

Оглавление

1. WebIUS Browser	2
1.1 Общее описание	2
1.2 Архитектура WebIUS Browser	2
2. screen.js	5
2.1 Пример создания экрана с кнопками	8
2.2 Пример создания экрана с полем Input	10
3. support.htm	12
4. API функции WebIUS Browser	13

1. WebIUS Browser

1.1 Общее описание

WebIUS Browser является интерфейсной надстройкой к ПО **TelIME7**, которая представляет собой Windows Forms приложение, содержащее в себе стандартный браузер Internet Explorer, Google Chrome. Данная интеграционная платформа включает в себя функционал программного обеспечения **TelIME7**, а также возможность расширения доступного набора функций, за счет подключения внешних сервисов.

При помощи **WebIUS Browser** решаются следующие задачи:

- Реализация пользовательского графического интерфейса.
- Реализация сценария бизнес-логики обслуживания клиента.
- Управление оборудованием через взаимодействие с API **TelIME7**.
- Управление прохождением NDC-сценария.
- Возможность взаимодействия с удаленными внешними банковскими сервисами.
- Возможность расширения функционала за счет подключения разнообразных плагинов, позволяющих управлять нестандартной дополнительной периферией из сценария бизнес-логики обслуживания клиента, а также интегрировать стороннее программное обеспечение.

1.2 Архитектура WebIUS Browser

WebIUS Browser состоит из трех логических частей (см. рис. 1):

1. **User Interface** (html) – пользовательский интерфейс, который представляет собой отображаемые на экране устройства страницы с графическими элементами.

В **WebIUS Browser** пользовательский интерфейс реализован на шаблонах экранных форм с динамическим контентом, что обеспечивает разделение графического представления от бизнес-логики сценария. Такой подход позволяет добавлять новые ветки сценария без обращения к дизайнерам, заменять текущий набор экранных форм без внесения изменений в бизнес-логику, а также обеспечивает возможность управлять пакетами шаблонов в зависимости от ситуации: использование графики для людей с ограниченными возможностями, использование графики для привилегированных клиентов и т.п.

2. **Script** (JavaScript) – описание бизнес-логики сценария, в котором определяется необходимая последовательность шагов и действий, требуемых для выполнения клиентом при совершении той или иной операции, а также запросы, направленные к пользователю, банкомату и

хосту. Поскольку сценарий построен на событийной модели и выполняется асинхронно, предусматривается возможность использовать различные обработчики событий от пользователя, **TellME7**, процессинга или от других внешних систем.

Основным преимуществом данной модели является абсолютно независимая обработка событий. То есть выполнение бизнес-логики сценария и работа с клиентом происходит параллельно с выполнением обработчиков.

Реализованный в **WebIUS Browser** сценарий не имеет жесткой привязки к платежной системе **TellME7**, в рамках которой происходит работа с платежными средствами. Это обеспечивает гибкость создаваемой логики, которая достигается благодаря отсутствию описания работы с NDC-сценарием в скрипте бизнес-логики и вынесению его в другой класс – **Support NDC**.

3. **Support NDC** – связующее звено между скриптом бизнес-логики, NDC-сценарием и **TellME7**.

Support NDC является промежуточным модулем, содержащим описание работы с NDC-сценарием, в котором единожды прописываются все используемые опкоды, буферы и стейты для взаимодействия с NDC-хостом. Все дальнейшие изменения вносятся в **Script**. Таким образом, становится возможным абстрагироваться от NDC-протокола и проводить операции независимо от его пошаговой архитектуры.

Главное преимущество отделения сценария бизнес-логики от работы с NDC-сценарием заключается в возможности сохранить пользовательский опыт и интеграцию с другими банковскими подсистемами при смене используемого протокола взаимодействия с процессингом банка без внесения в них каких-либо изменений.

