Version 10

Terminal Integration Guide

SecurOS POS Terminal Integration Guide (POSINTEG - EN, build 122 on 24.04.2020).

Printed in US.

Contents

1 Preface

This section contains general information about the document, the means of its design and use, as well as how to get additional technical support for the product.

1.1 Scope

This manual describes the process of POS terminal integration in SecurOS system for their teamwork with the SecurOS POS Module.

1.2 Target Audience

This manual is intended for SecurOS system integrators and administrators who are experienced users of the Microsoft Windows operating system, with expertise in CCTV technology, computer hardware, configuring a local area network and TCP/IP networking.

1.3 Using This Manual

This document is organized as a book, so the user can print it or use the electronic version. In the latter case one can use Adobe Reader's Bookmark feature as well as the cross-reference hyperlinks to navigate through content. In several topics this manual refers to other SecurOS manuals (**SecurOS Quick User Guide** etc.), which can be found as separate files on the SecurOS installation CD or downloaded from our website (**www.issivs.com**).

1.4 Getting Technical Support

If you have any questions after reading this manual, please address them to your system administrator or supervisor.

For any further information you can contact the Intelligent Security Systems Technical Support Team:

Note. To get a quick response to a request use the Technical Support Portal, which www address is listed below.

- **in USA:**

  phone: +1 732 855 1111 (Monday to Friday, 8:30am - 6pm EST);

  e-mail: **support@issivs.com**

  www: **https://support.issivs.com**

- **in Russia:**

  phone: +7 (495) 645 21 21 (Monday to Thursday, 9am - 6pm MST; Friday 9am - 5pm MST);

  www: **https://help.iss.ru**

Note. See the **https://help.iss.ru/user/manual** for the Portal User Guide.

- **in Brazil:**
  phone: +55 11 2262 2894 (Monday to Friday, 9am - 6pm BRT);
  e-mail: **suporte@issivs.com**
  www: **https://support.issivs.com**
- **in Mexico:**
  phone: +52 1 551330 0181 (Monday to Friday, 9am - 6pm CDT);
  e-mail: **supportlatam@issivs.com**
  www: **https://support.issivs.com**
- **in Colombia/Ecuador:**
  phone: +57 300 442 2808 (Monday to Friday, 9am - 6pm COT/ECT);
  e-mail: **supportlatam@issivs.com**
  www: **https://support.issivs.com**
- **in Chile:**
  phone: +56 9 6573 2993 (Monday to Friday, 9am - 6pm CLT);
  e-mail: **supportlatam@issivs.com**
  www: **https://support.issivs.com**
- **in Ukraine:**
  phone: +380 (44) 299 08 10 (Monday to Friday, 9am - 6pm EET);
  e-mail: **supportua@issivs.com**
  www: **https://support.issivs.com**
- **in Peru/Bolivia:**
  phone: +51 997 111 678 (Monday to Friday, 9am - 6pm PET/BOT);
  e-mail: **supportlatam@issivs.com**
  www: **https://support.issivs.com**
- **in Argentina:**
  phone: +54 91152528779 (Monday to Friday, 9am - 6pm ART);
  e-mail: **supportlatam@issivs.com**
  www: **https://support.issivs.com**

To solve problems faster, we recommend preparing the service information described in the **Technical Support Information** Section before addressing the Technical Support Team.

1.5 Design Convention

For representation of various terms and titles the following fonts and formatting tools are used in this document.

Font	Description
bold type	Used in writing workstation names, utilities or screens, windows and dialog boxes as well as the names of their elements (GUI elements).
italic type	Used to mark out the SecurOS objects.
bold italic type	Used to mark out the elements of homogeneous lists.
`monospace`	Used to mark out macro text and programming code, file names and their paths. Also it is used to specify the necessary options, to mark out values specified by the user from the keyboard (manually).
green	Used to mark out the cross-references within the document and links to the external available ones.

1.6 Design Elements

Warning! Serves to alert the user to information which is necessary for the correct perception of the text set out below. Typically, this information has a warning character.

Note. Note text in topic body.

Additional Information

Used to display additional information. These type of elements contain, for example, the description of options for executing a task or reference to additional literature.

2 General Description

Integration of a POS terminal with SecurOS SecurOS POS Module gives the user additional opportunities on management and monitoring of the security system, and as a sharing opportunity of various POS terminals with other subsystems (e. g., video and audio monitoring, see **SecurOS Administration Guide**).

