Contents

1. WebIUS Browser

1.1 General Description

WebIUS Browser is an interface add-on to the **TellME7** software, which is a Windows Forms application that contains a standard Internet Explorer and Google Chrome. This integration platform incorporates the **TellME7** software functionality, along with the capability to expand the available set of functions by connecting external services.

WebIUS Browser is designed to address the following issues:

- Implementation of the user graphical interface.
- Implementation of the customer service business logic script.
- Hardware management through **TellME7** API.
- NDC Script Progress Management.
- Ability to interact with remote external banking services.
- Ability to extend the functionality by connecting various plugins. This allows the management of non-standard additional periphery from the customer service business logic script, as well as the integration of third-party software.

1.2 WebIUS Browser Architecture

WebIUS Browser consists of three logical parts (see fig. 1):

1. **User Interface** (html) – A user interface is a set of graphical elements displayed on the screen of a device.
   In **WebIUS Browser** , the user interface is implemented on screen form templates with dynamic content, thereby providing separation of the graphical representation from the business logic of the script. This approach allows you to add new script branches without turning to designers, replace the current set of screen forms without making changes to the business logic, and provides the ability to manage template packages on a case-by-case basis: using graphics for people with disabilities, using graphics for privileged customers, etc.
2. **Script** (JavaScript) is a description of the business logic of a script, in which it defines the necessary sequence of steps and actions required to perform by the client when performing an operation, as well as requests directed to the user, ATM and host. As the script is based on an event model and runs asynchronously, it is possible to use different event handlers from the user, **TellME7**, processing or other external systems.

   The primary benefit of this model is its ability to process events in a completely independent manner. In other words, the business logic of the script is executed simultaneously with the client's work and the handlers.
   The script implemented in **WebIUS** Browser is not rigidly bound to the **TellME7**

payment system, within the framework of which the work with payment means takes place. This approach offers greater flexibility in the creation of logic, due to the absence of a description of the work with NDC-script in the business logic script and its transfer to another class **(Support NDC).**

3. **Support NDC** - the link between the business logic script, NDC -script and **TellME7**.

   Support NDC is an intermediate module that contains a description of the work with NDC-script, in which all the used options, buffers and states for interaction with NDC-host are written once.

   All further changes are made to the **Script**. Thus, it is possible to abstract from the NDC protocol and carry out operations independently of its step-by-step architecture.

   The key benefit of separating the business logic script from the NDC script is that it allows the user experience to be maintained and ensures integration with other banking subsystems when changing the protocol used to interact with the bank's processing. This is achieved without needing to make any changes to the subsystems themselves.

