игрока

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_NONE           (10002)  
  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка  
  
errorID = EVENT_NONE        (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_ERROR_REQUEST_ATTACK_NPC'    сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)              код ошибки  
  
}
```

**{URL\_API}/ratingList/{FID}.json**

Возвращает рейтинг игроков.

Аргументы:

{POST}

**hid** = CLIENT\_HID

HID пользователя

**limit** = LIMIT

максимальное количество записей, по умолчанию 5

[GET]

**{FID}** = FRAME\_ID

идентификатор фрейма

Возвращает в формате JSON:

{

**msgID** = MSG\_LIST\_RATINGS

(10014)

**frameID** = FRAME\_ID

При успехе:

**result** = 'userentity1|-|userentity2|-|...userentityN', где *userentity* – это снимок сущности игрока (см. алгоритм 2)

**errorMessage** = пустая строка

**errorID** = EVENT\_NONE

(0)

При неудаче:

**result** = *null*

**errorMessage** = 'LANG\_ERROR\_RATING\_LIST' сообщение об ошибке

**errorID** = EVENT\_SERVER\_ERROR

(64)

код ошибки

}

**{URL\_API}/resetTrigger/{FID}.json**

Сброс триггера заданной игры. Триггер необходимо сбрасывать перед каждым действием игрока.

Аргументы:

[POST]

**hid** = CLIENT\_HID                      HID пользователя

**gameID** = GAME\_ID                      идентификатор игры

[GET]

**{FID}** = FRAME\_ID                      идентификатор фрейма

Возвращает в формате JSON:

 $\{$ **msgID** = MSG\_NONE (10002)

**frameID = FRAME\_ID**

При успехе:

```
result = TRUE
```

**errorMessage** = пустая строка

**errorID** = EVENT\_NONE (0)

При неудаче:

```
result = FALSE
```

**errorMessage** = 'LANG\_ERROR\_RESET\_TRIGGER'      сообщение об ошибке

<b>errorID = EVENT_SERVER_ERROR</b>	(64)	код ошибки
-------------------------------------	------	------------

}

**{URL\_API}/signUp/{FID}.json**

Авторизация пользователя заданным логином и паролем.

Аргументы:

[POST]

<b>login</b> = LOGIN	ЛОГИН
----------------------	-------

**password** = PASSWORD      пароль

[GET]

<b>{FID}</b> =FRAME_ID	идентификатор фрейма
------------------------	----------------------

Возвращает в формате JSON:

 $\{$ 

```
msgID = MSG_AUTH_RESPONSE      (10003)
```

**frameID** = FRAME\_ID

При успехе:

**result** = 'authmodel'                  снимок модели авторизации (см. алгоритм 4)

**errorMessage** = пустая строка

**errorID = EVENT\_NONE** (0)

При неудаче:

```
result = null
```

**errorMessage** = 'LANG\_SIGNUP\_ERROR'      сообщение об ошибке

**errorID** = EVENT\_AUTH\_ERROR\_SIGNUP (40) код ошибки

}

**{URL\_API}/logout/{FID}.json**

Разлогинивание текущего пользователя. Возвращает данные аутентификации.

Аргументы:

[POST]

**hid** = CLIENT\_HID                      HID пользователя

**ownerID** = CLIENT\_ID                      идентификатор пользователя

[GET]

**{FID}** = FRAME\_ID                      идентификатор фрейма

Возвращает в формате JSON:

 $\{$ **msgID** = MSG\_NONE (10002)

**frameID = FRAME\_ID**

При успехе:

```
result = TRUE
```

**errorMessage** = пустая строка

**errorID** = EVENT\_NONE (0)

При неудаче:

```
result = FALSE
```

**errorMessage** = 'LANG\_USER\_NOT\_FOUND'      сообщение об ошибке

**errorID = EVENT\_SERVER\_ERROR** (64) код ошибки

}



**{URL\_API}/checkAuth/{FID}.json**

Проверка на авторизированность.

Аргументы:

[POST]

