Оглавление

1. WebIUS Browser

1.1 Общее описание

WebIUS Browser является интерфейсной надстройкой к ПО **TellME7**, которая представляет собой Windows Forms приложение, содержащее в себе стандартный браузер Internet Explorer, Google Chrome. Данная интеграционная платформа включает в себя функционал программного обеспечения **TellME7**, 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 **TellME7**.
- 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.

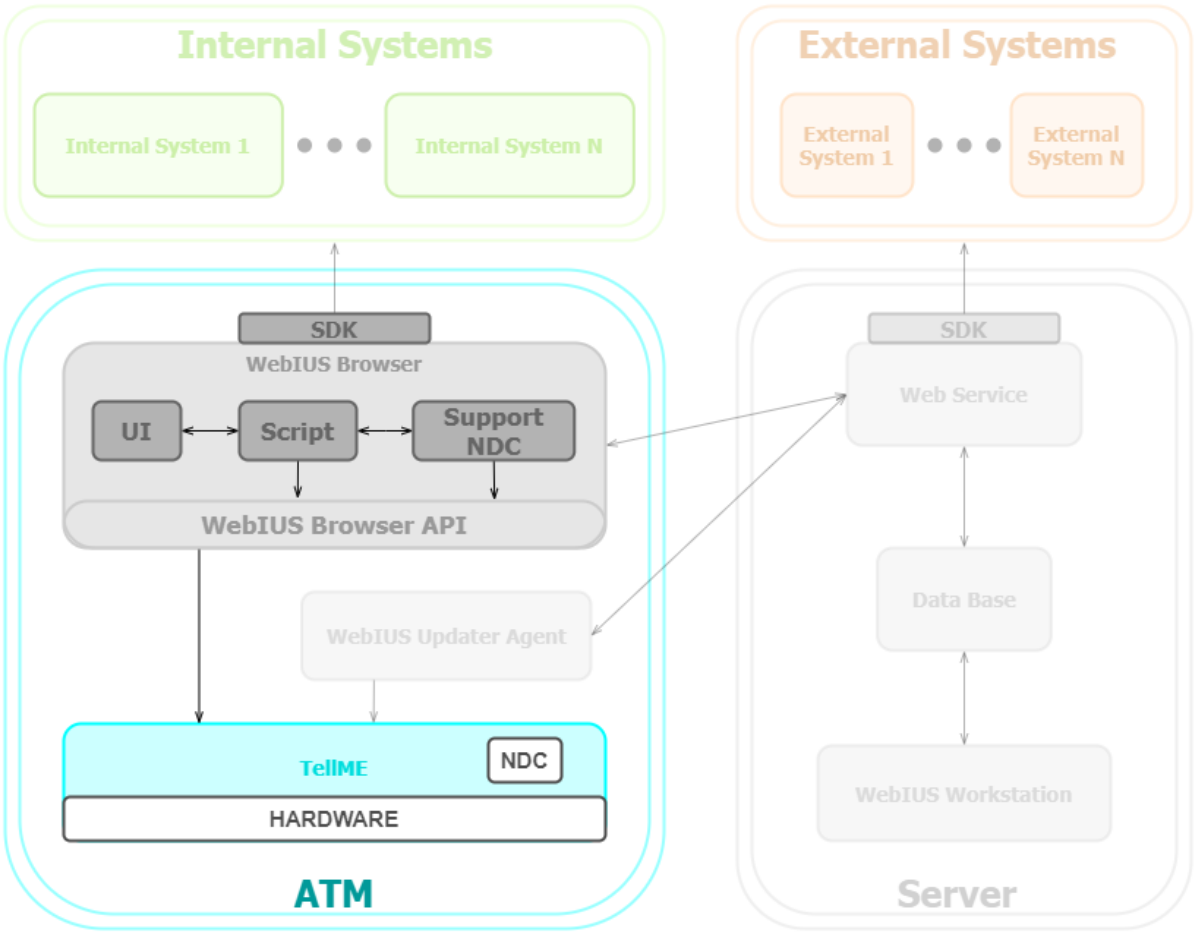

Рис. 1 - WebIUS Browser

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	
`scr.addButtonCancel (WebIUS_callback)`	Добавление обработчика нажатия кнопки Cancel Epp-клавиатуры. Параметры: WebIUS_callback – функция, исполняемая при нажатии кнопки Cancel.
`scr.addButtonEnter (WebIUS_callback)`	Добавление обработчика нажатия кнопки Enter Epp-клавиатуры. Параметры: WebIUS_callback – функция, исполняемая при нажатии кнопки Enter.
`scr.addCall (event, callback)`	Добавление обработчика события. Параметры: event – название события, callback – функция, вызывающаяся при срабатывании события event.
`scr.addNDCCaller (callback)`	Добавление обработчика события NDC. Параметры: callback – функция, вызывающаяся при срабатывании события event.
`scr.addOnError (WebIUS_callback)`	Добавление обработчика события типа «ошибка». Параметры: WebIUS_callback - функция, исполняемая при возникновении ошибки.
Работа с элементами экрана	
`scr.buttonExists (buttonName)`	Проверка существования кнопки. Параметры: buttonName – имя кнопки.

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

Работа с экранами

`scr.clearScr ()`	Очищение экрана.
`scr.name`	Имя экрана.
`scr.nextScreen (WebIUS_scrn)`	Смена экранов. Параметры: WebIUS_scrn – имя экрана.

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

Другие функции

`scr.setTimeOutJson (outObj, timeCall)`	Назначение таймаутов. Параметры: outObj - время, timeCall - функция, вызывающаяся 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 EPP-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	
`int getAcceptNotesMaxCount ()`	Получить максимальное количество купюр для внесения за один раз.
`string getBCRData ()`	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.
`void startBCRScan ()`	Начать сканирование штрих-кода через баркод ридер.
`void stopBCRScan ()`	Закончить сканирование штрих-кода через баркод ридер.
`string beepStart (string ParameterValue)`	Начать бип. Данная функция может использоваться при необходимости обозначения какого-либо события звуковым сигналом. Параметры: ParameterValue – id звуковой дорожки.