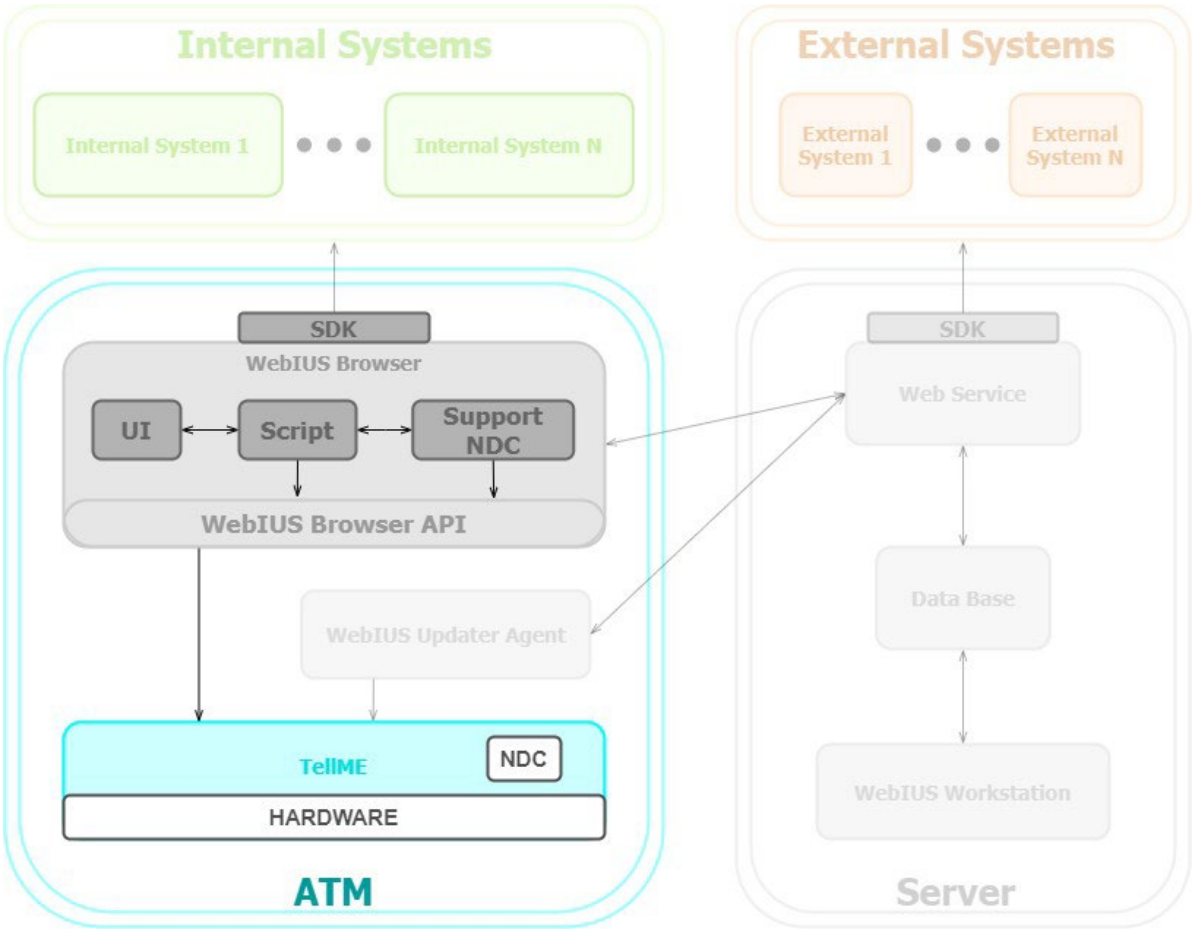

Рис. 1 - WebIUS Browser

2. screen.js

The central concept of a script is a session. A session is defined as a set of data associated with a specific ATM during a given service session. Each session can be in one of the states specified in the script. This state closely resembles what one would see on a screen in an ATM, and so **WebIUS Browser** employs the **screen.js** class to describe it.

For each state of the session, a set of variables and reactions to the events provided in the system can be described. Variables are used to store data about the session. Reactions to events are expressed through a sequence of JavaScript language operators. The system provides a fixed set of events expressed in methods of the **screen.js class**:

Properties and methods of the screen.js class

Name	Description
Working with event handlers	
`scr.addButtonCancel (WebIUS_callback)`	Adding a Cancel Epp keyboard handler. Parameters: WebIUS_callback is a function that can be executed when you press the Cancel button.
`scr.addButtonEnter (WebIUS_callback)`	Adding an Enter Epp keyboard handler. Parameters: WebIUS_callback is a function that can be executed when you press the Enter button.
`scr.addCall (event, callback)`	Adding an event handler. Parameters: event - event name, callback - function called when event triggers.
`scr.addNDCCaller (callback)`	Parameters: callback - the function that is called when the event is triggered.
`scr.addOnError (WebIUS_callback)`	Adding an event handler of type «error». Parameters: WebIUS_callback is a function that can be executed when an error occurs.
Working with screen elements	
`scr.buttonExists (buttonName)`	Checking if the button exists. Options: buttonName.

`scr.setButtonJson (buttonObj, buttonCallback)`	Filling the button element with information. It has the following parameters: buttonObj - button object, buttonCallback - function called when button is pressed. scr.setExtJson (extObj) Filling the additional information to be sent to the screen. Parameters: extObj is an object of additional information.
`scr.setExtJson (extObj)`	Filling the additional information to be sent to the screen. Parameters: extObj is an object of additional information
`scr.setImageJson (imageObj)`	Filling the image element with information. It has the following parameters: imageObj - image object.
`scr.setInputJson (inputObj, inputCallback)`	Filling the input element. It has the following parameters: inputObj - input object, inputCallback - the function called when entering data into the input field.аполнение элемента input.
`scr.setLabelJson (labelObj)`	Filling the label element. It has the following parameters: labelObj - label object.
`scr.setListJson (listObj, listCallback)`	Filling of the list element. It has the following parameters: llistObj - list object, listCallback - called function.
`scr.setModalMessageJson (modalObj, modalCall)`	Filling the modal screens with information.It has the following parameters: modalObj - modal window object, modalCall - called function.
`scr.deleteButton (buttonName)`	Deleting Json object button. Options: buttonName is the button name.
`scr.deleteInput (inputName)`	Deleting JSON object input. Parameters: inputName is the name of the input field.
`scr.deleteLabel (labelName)`	Deleting Json object label. Options: labelName.
`scr.deleteList (listName)`	Deleting of JSON object list. Options: listName is the name of the list.
`scr.deleteModalMessage ()`	Удаление Json объекта modalMessage.
`scr.deleteWait ()`	Deleting Json modalMessage object.
`scr.scrElementExists (elementName)`	Checking if the screen element exists. Parameters: elementName.