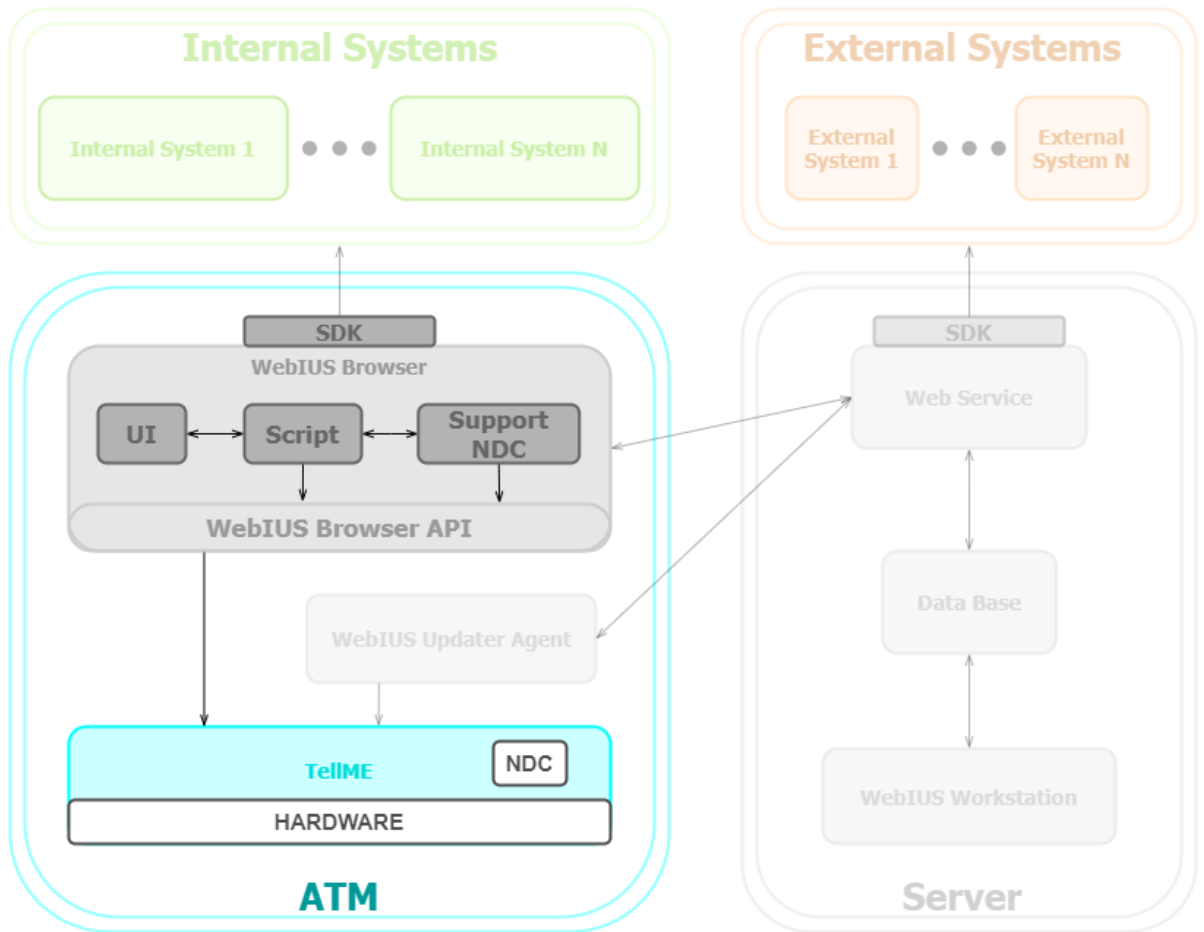


Рис. 1 - WebIUS Browser

2. screen.js

Центральным понятием сценария является сессия. Под сессией следует понимать совокупность данных, связанных с одним устройством самообслуживания в течение одного сеанса обслуживания. Каждая сессия может находиться в одном из специфицированных в скрипте состояний. Такое состояние приблизительно соответствует некоторому виду экрана в устройстве самообслуживания, поэтому в **WebIUS Browser** для описания такого состояния используется класс **screen.js**.