```
hid = CLIENT_HID
```

HID пользователя

[GET]

**{FID}** = FRAME\_ID

идентификатор фрейма

Возвращает в формате JSON:

 $\{$ 

```
msgID = MSG_AUTH_RESPONSE      (10003)
```

**frameID = FRAME\_ID**

При успехе:

**result** = '*authmodel*'                  снимок модели авторизации (см. алгоритм 4)

**errorMessage** = пустая строка

$$\text{errorID} = \text{EVENT\_NONE} \quad (0)$$

При неудаче:

```
result = null
```

**errorMessage** = 'LANG\_USER\_NOT\_AUTHORIZED'    сообщение об ошибке

<b>errorID</b> = EVENT_SERVER_ERROR	(64)	код ошибки
-------------------------------------	------	------------

}

**{URL\_API}/register/{FID}.json**

Регистрация нового пользователя.

Аргументы:

[POST]

<b>login</b> = LOGIN	логин
<b>password</b> = PASSWORD	пароль
<b>name</b> = FULLNAME	полное имя пользователя

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_AUTH_RESPONSE      (10003)  
  
  frameID = FRAME_ID
```

При успехе:

<b>result</b> = 'authmodel'	снимок модели авторизации (см. алгоритм 4)
<b>errorMessage</b> = пустая строка	
<b>errorID</b> = EVENT_NONE	(0)

При неудаче:

<b>result</b> = null	
<b>errorMessage</b> = 'LANG_ERROR_REGISTRY'	сообщение об ошибке
<b>errorID</b> = EVENT_AUTH_ERROR_REGISTRATION	(41) код ошибки

```
}
```

**{URL\_API}/saveProfile/{FID}.json**

сохраняет текущий профиль пользователя

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>userID</b> = CLIENT_ID	идентификатор пользователя
<b>image</b> = IMAGE	закодированное изображение ('ссылка BASE64(data)')
<b>name</b> = FULLNAME	полное имя пользователя

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_NONE           (10002)  
  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка
```

```
errorID = EVENT_NONE        (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_USER_NOT_FOUND'      сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)      код ошибки  
  
}
```

**{URL\_API}/deleteProfile/{FID}.json**

Удаляет профиль пользователя.

Аргументы:

[POST]

```
hid = CLIENT_HID
```

HID пользователя

**userID = CLIENT\_ID**

идентификатор пользователя

[GET]

**{FID}** = FRAME\_ID

идентификатор фрейма

Возвращает в формате JSON:

 $\{$ 

```
msgID = MSG_NONE
```

(10002)

**frameID** = FRAME\_ID

При успехе:

```
result = TRUE
```

**errorMessage** = пустая строка

```
errorID = EVENT_NONE
```

(0)

При неудаче:

```
result = FALSE
```

```
errorMessage = 'LANG_USER_NOT_FOUND'
```

сообщение об ошибке

```
errorID = EVENT_SERVER_ERROR
```

(64)

код ошибки

}

**{URL\_API}/costGames/{FID}.json**

Возвращает стоимость игр.

Аргументы:

[POST]

[GET]

**{FID}** = FRAME\_ID                      идентификатор фрейма

Возвращает в формате JSON:

{

**msgID** = MSG\_COST\_GAMES              (10019)

**frameID** = FRAME\_ID

**costs** = '*cost1|cost2|cost3|cost4*'      где cost – это стоимость соответствующей игры в виртуальных монетах

**balanceURL** = '*balanceURL*'              ссылка на страницу платёжной системы или пустая строка

**timestamp** = '*serverTimeStamp*'              текущее время сервера в формате RFC2822

}

**{URL\_API}/listSavingGames/{FID}.json**

Возвращает список «сохранялок».

Аргументы:

[POST]

**hid** = CLIENT\_HID

HID пользователя

**userID** = CLIENT\_ID

идентификатор пользователя

[GET]

**{FID}** = FRAME\_ID

идентификатор фрейма

Возвращает в формате JSON:

{

**msgID** = MSG\_LIST\_SAVING\_GAMES (10020)

**frameID** = FRAME\_ID

При успехе:

**result** = 'snapshot1|+|snapshot2|+|...snapshotN', где *snapshot* – это игровой снимок (см. алгоритм 1)

**errorID** = EVENT\_NONE (0)

**errorMessage** = пустая строка

При неудаче:

**result** = null

**errorMessage** = 'LANG\_ERROR\_LIST\_SAVING\_GAMES' сообщение об ошибке

**errorID** = EVENT\_SERVER\_ERROR (64) код ошибки

}

**{URL\_API}/restoreGame/{FID}.json**

Восстанавливает заданную игру.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>ownerID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_RESTORED_GAME      (10021)  
  
  frameID = FRAME_ID
```