To integrate a POS terminal with the SecurOS system do the following:

1. Install SecurOS Integrated video management platform system software.
2. Install SecurOS POS Module software. Create and configure necessary SecurOS system objects and the Module objects. See **SecurOS POS User Guide** for detailed information about system configuration and POS terminal hardware installation.
3. Create a cash voucher syntax analyzer (parser) templates (see **Integration Procedure**).
4. Launch SecurOS.
5. Configure *POS Terminal* object. In the **Receipt parsing profile** field of the object settings window select created parser file name (see Figure 1).

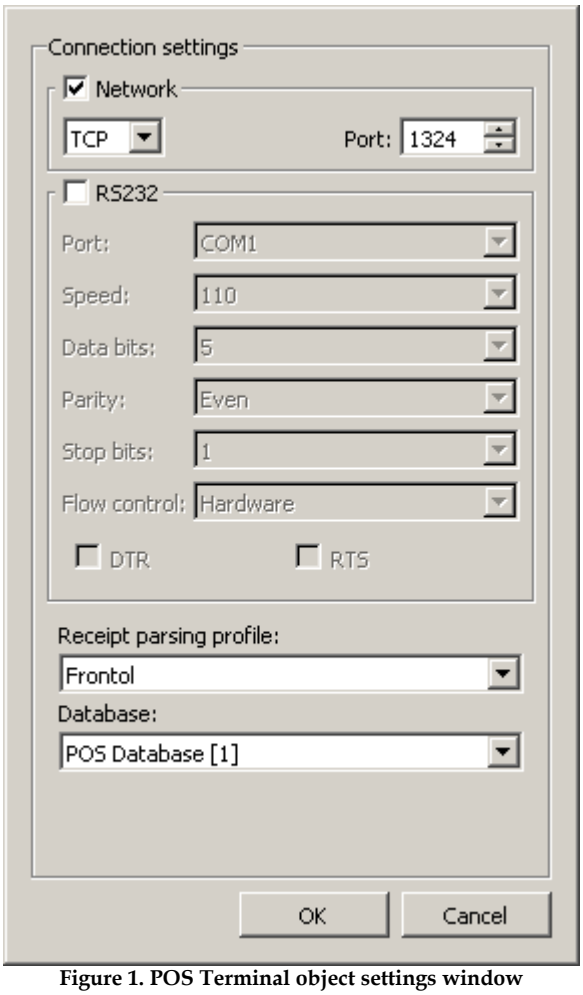

Figure 1. POS Terminal object settings window

3 Integration Procedure

Terminal integration mechanism is based on running JavaScript (JS parser) script writing with use of JavaScript integration library.

Cash voucher integration procedure contains the following steps:

1. **Creating Parser XML Description File**.
2. **JS Parser Implementation**.

3.1 Creating Parser XML Description File

There are parsers as XML descriptors and JS executive files in the `\POSParsers` subfolder of the SecurOS folder.Each parser is used for the specific POS terminal type. JS files are used for operational parser implementation. The XML descriptor file is the following:

Listing 1. XML parser description example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
   <comment>Test profile</comment>
   <entry key="profile_name">POSParsers/test.js</entry>
   <entry key="charsetname">cp866</entry>
   <entry key="classname">ru.iss.pos.server.parser.js.
   ReceiptProcessorJSImpl</entry>
</properties>
```

Note. Possible degrees of freedom for the XML parser modification are described in DTD document, see it on the http://java.sun.com/dtd/properties.dtd web site.

To create parser XML description file do the following:

1. Copy one of XML descriptors to a new file and rename it according with a new parser.
2. Modify XML descriptor file in the following way:
   - Type in a path to the JS file (relative to SecurOS folder) for `profile_name` attribute.
   - Type in a code page of data received from POS terminal through COM or UDP port for `charsetname` attribute. Code page name must correspond to Java standard. See http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html to get supported code pages list. For example, when data from a POS terminal are in DOS code page then `charsetname` attribute value must be set as cp866 (see above Listing 1). If `charsetname` attribute is not defined then no recoding operations will be made. See also http://java.sun.com/j2se/1.4.2/docs/api/java/io/InputStreamReader.html for documentation on data flow processing procedures in Java language.

Warning! `charsetname` attribute is not used for data transferred through UDP port from terminals.

3.2 JS Parser Implementation

Note. A parser script must be written due to JavaScript v 1.5 rules. For more information see http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference.

JS script of cash vouchers syntax analyzer (parser) is used for data analysis and extraction received from a cash voucher on a terminal.

To create parser JS script do the following:

1. Copy `dummy.js` file from `\POSParsers` subfolder of `SecurOS` folder to the folder defined in `profile_name` attribute of XML descriptor file parser.
2. Rename copied to the `<profile_name>.js` one.
3. Edit the JS parser file to be in accordance to the integrated terminal cash voucher format.

On writing the JS parser remember the following:

1. Script is called every time on a new cash voucher line is received. Script makes syntax analysis of the line and extract data for displaying on POS Inspector Module titles and using in Module events parameters.
2. The appointed JS script has to contain the `createComponents` factory function creating the parser main object. This object must have the `dispatcher` method that will be called on receiving a new line data (see the example below).

Listing 2. Parser script example. In this case, this script results in one line shift without printing any cash voucher information.