Для каждого состояния сессии может быть описан набор переменных и реакции на предусмотренные в системе события. Переменные используются как хранилище данных о сессии. Реакции на события записываются в виде последовательности операторов JavaScript языка. В системе предусмотрен фиксированный набор событий, выраженный в методах класса **screen.js**:

Свойства и методы класса screen.js

Наименование	Описание
Работа с обработчиками событий	
<code>scr.addButtonCancel (WebIUS_callback)</code>	Добавление обработчика нажатия кнопки Cancel Еpp-клавиатуры. Параметры: WebIUS_callback – функция, исполняемая при нажатии кнопки Cancel.
<code>scr.addButtonEnter (WebIUS_callback)</code>	Добавление обработчика нажатия кнопки Enter Еpp-клавиатуры. Параметры: WebIUS_callback – функция, исполняемая при нажатии кнопки Enter.
<code>scr.addCall (event, callback)</code>	Добавление обработчика события. Параметры: event – название события, callback – функция, вызываемая при срабатывании события event.
<code>scr.addNDCCaller (callback)</code>	Добавление обработчика события NDC. Параметры: callback – функция, вызываемая при срабатывании события event.
<code>scr.addOnError (WebIUS_callback)</code>	Добавление обработчика события типа «ошибка». Параметры: WebIUS_callback – функция, исполняемая при возникновении ошибки.

Работа с элементами экрана

<code>scr.buttonExists (buttonName)</code>	Проверка существования кнопки. Параметры: buttonName – имя кнопки.
--	--

<code>scr.setButtonJson (buttonObj, buttonCallback)</code>	Наполнение элемента <code>button</code> информацией. Имеет следующие параметры: <code>buttonObj</code> – объект <code>button</code> , <code>buttonCallback</code> – функция, вызываемая при нажатии <code>button</code> .
<code>scr.setExtJson (extObj)</code>	Наполнение дополнительной информации, передаваемой в экран. Параметры: <code>extObj</code> – объект дополнительной информации.
<code>scr.setImageJson (imageObj)</code>	Наполнение элемента <code>image</code> информацией. Имеет следующие параметры: <code>imageObj</code> – объект <code>image</code> .
<code>scr.setInputJson (inputObj, inputCallback)</code>	Наполнение элемента <code>input</code> . Имеет следующие параметры: <code>inputObj</code> – объект <code>input</code> , <code>inputCallback</code> – функция, вызываемая при вводе данных в поле <code>input</code> .
<code>scr.setLabelJson (labelObj)</code>	Наполнение элемента <code>label</code> . Имеет следующие параметры: <code>labelObj</code> – объект <code>label</code> .
<code>scr.setListJson (listObj, listCallback)</code>	Наполнение элемента <code>list</code> . Имеет следующие параметры: <code>listObj</code> – объект <code>list</code> , <code>listCallback</code> – вызываемая функция.
<code>scr.setModalMessageJson (modalObj, modalCall)</code>	Наполнение модальных экранов информацией. Имеет следующие параметры: <code>modalObj</code> – объект модальное окно, <code>modalCall</code> – вызываемая функция.
<code>scr.deleteButton (buttonName)</code>	Удаление <code>Json</code> объекта <code>button</code> . Параметры: <code>buttonName</code> – имя кнопки.
<code>scr.deleteInput (inputName)</code>	Удаление <code>Json</code> объекта <code>input</code> . Параметры: <code>inputName</code> – имя поля ввода.
<code>scr.deleteLabel (labelName)</code>	Удаление <code>Json</code> объекта <code>label</code> . Параметры: <code>labelName</code> – имя лейбла.
<code>scr.deleteList (listName)</code>	Удаление <code>Json</code> объекта <code>list</code> . Параметры: <code>listName</code> – имя списка.
<code>scr.deleteModalMessage ()</code>	Удаление <code>Json</code> объекта <code>modalMessage</code> .
<code>scr.deleteWait ()</code>	Удаление <code>Json</code> объекта <code>wait</code> .
<code>scr.scrElementExists (elementName)</code>	Проверка существования элемента экрана. Параметры: <code>elementName</code> – имя элемента.