При успехе:

<b>result</b> = <i>snapshot</i>	игровой снимок (см. алгоритм 1)
<b>errorMessage</b> = пустая строка	
<b>errorID</b> = EVENT_NONE	(0)

При неудаче:

<b>result</b> = <i>null</i>	
<b>errorMessage</b> = 'LANG_ERROR_RESTORE_GAMES'	сообщение об ошибке
<b>errorID</b> = EVENT_SERVER_ERROR	(64) код ошибки

```
}
```

**{URL\_API}/sendMessage/{FID}.json**

Посылает сообщение в игровой чат.

Аргументы:

[POST]

<b>hid</b> = CLIENT_HID	HID пользователя
<b>userID</b> = CLIENT_ID	идентификатор пользователя
<b>gameID</b> = GAME_ID	идентификатор игры, 0 – общий чат
<b>msg</b> = MESSAGE	сообщение

[GET]

<b>{FID}</b> = FRAME_ID	идентификатор фрейма
-------------------------	----------------------

Возвращает в формате JSON:

```
{  
  
  msgID = MSG_NONE          (10002)  
  
  frameID = FRAME_ID
```

При успехе:

```
result = TRUE  
  
errorMessage = пустая строка
```

```
errorID = EVENT_NONE      (0)
```

При неудаче:

```
result = FALSE  
  
errorMessage = 'LANG_ERROR_MESSAGE_ADD'    сообщение об ошибке  
  
errorID = EVENT_SERVER_ERROR    (64)      код ошибки  
  
}
```



**{URL\_API}/chatMessages/{FID}.json**

Возвращает список последних сообщений игрового чата.

Аргументы:

[POST]

**hid** = CLIENT\_HID

HID пользователя

**gameID** = GAME\_ID

идентификатор игры, 0 –общий чат

[GET]

**{FID}** = FRAME\_ID

идентификатор фрейма

Возвращает в формате JSON:

{

**msgID** = MSG\_CHAT\_MESSAGES (10022)

**frameID** = FRAME\_ID

При успехе:

**result** = 'message1|+|message2|+|...messageN'

где *message* – снимок сообщения  
(см. алгоритм 5)

**errorMessage** = пустая строка

**errorID** = EVENT\_NONE (0)

При неудаче:

**result** = null

**errorMessage** = LANG\_ERROR\_CHAT\_MESSAGES сообщение об ошибке

**errorID** = EVENT\_SERVER\_ERROR (64) код ошибки

}

## Алгоритм 1. Формирования игрового снимка.

```
/*
 * функция сериализации игровой карты
 * $suit      :int    - масть карты [0..3]
 * $rank      :int    - значение карты [0..14]
 */
function serializeCard($suit, $rank) {
    $text = '0123456789ABCDEF';
    return substr($text, $suit, 1) . substr($text, $rank, 1);
}

/*
 * функция сериализации карты игрока
 */
function serializePlayerCard(карта) {
    return json_encode(array(
        'card' => serializeCard(масть, значение),
        'selected' => булево значение, показывающее выборку карты игроком,
        'enabled' => булево значение, показывающее возможность хода картой,
        'hide' => булево значение, показывающее показывать ли лицевую
                    сторону карты или нет
    ));
}

/*
 * функция сериализации игрока
 */
function serializePlayer(игрок) {
    return json_encode(array(
        'name' => имя игрока,
        'type' => числовой идентификатор типов игроков,
        'id' => числовой идентификатор игрока,
        'action' => числовой идентификатор действия игрока,
        'description' => описание игрока,
        'imageSrc' => ссылка на изображение аватара,
        'status' => числовой идентификатор статуса игрока,
        'balance' => баланс игрока в виртуальных монетах,
        'cards' => serializePlayerCard(карта 1)
                    . '|' .
                    serializePlayerCard(карта 2)
                    . '|' .
                    ...
                    serializePlayerCard(последняя карта игрока)
    ));
}
```

```
/*
 * формируем игровой снимок
 */
$gameSnapshot = json_encode(array(
    'gameID' => числовой идентификатор игры,
    'gameOwnerID' => числовой идентификатор владельца игры,
    'gameAttackerID' => числовой идентификатор атакующего игрока,
    'gameDefenderID' => числовой идентификатор защищающегося игрока,
    'gameType' => числовой идентификатор типа игры [0..3],
    'gameStatus' => числовой идентификатор статуса игры [0..8],
    'actionID' => числовой идентификатор игрового действия [0..2],
    'pack' => json_encode(array(
        'size' => количество карт в колоде,
        'trump' => serializeCard(козырная масть, значение карты)
    )),
    'desktop' => json_encode(array(
        'attack' => serializeCard(масть 1, значение 1)
        . '|' .
        serializeCard(масть 2, значение 2)
        . '|' .
        ...
        serializeCard(масть 6, значение 6),
        'defend' => serializeCard(масть 1, значение 1)
        . '|' .
        serializeCard(масть 2, значение 2)
        . '|' .
        ...
        serializeCard(масть 6, значение 6)
    )),
    'players' => serializePlayer(игрок 1)
    . '|*' .
    serializePlayer(игрок 2)
    ... - добавляем всех игроков, разделяя их строкой '|*|'
));
```

## Алгоритм 2. Формирование снимка сущности игрока.

```
/*
 * функция сериализации игровой статистики
 */
function serializeGameStat(игра) {
    return json_encode(array(
        'gamesStarted' => количество начатых игр,
        'gamesEnded' => количество законченных игр,
        'gamesWins' => количество игр законченных победой,
        'gamesLoses' => количество игр законченных с поражением,
        'gamesNeutrals' => количество игр завершённых в ничью,
        'gamesErWins' => количество побед с погонами,
        'gamesErLoses' => количество поражений с погонами
    ));
}

/*
 * формируем снимок сущности игрока
 */
$snapshotUserEntity = json_encode(array(
    'uid' => числовой идентификатор игрока (идентификатор пользователя),
    'name' => имя игрока,
    'type' => числовой идентификатор типа игрока,
    'imageSrc' => ссылка на аватар,
    'description' => описание игрока,
    'balance' => баланс игрока в виртуальных монетах,
    'lastGameTime' => время последней игры в формате RFC2822,
    'lastGameID' => числовой идентификатор последней игры,
    'rating' => рейтинг игрока в формате числа с плавающей точкой,
    'stats' => serializeGameStat(подкидной дурак)
        . '|' .
        serializeGameStat(переводной дурак)
        . '|' .
        serializeGameStat(подкидной дурак пара на пару)
        . '|' .
        serializeGameStat(переводной дурак пара на пару)
));
```

### Алгоритм 3. Формирование хеша игровой карты.

```
/*  
 * формирование хеша игровой карты  
 * $suit    :int      - числовое значение масти карты [0..3]  
 * $rank    :int      - числовое значение значения карты [0..14]  
 */  
$hash = $suit * 15 + $rank;
```