```
function createComponents() {
   var self = new Object();
   self.dispatcher = function(data) {
   listener.pushFilteredString(data); //data line parser script
   }
   return self;
}
```

3. Before a parser execution the callback object is set as `listener` predefined global variable. Methods of `listener` object (ыуу **Methods of listener Object**) are used for cash voucher parsing and notification of the *POS Terminal* object about recognized cash voucher/ sales/events for subsequent displaying in titles.

3.2.1 Methods of listener Object

This section describes the `listener` object methods available from the parser script.

3.2.1.1 pushReceiptBegin

Syntax: `pushReceiptBegin (code)`
Description: Indicate a new cash voucher beginning
Parameters:

`code`	Current cash voucher code from the protocol, or `null` when cash voucher code is not accessible from the protocol. Argument type: String
Returned value:	
	New cash voucher ID

3.2.1.2 pushCashierName

Syntax: `pushCashierName (cashierName)`	
Description: Set a current cashier name	
Parameters:	
`cashierName`	Current cashier name. Argument type: String

3.2.1.3 pushNewSale

Syntax: `pushNewSale (productName)`	
Description: Add a new record about commodity sale into a current cash voucher. Returns the sale record ID	
Parameters:	
`productName`	Current commodity name. Argument type: String

3.2.1.4 pushSaleAmount

Syntax: `pushSaleAmount (amount)`	
Description: Set a product price	
Parameters:	
`amount`	Commodity price. Argument type: Float

3.2.1.5 pushSaleDiscount

Syntax: `pushSaleDiscount (percents)`	
Description: Set a commodity discount (in percents)	
Parameters:	
`percents`	Commodity discount value (in percentage). Argument type: Float

3.2.1.6 pushSaleProductsCount

Syntax: `pushSaleProductsCount (count)`	
Description: Set a commodity quantity (or weight)	
Parameters:	
`count`	Commodity quantity (or weight). Argument type: float

3.2.1.7 pushSaleArticle

Syntax: `pushSaleArticle (article)`	
Description: Set a commodity article	
Parameters:	
`article`	Commodity article. Argument type: String

3.2.1.8 pushSaleEnd

Syntax: `pushSaleEnd ()`
Description: Indicate the end of the current sale
Parameters: none

3.2.1.9 pushReceiptDiscount

Syntax: `pushReceiptDiscount (percents)`	
Description: Set a common discount for overall sale (in percents). **Warning!** This operation cancels separate commodities discounts.	
Parameters:	
`percents`	Overall discount for the whole cash voucher (in percentage). Argument type: Float

3.2.1.10 pushReceiptEnd

Syntax: `pushReceiptEnd (totalAmount)`
Description: Indicate the end of the current cash voucher
Parameters:

`totalAmount`	Total amount of the current cash voucher taken from the protocol, or `null`. In the latter case (`null`), total amount value is set by taking into account all sales within the current cash voucher (as in `pushReceiptEnd ()`). Argument type: Float

Syntax: `pushReceiptEnd ()`
Description: Indicate the end of the current cash voucher. Its total amount value is set by taking into account all sales within the current cash voucher
Parameters: none

3.2.1.11 pushFilteredString

Syntax: `pushFilteredString (line)`
Description: Add a new line to titles. Then push down cash voucher line displayed in titles on this new line
Parameters:

`line`	Cash voucher line symbol in titles. Argument type: String

3.2.1.12 pushEvent

Syntax: `pushEvent (eventDescription)`
Description: Send events into SecurOS system on behalf of the Module *POS Terminal* object (see event list in **SecurOS POS User Guide**)
Parameters:

`eventDescription`	Event name. Argument type: String
Returned value:	
	Event identifier

Syntax: `pushEvent (eventDescription, cashierName, amount)`
Description: Send events into SecurOS system on behalf of the Module *POS Terminal* object (see event list in **SecurOS POS User Guide**)
Parameters:

`eventDescription`	Event name. Argument type: String. `receiptCode` – cash voucher number for this event, or `null` if the number is undefined. Argument type: String
`cashierName`	Cashier name for this event, or `null` if the name is undefined. Argument type: String

`amount`	Total amount of cash voucher (sale) for this event, or `null` if the amount is undefined. Argument type: Float
Returned value:	
	Event identifier

3.2.1.13 setEventReceiptCode

Syntax: `setEventReceiptCode (eventId, receiptCode)`
Description: Set a cash voucher code for the event with specified `eventId`
Parameters:

`eventId`	Event identifier. Argument type: long
`receiptCode`	Cash voucher code. Argument type: String.

3.2.1.14 setReceiptCode

Syntax: `setReceiptCode (receiptId, receiptCode, updateSalesCode)`
Description: Set a cash voucher code with the given `receiptId`. If `updateSalesCode == true` then the code is set for all sales of the current cash voucher, otherwise the code is set for the cash voucher only
Parameters:

`receiptId`	Cash voucher ID. Argument type: long
`receiptCode`	Cash voucher code. Argument type: String
`updateSalesCode`	Logical condition (see method description). Argument type: boolean

4 Integration Example

There is an example of JS parser for a POS Terminal software integration below.

Listing 3. Initial bill