Working with screens

`scr.clearScr ()`	Clearing the screen.
`scr.name`	Screen name.
`scr.nextScreen (WebIUS_scrn)`	Changing screens. Parameters: WebIUS_scrn - screen name.

`scr.nextScreen (WebIUS_scrn, WebIUS_params)`	Changing screens. Parameters: WebIUS_scrn - screen name, WebIUs_params - additional parameters.
`scr.refreshScreen ()`	Refreshing screen information.
`scr.render (scrTypeName, scrAddress, scrCachClear)`	Filling the screen. It has the following parameters: scrTypeName - the type of screen, scrAddress - URL to call the page.

Other functions

`scr.setTimeOutJson (outObj, timeCall)`	Assigning timeouts. Parameters: outObj - time, timeCall - function called when the set time expires.азначение таймаутов.

When executing a script, the first step is to initialize all used script states. In an existing script, this action is performed by the **initScreens()** function.

2.1 Example of creating a screen with buttons

```
/** Creating a screen with buttons */
var screen_with_buttons = function (){

   /**
    * Assigning an event handler for the "Back" button
    *
    * @param {object} someArgs - array of elements containing
the name of the button,
    * as well as the result of actions performed on the page
    */
    var onbtn_return = function (someArgs) {
        /** Go to the previous screen */
        scr.nextScreen(previous_screen);
    };

   /**
    * Assigning an event handler for the "Continue" button
    *
    * @param {object} someArgs - array of elements containing
the name of the button,
    * as well as the result of actions performed on the page
    */
    var onbtn_continue = function (someArgs) {

        /** go to the next screen */
        scr.nextScreen(next_screen);
    };

    /**
      Creating a button element on the screen, assigning a
handler for pressing
      the button on the screen
    */
    scr.setButtonJson({name:"Btn4",
text:getLangText("btn_return"), ext:""}, onbtn_return);
    scr.setButtonJson({name:"Btn8",
text:getLangText("btn_continue")}, onbtn_continue);

    /**
      Creating a button element on screen, assigning EPP keyboard
я
      button press handler
    */
    scr.setButtonJson({name:"cancel",
text:getLangText("btn_return"), visible:false}, onbtn_return);
    scr.setButtonJson({name:"continue",
text:getLangText("btn_continue"), visible:false},
onbtn_continue);
};
```

```
/** Initializing screens */
function initScreens() {

   /**
    * Creating an object - default screen
    *
    * @param {object} main - screen object
    * @param {string} "" - screen name
    */
    scr = new Screen(main,"");

   /**
    * Creating an object - screen with buttons
    *
    * @param {object} screen_with_buttons - object of the
screen with buttons
    * @param {string} "screen_with_buttons" - screen name
    */
    screen_with_buttons = new Screen(screen_with_buttons,
"screen_with_buttons");
};
```