Работа с экранами

<code>scr.clearScr ()</code>	Очищение экрана.
<code>scr.name</code>	Имя экрана.
<code>scr.nextScreen (WebIUS_scrn)</code>	Смена экранов. Параметры: <code>WebIUS_scrn</code> – имя экрана.

<code>scr.nextScreen (WebIUS_scrn, WebIUS_params)</code>	Смена экранов. Параметры: WebIUS_scrn – имя экрана, WebIUs_params – дополнительные параметры.
<code>scr.refreshScreen ()</code>	Обновление информации экрана.
<code>scr.render (scrTypeName, scrAddress, scrCachClear)</code>	Заполнение экрана. Имеет следующие параметры: scrTypeName – тип экрана, scrAddress – URL-адрес для вызова страницы.

Другие функции

<code>scr.setTimeoutJson (outObj, timeCall)</code>	Назначение таймаутов. Параметры: outObj - время, timeCall - функция, вызываемая при истечении установленного времени.
--	---

При исполнении сценария первым этапом является инициализация всех используемых состояний сценария. В существующем сценарии это действие выполняет функция **initScreens()**.

2.1 Пример создания экрана с кнопками

```

/** Создание экрана с кнопками */
var screen_with_buttons = function () {

    /**
     * Назначение обработчика событий для кнопки «Назад»
     *
     * @param {object} someArgs - массив элементов, содержащий
имя кнопки,
     * а также результат произведенных действий на странице
     */
    var onbtn_return = function (someArgs) {
        /** Переход на предыдущий экран */
        scr.nextScreen(previous_screen);
    };

    /**
     * Назначение обработчика событий для кнопки «Продолжить»
     *
     * @param {object} someArgs - массив элементов, содержащий
имя кнопки,
     * а также результат произведенных действий на странице
     */
    var onbtn_continue = function (someArgs) {

        /** Переход на следующий экран */
        scr.nextScreen(next_screen);
    };

    /**
     * Создание элемента button на экране, назначение обработчика
нажатия
     * кнопки на экране
     */
    scr.setButtonJson({name:"Btn4",
text:getLangText("btn_return"), ext:""}, onbtn_return);
    scr.setButtonJson({name:"Btn8",
text:getLangText("btn_continue")}, onbtn_continue);

    /**
     * Создание элемента button на экране, назначение обработчика
нажатия
     * кнопок ЕРР-клавиатуры
     */
    scr.setButtonJson({name:"cancel",
text:getLangText("btn_return"), visible:false}, onbtn_return);
    scr.setButtonJson({name:"continue",
text:getLangText("btn_continue"), visible:false},
onbtn_continue);
};

```



```
/** Инициализация экранов */  
function initScreens() {  
  
    /**  
    * Создание объекта - экран по умолчанию  
    *  
    * @param {object} main - объект экрана  
    * @param {string} "" - имя экрана  
    */  
    scr = new Screen(main, "");  
  
    /**  
    * Создание объекта - экран с кнопками  
    *  
    * @param {object} screen_with_buttons - объект экрана с  
кнопками  
    * @param {string} "screen_with_buttons" - имя экрана  
    */  
    screen_with_buttons = new Screen(screen_with_buttons,  
"screen_with_buttons");  
};
```