## Алгоритм 4. Формирование снимка модели авторизации.

```
/*
 * функция сериализации игровой статистики
 */
function serializeGameStat(игра) {
    return json_encode(array(
        'gamesStarted' => количество начатых игр,
        'gamesEnded' => количество законченных игр,
        'gamesWins' => количество игр законченных победой,
        'gamesLoses' => количество игр законченных с поражением,
        'gamesNeutrals' => количество игр завершённых в ничью,
        'gamesErWins' => количество побед с погонами,
        'gamesErLoses' => количество поражений с погонами
    ));
}

/*
 * формируем снимок модели авторизации
 */
$snapshotAuthModel = json_encode(array(
    'hid' => строка идентификатор (токен) пользователя (HID),
    'uid' => числовой идентификатор игрока (идентификатор пользователя),
    'name' => имя игрока,
    'type' => числовой идентификатор типа игрока,
    'imageSrc' => ссылка на аватар,
    'description' => описание игрока,
    'balance' => баланс игрока в виртуальных монетах,
    'lastGameTime' => время последней игры в формате RFC2822,
    'lastGameID' => числовой идентификатор последней игры,
    'rating' => рейтинг игрока в формате числа с плавающей точкой,
    'stats' => serializeGameStat(подкидной дурак)
        . '|' .
        serializeGameStat(переводной дурак)
        . '|' .
        serializeGameStat(подкидной дурак пара на пару)
        . '|' .
        serializeGameStat(переводной дурак пара на пару)
));
```

## Алгоритм 5. Формирование снимка сообщения.

```
/*
 * формирование снимка сообщения
 */
$snapshotChatMessage = json_encode(array(
    'id' => числовой идентификатор сообщения,
    'uid' => числовой идентификатор автора сообщения,
    'name' => имя автора сообщения,
    'imageSrc' => ссылка на аватар автора сообщения,
    'message' => текст сообщения,
    'time' => время отправки сообщения в формате RFC2822
));
```

## Приложение 1. Типы игры.

ЗНАЧЕНИЕ	ОПИСАНИЕ
0	подкидной дурак
1	переводной дурак
2	подкидной дурак пара на пару
3	переводной дурак пара на пару



## Приложение 2. Игровые статусы игровой модели.

ЗНАЧЕНИЕ	ОПИСАНИЕ
0	игра не создана
1	идёт набор игроков, к данной игре можно присоединиться
2	идёт тасование колоды
3	идёт раздача карт
4	идёт игра
5	игрок не смог отбиться и взял карты, другие игроки подкидывают ему ещё карты
6	игрок успешно отбился
7	игра окончена
8	игра прервана

### Приложение 3. Игровые действия.

ЗНАЧЕНИЕ	ОПИСАНИЕ
0	идёт атака
1	идёт защита

## Приложение 4. Типы игроков.

ЗНАЧЕНИЕ	ОПИСАНИЕ
0	реальный игрок
1	компьютерный персонаж

## Приложение 5. Действия игрока.

ЗНАЧЕНИЕ	ОПИСАНИЕ
0	бездействие
1	игрок атакует
2	игрок отбивается
3	игрок набирает карты из колоды
4	игрок переводит ход

## Приложение 6. Игровые статусы игрока.

ЗНАЧЕНИЕ	ОПИСАНИЕ
0	неопределённый статус
1	владелец игры, производящий набор команды
2	игрок желает вступить в команду игроков
3	игрока взяли в команду игроков
4	игрок играет в игру
5	игрок потерпел поражение
6	игрок победил
7	игрок покинул игру

## Приложение 7. Игральные карты.

### Масти.

ЗНАЧЕНИЕ	ОПИСАНИЕ
0	пики
1	крести
2	буби
4	черви

### Значения карт.

ЗНАЧЕНИЕ	ОПИСАНИЕ
0	нет карты
1	джокер
2	двойка
3	тройка
4	четвёрка
5	пятёрка
6	шестёрка
7	семёрка
8	восьмёрка
9	девятка
10	десятка
11	валет
12	дама
13	король
14	туз