2.2 Example of creating a screen with Input field

```
/** Example of creating a screen with Input field */
var screen_with_Input = function () {

    /** Variable for storing service information (card number) */
    msession.service.cardNum = "";

        /**
         * Assignment of the event handler for the "Clear" button
         *
         * @param {object} someArgs - array of elements
containing the name
         * of the button, as well as the result of actions
performed on the page
         */
         var OnClear = function (someArgs){
             msession.service.cardNum = "";

             /**
               Updating the Input element on the screen, assigning
               the event handler
             */
             scr.setInputJson(inputObj:{name:"card_number",
text:session.service.cardNum, mask:"**** **** **** ****",
maxLength:16}, onInput);
               /** Refreshing screen information */
             scr.refreshScreen();
         };

        /**
         * Assigning an event handler for the Input field
         *
         * @param {object} someArgs - an array of
elements, containing the name of the
         * input field, the value of this field
         */
         var onInput = function(someArgs) {
             var pKey = someArgs[0], pValue = someArgs[1];
             session.service.cardNum = pValue;
         }

    /**
      Creating an input element on the screen, assigning the
      Input field event handler
    */
    scr.setInputJson(inputObj:{name:"card_number",
text:session.service.cardNum, mask:"**** **** **** ****",
maxLength:16}, onInput);
```

```
    /**
      Creating a button element on the screen, assigning a
handler for pressing the button on the screen
    */
    scr.setButtonJson({name:"Btn7",
text:getLangText("btn_clear"),ext:""}, onClear);

    /**
      Creating a button element on the screen, assigning the EPP
keyboard button press handler
    */
    scr.setButtonJson(({name:"clear",
text:getLangText('btn_clear'), visible:false}, onClear);
};

/** Initializing screens */
function initScreens() {

   /**
    * Creating an object - default screen
    *
    *@param {object} main - screen object
    *@param {string} "" - screen name
    */
    scr = new Screen(main,"");

   /**
    * Creating an object - screen with Input field
    *
    *@param {object} screen_with_Input - screen object with Input
    field
    *@param {string} "screen_with_Input" - screen name
    */
    screen_with_Input = new Screen(screen_with_Input,
"screen_with_Input");
};
```

3. support.htm

support.htm is the intermediary between NDC script, TellME and business logic script. It uses two main functions:

1. processFromTellME(param) – function to work with requests from TellME;
2. processFromScript(forceReq) – function to work with requests from **Script.js**.

To call **support.htm** from the business logic script you need to use the following function containing the URL address **support.htm**:

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

The loadPage() function has the following parameters:

1. `help` – help is the name of the window in which you want to load the URL address. The names of the windows are specified in the file **atm_web.exe.config**;
2. URL address consisting of:
   a. the address of the opened page;
   b. **Reload** parameter, in which the following values can be set:
      - 1 – load the page in the window;
      - 0 – if the page is already loaded - do not reload it, call the method **processSignalFromBrowser(param)** (parameters in URL address are passed to param).
   c. the source parameter indicating where the request came from:
      - `source=script` – из Script.js;
      - `source=tellme` – из TellME.

      Depending on the value of this parameter in the method **processSignalFromBrowser()** further actions are defined.

4. WebIUS Browser API functions

WebIUS Browser also has the following functions:

Name	Description
Interaction with hardware	
`int getAcceptNotesMaxCount ()`	Get the maximum number of bills to be accepted at one time.
`string getBCRData ()`	Get data from the barcode reader. Returns a string of scanner data or an error code string in the format «ERROR_ON_GET_DATA:err_code», where err_code is the numeric value of the hexadecimal error code. If err_code has the value «2A000048» (TIMEOUT), you can continue to call this method, but not more often than every 150 milliseconds or stop scanning by calling the BCRStopScan method. At other values err_code and upon successful reading, the scan operation will be automatically completed.
`void startBCRScan ()`	Start scanning the barcode via barcode reader.
`void stopBCRScan ()`	Finish scanning the barcode via barcode reader.
`string beepStart (string ParameterValue)`	Start beep. This function can be used when an event needs to be signaled with a beep. Parameters: ParameterValue - audio track id.
`string beepStop (string ParameterValue)`	Finish the beep. Parameters: ParameterValue - audio track id.
`string bimModuleManyNotes ()`	Get the type of the bill acceptor. Returns "0" - the type of bill acceptor is "single bills", "1" - the type of bill acceptor is "stack of bills".
`int getReceiptState ()`	Get the status of the receipt printer. Returns "1" - available, "2" - unavailable..

`string getNDCCurrency ()`	Get the currency of the accepted amount. Returns the numeric value of the currency code defined in NDC.
`string getAllAcceptedNotes (string currency)`	Get the denomination and the number of accepted bills. Parameters: currency - currency value. Returns a string with the data of accepted bills.
`int printReceipt (string str)`	Printing using the receipt printer. Returns the result of function execution: «0» - successfully sent to print, «1» - failed. Parameters: str - data to print..
`void setJournalPrint (string data)`	Print to journal printer. Options: data - data to print.