2.2 Пример создания экрана с полем Input

```

/** Пример создания экрана с полем Input */
var screen_with_Input = function () {

    /** Переменная для хранения служебной информацией (номера
    карты) */
    msession.service.cardNum = "";

    /**
     * Назначение обработчика события для кнопки «Очистить»
     *
     * @param {object} someArgs - массив элементов,
    содержащий имя
     * кнопки, а также результат произведенных действий на
    странице
     */
    var OnClear = function (someArgs) {
        msession.service.cardNum = "";

        /**
         * Обновление элемента Input на экране, назначение
         * обработчика события
         */
        scr.setInputJson(inputObj:{name:"card_number",
text:session.service.cardNum, mask:"**** *",
maxLength:16}, onInput);
        /** Обновление информации экрана */
        scr.refreshScreen();
    };

    /**
     * Назначение обработчика события для поля Input
     *
     * @param {object} someArgs - массив элементов,
    содержащий имя поля
     * ввода, значение этого поля
     */
    var onInput = function(someArgs) {
        var pKey = someArgs[0], pValue = someArgs[1];
        session.service.cardNum = pValue;
    }

    /**
     * Создание элемента input на экране, назначение обработчика
    события
     * поля Input
     */
    scr.setInputJson(inputObj:{name:"card_number",
text:session.service.cardNum, mask:"**** *",
maxLength:16}, onInput);

```

```
/**
    Создание элемента button на экране, назначение обработчика
    нажатия
    кнопки на экране
*/
scr.setButtonJson({name:"Btn7",
text:getLangText("btn_clear"),ext:""}, onClear);

/**
    Создание элемента button на экране, назначение обработчика
    нажатия
    кнопок EPP-клавиатуры
*/
scr.setButtonJson({name:"clear",
text:getLangText('btn_clear'), visible:false}, onClear);
};

/** Инициализация экранов */
function initScreens() {

    /**
    * Создание объекта - экран по умолчанию
    *
    * @param {object} main - объект экрана
    * @param {string} "" - имя экрана
    */
    scr = new Screen(main,"");

    /**
    * Создание объекта - экран с полем Input
    *
    * @param {object} screen_with_Input - объект экрана с полем
    Input
    * @param {string} "screen_with_Input" - имя экрана
    */
    screen_with_Input = new Screen(screen_with_Input,
"screen_with_Input");
};
```

3. support.htm

support.htm является посредником между NDC сценарием, TellME и скриптом бизнес логики. В нем используются две основные функции:

1. `processFromTellME(param)` – функция для работы с запросами от TellME;
2. `processFromScript(forceReq)` – функция для работы с запросами от **Script.js**.

Для вызова **support.htm** из скрипта бизнес логики необходимо использовать следующую функцию, содержащую URL-адрес **support.htm**:

```
window.external.exchange.loadPage("help", "
C:/WebIUSBrowser/config/SCRIPT/support.htm?Reload=0&source=scrip
t&webius="+req+"&language="+session.lang);
```

Функция `loadPage()` имеет следующие параметры:

1. `help` – имя окна, в котором нужно загрузить URL-адрес. Имена окон задаются в файле **atm_web.exe.config**;
2. URL-адрес, состоящий из:
 - a. адреса, открываемой страницы;
 - b. параметр `Reload`, в котором могут быть выставлены следующие значения:
 - 1 – загрузить страницу в окно;
 - 0 – если страница уже загружена - не перезагружать её, вызвать метод **processSignalFromBrowser(param)** (в `param` передаются параметры, находящиеся в URL-адресе).
 - c. параметра `source`, указывающего на то, откуда пришел запрос:
 - `source=script` – из `Script.js`;
 - `source=tellme` – из `TellME`.

В зависимости от значения данного параметра в методе **processSignalFromBrowser()** определяются дальнейшие действия.