`string beepStop (string ParameterValue)`	Закончить бип. Параметры: ParameterValue – id звуковой дорожки.
`string bimModuleManyNotes ()`	Получить тип купюроприймника. Возвращает «0» - тип купюроприемника - «покупюрник», «1» - тип купюроприемника – «пачечник».
`int getReceiptState ()`	Получить состояние чекового принтера. Возвращает «1» - доступен, «2» - недоступен.

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

Взаимодействие с NDC

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

Работа с файлами

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

Работа с GUI

`string setScrResult (object SCRResultJSON, string target = "")`	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).
`string getScrData (string target = "")`	Получить объект, на основе которого будет построено отображение в GUI. Параметры: target – предустановленное значение (id окна).
`SCRData scrData`	Модуль работы с GUI элементами.
`void render (string scrTypeName, string scrCustomAddress = "", string cacheClear = "")`	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

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

`void removeMemoryKey (string name)`	Удалить значение из временной памяти. Параметры: name - имя сохраненного поля данных.

Управление сессией

`int getInActiveTimeout()`	Возвращает таймер неактивности на экране внесения наличных в секундах.
`void initSCSOnScreen()`	Начать сессию по нажатию на экран. Функция используется на экране ожидания клиента.
`void setSCSButton (int pos, string value)`	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

`void loadPage (string window, string url)`	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

`WebIUS_Plugin wbCustom`	Модуль запроса истории операций клиента. Для использования модуля необходим сервер WebIUS.
`string receiveCardHolderOperationsInfo (string id)`	Запрос истории по id клиента, выполняющийся ассинхронно (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.
`void saveCardHolderOperationInfo (string id, string dataJson)`	Сохранение JSON по id. Параметры: dataJson – Json данные.

Другое

`string newGuid ()`	Сгенерировать GUID. Возвращает GUID в формате xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.

`int writeSessionInfo (string Id, string Text)`	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.
`int writeLogTrace (string method, string str)`	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)
```