Interaction with NDC

`string getNDCAmount ()`	Get the current value of the sum defined in NDC.
`string getNDCBufferValue (string _BufferName)`	Get the value from the NDC buffer. Parameters: _BufferName.
`string getNDCScreen (string ScreenName)`	Get text from NDC screen with replaced non-printable characters. Options: ScreenName.
`string getNDCScreenOrig (string ScreenName)`	Get text from NDC screen. Parameters: ScreenName.
`int getTimer (int number)`	Get the NDC timer number, for example, the customer waiting timer number, waiting timer numbers of the pick up from the car, etc. Parameters: number is the timer number
`void setNDCBufferValue (string _BufferName, string _BufferValue)`	Save value in NDC buffer. Parameters: _BufferName; _BufferValue
`void setNDCButton (int pos, string value)`	Press the button in NDC. Options: pos - button position, value.
`string setNDCScreenText (string ScreenName, string ScreenText)`	Writing text to the NDC screen file. Returns "0" if the function completes successfully, "1" if ScreenName is empty, "2" if ScreenText is empty, and "3" if an error occurs. Parameters: ScreenName; ScreenText - text.

Working with files

`int deleteFile (string FileName)`	Delete file. Returns "0" if deleted successfully, "1" if file is missing, "2" if an error occurs.Parameters: FileName - the name of the file.
`string readFile (string FileName)`	П Read file. Returns either the contents of the file or an empty string. Parameters: FileName is the name of the file.
`int writeToFile (string FileName, string Text, bool AllowAppend)`	Write to file. Returns "0" - if it is successful, "1" - if an error occurs. Parameters: FileName - file name; Text - text; AllowAppend

Working with GUI

`string setScrResult (object SCRResultJSON, string target = "")`	Processing the result of GUI page operation. Returns the result in JSON format. Parameters: SCRResultJSON - object containing the result of display operation/state of page operation; target - preset value (window id).
`string getScrData (string target = "")`	Get the object based on which the display will be built in the GUI. Parameters: target - preset value (window id).
`SCRData scrData`	The module of work with GUI elements.
`void render (string scrTypeName, string scrCustomAddress = "", string cacheClear = "")`	Send for rendering. Parameters: scrTypeName - screen type (the type is set in the browser configuration file); scrCustomAddress - full path to the page to be rendered; cacheClear – the reserved service parameter.

Working with temporary memory

`void clearMemory ()`	Clear temporary memory.
`string getMemory (string name)`	Getting value from temporary memory. Parameters: name - name of the stored data field.
`void setMemory (string name, string value)`	Save the value into temporary memory. Parameters: name - name of the saved data field, value - value.

`void removeMemoryKey (string name)`	Remove the value from temporary memory. Parameters: name - name of the stored data field.

Session management

`int getInActiveTimeout()`	Returns the inactivity timer on the cash deposit screen in seconds.
`void initSCSOnScreen()`	Start a session touching the screen. The function is used on the client waiting screen.
`void setSCSButton (int pos, string value)`	Press the button in TellME. The function is used on the client waiting screen. Parameters: pos - FDK button position (from 1 to 8), value - value.

Working with pages

`void loadPage (string window, string url)`	Loading a page in a window. This function transfers information from form to form via url. Parameters: window; url - page url.

Working with the history of client's operations

`WebIUS_Plugin wbCustom`	Module for requesting the history of client operations.
`string receiveCardHolderOperations Info (string id)`	History request by customer id, executed asynchronously (if id is empty, hash + salt from card's PAN is used). Returns empty string if id is not empty or masked PAN card if id = "". Parameters: id - id of the client..
`void saveCardHolderOperationInfo (string id, string dataJson)`	Saving JSON by id. Parameters: dataJson - Json data.

Другое

`string newGuid ()`	Generate GUID. Returns the GUID in the format xxxxxxxxx-xxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxx.

`int writeSessionInfo (string Id, string Text)`	Writing to a separate sessions folder in a file named <Id>.session. Returns "0" if the function completes successfully, "1" if an error occurs. Parameters: Id - session id; Text - text.
`int writeLogTrace (string method, string str)`	Writing to a separate log with the .trace extension. Returns "0" - in case of successful completion of the function, "1" - in case of error. Parameters: method; str - string..

Global functions are used to call them.

For example, to get the text from the NDC screen you should use the following string:`window.external.exchange.getNDCScreen(ScreenName)`