4. API функции WebIUS Browser

В **WebIUS Browser** также доступны функции, представленные в следующей таблице:

Наименование	Описание
Взаимодействие с аппаратной частью	
<code>int getAcceptNotesMaxCount ()</code>	Получить максимальное количество купюр для внесения за один раз.
<code>string getBCRData ()</code>	Получить данные от баркод ридера. Возвращает строку с полученными от сканера данными или строку с кодом ошибки в формате «ERROR_ON_GET_DATA:err_code», где err_code – числовое значение кода ошибки в шестнадцатеричном виде. Если err_code будет иметь значение «2A000048» (TIMEOUT), то можно дальше вызывать данный метод, но не чаще, чем раз в 150 миллисекунд или останавливать сканирование вызовом метода BCRStopScan. При других значениях err_code и при успешном считывании операция сканирования будет завершена автоматически.
<code>void startBCRScan ()</code>	Начать сканирование штрих-кода через баркод ридер.
<code>void stopBCRScan ()</code>	Закончить сканирование штрих-кода через баркод ридер.
<code>string beepStart (string ParameterValue)</code>	Начать бип. Данная функция может использоваться при необходимости обозначения какого-либо события звуковым сигналом. Параметры: ParameterValue – id звуковой дорожки.
<code>string beepStop (string ParameterValue)</code>	Закончить бип. Параметры: ParameterValue – id звуковой дорожки.
<code>string bimModuleManyNotes ()</code>	Получить тип купюроприемника. Возвращает «0» - тип купюроприемника - «покупюрник», «1» - тип купюроприемника – «пачечник».
<code>int getReceiptState ()</code>	Получить состояние чекового принтера. Возвращает «1» - доступен, «2» - недоступен.

<code>string getNDCCurrency ()</code>	Получить валюту принятой суммы. Возвращает числовое значение кода валюты, определенное в NDC.
<code>string getAllAcceptedNotes (string currency)</code>	Получить номинал и количество внесенных купюр. Параметры: <code>currency</code> – значение валюты. Возвращает строку с данными принятых купюр.
<code>int printReceipt (string str)</code>	Печать через чековый принтер. Возвращает результат выполнения функции: «0» - успешно отправлено на печать, «1» - ошибка. Параметры: <code>str</code> – данные на печать.
<code>void setJournalPrint (string data)</code>	Печать в журнальный принтер. Параметры: <code>data</code> – данные на печать.

Взаимодействие с NDC

<code>string getNDCAmount ()</code>	Получить текущее значение суммы определенное в NDC.
<code>string getNDCBufferValue (string _BufferName)</code>	Получить значение из NDC буфера. Параметры: <code>_BufferName</code> – имя буфера.
<code>string getNDCScreen (string ScreenName)</code>	Получить текст из NDC скрина с замененными непечатными символами. Параметры: <code>ScreenName</code> – имя скрина.
<code>string getNDCScreenOrig (string ScreenName)</code>	Получить текст из NDC скрина. Параметры: <code>ScreenName</code> – имя скрина.
<code>int getTimer (int number)</code>	Получить номер таймера NDC, например, номер таймера ожидания клиента, номера таймера ожидания забора купюр из купюроприемника и т.д. Параметры: <code>number</code> – номер таймера.
<code>void setNDCBufferValue (string _BufferName, string _BufferValue)</code>	Сохранить значение в NDC буфер. Параметры: <code>_BufferName</code> - имя буфера; <code>_BufferValue</code> – значение.
<code>void setNDCButton (int pos, string value)</code>	Нажать на кнопку в NDC. Параметры: <code>pos</code> – позиция кнопки, <code>value</code> - значение.
<code>string setNDCScreenText (string ScreenName, string ScreenText)</code>	Запись текста в файл NDC экрана. Возвращает «0» - при успешном завершении работы функции, «1» - при пустом значении <code>ScreenName</code> , «2» - при пустом значении <code>ScreenText</code> , «3» - при возникновении ошибки. Параметры: <code>ScreenName</code> – имя экрана; <code>ScreenText</code> - текст.

Работа с файлами

<code>int deleteFile (string FileName)</code>	Удалить файл. Возвращает «0» - при успешном удалении, «1» - при отсутствии файла, «2» - при возникновении ошибки. Параметры: FileName – имя файла.
<code>string readFile (string FileName)</code>	Прочитать файл. Возвращает либо содержание файла, либо пустую строку. Параметры: FileName – имя файла.
<code>int writeToFile (string FileName, string Text, bool AllowAppend)</code>	Записать в файл. Возвращает «0» - при успешной записи, «1» - при возникновении ошибки. Параметры: FileName – имя файла; Text - текст; AllowAppend – разрешить добавление.

Работа с GUI

<code>string setScrResult (object SCRResultJSON, string target = "")</code>	Обработка результата работы страницы GUI. Возвращает результат в формате JSON. Параметры: SCRResultJSON – объект, содержащий результат работы отображения/состояние работы страницы; target – предустановленное значение (id окна).
<code>string getScrData (string target = "")</code>	Получить объект, на основе которого будет построено отображение в GUI. Параметры: target – предустановленное значение (id окна).
<code>SCRData scrData</code>	Модуль работы с GUI элементами.
<code>void render (string scrTypeName, string scrCustomAddress = "", string cacheClear = "")</code>	Отправить на отрисовку. Параметры: scrTypeName – тип экрана (тип задается в файле конфигурации браузера); scrCustomAddress – полный путь до отображаемой страницы; cacheClear – зарезервированный служебный параметр.

Работа с временной памятью

<code>void clearMemory ()</code>	Очистить временную память.
<code>string getMemory (string name)</code>	Получить значение из временной памяти. Параметры: name – имя сохраненного поля данных.
<code>void setMemory (string name, string value)</code>	Сохранить значение во временную память. Параметры: name - имя сохраняемого поля данных, value - значение.

<code>void removeMemoryKey (string name)</code>	Удалить значение из временной памяти. Параметры: name - имя сохраненного поля данных.
---	---

Управление сессией

<code>int getInactiveTimeout ()</code>	Возвращает таймер неактивности на экране внесения наличных в секундах.
<code>void initSCSOnScreen ()</code>	Начать сессию по нажатию на экран. Функция используется на экране ожидания клиента.
<code>void setSCSButton (int pos, string value)</code>	Нажать на кнопку в TellME. Функция используется на экране ожидания клиента. Параметры: pos – позиция кнопки FDK (от 1 до 8), value - значение.

Работа со страницами

<code>void loadPage (string window, string url)</code>	Загрузка страницы в окне. При помощи данной функции происходит передача информации от формы к форме через url. Параметры: window – окно; url – url страницы.
--	--

Работа с историей операций клиента

<code>WebIUS_Plugin wbCustom</code>	Модуль запроса истории операций клиента. Для использования модуля необходим сервер WebIUS.
<code>string receiveCardHolderOperationsInfo (string id)</code>	Запрос истории по id клиента, выполняющийся асинхронно (при пустом id используется hash + соль от PAN карты). Возвращает пустую строку при непустом значении id или маскированный PAN карты при id = "". Параметры: id – id клиента.
<code>void saveCardHolderOperationInfo (string id, string dataJson)</code>	Сохранение JSON по id. Параметры: dataJson – Json данные.

Другое

<code>string newGuid ()</code>	Сгенерировать GUID. Возвращает GUID в формате xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.
--------------------------------	---

<pre>int writeSessionInfo (string Id, string Text)</pre>	Запись в отдельную папку sessions в файл с именем <Id>.session. Возвращает «0» - при успешном завершении функции, «1» - при возникновении ошибки. Параметры: Id – id сессии; Text -текст.
<pre>int writeLogTrace (string method, string str)</pre>	Запись в отдельный лог с расширением .trace. Возвращает «0» - при успешном завершении функции, «1» - при возникновении ошибки. Параметры: method - метод; str - строка.

Для их вызова используются глобальные функции.

Например, для получения текста из NDC скрина необходимо использовать следующую строку:

```
window.external.exchange.getNDCScreen(ScreenName)
```