```
* "Avalon Company" *
============================
Cashier: Ivanova Olga
============================
+ Rice 6 kg 100.0
+ Meat 1 kg 50.0
============================
Discount: 10 \% 15.0
TOTAL: 135.0
* Have a good day! *
```

Steps:

1. Create a <profile_name>.js JS parser and add the following code fragments.
2. Define bill beginning and ending.

Listing 4. Bill beginning definition

```
self.testReceiptBegin = function(data) {
   if (/\* "Avalon Company" \*/.test(data)) {
      listener.pushReceiptBegin();
      listener.pushFilteredString("----------------------------");
      listener.pushFilteredString(" Beginning ");
      listener.pushFilteredString("----------------------------");
      return true;
   }
   return false;
}
```

The second line of the code above compares string with the start string (bill beginning) by `test` method. See lines `3 - 7`: this code describes actions on bill beginning detection: send *POS Terminal* object a notification of bill beginning (line `3`), titles displaying (lines `4 - 6`). The way of bill ending detection is the same.

Listing 5. Bill ending definition

```
self.testReceiptEnd = function(data) {
   * Have a good day! \*/.test(data)) {
      listener.pushReceiptEnd();
      listener.pushFilteredString("----------------------------");
      listener.pushFilteredString(" Finished! ");
      listener.pushFilteredString("----------------------------");
      return true;
   }
   return false;
```

```
}
```

3. Define the cashier (cashier name).

Listing 6. Cashier definition

```
self.testCashierName = function(data) {
   var matches = /Cashier: (\w+\s*\w*)/.exec(data);
   if (matches != null && matches.length >= 2) {
   listener.pushCashierName(matches[1]);
   return true;
   }
   return false;
}
```

Regular expressions are used to single-out cashier first and second name (see line `2`, `exec` method). Line `4` describes a current cashier registration, if a line coincides with parser template. For the detailed description of the JavaScript regular expressions see http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference.

4. Define single sale.

Listing 7. Sales definition

```
self.testSale = function(data) {
   var matches = /\+\s(\w+)\s+(\d+)\s*\w*\s*(\d+\.?\d*)/.exec(data);
   if (matches != null && matches.length >= 3) {
      var productName = matches[1];
      var productCount = parseFloat(matches[2]);
      var amount = parseFloat(matches[3]);
      listener.pushNewSale(productName);
      listener.pushSaleProductsCount(productCount);
      listener.pushSaleAmount(amount);
      listener.pushSaleEnd();
      listener.pushFilteredString(data);
      return true;
   }
   return false;
}
```

Regular expressions are used to single-out sale parameters (see line `2` in the code above). Lines `7`, `8` and `9` define sales parameters: product name, product count and sale amount.

5. Add a commodity quantity (or weight) definition.

Listing 8. Total amount definition

```
self.testTotalSale = function(data) {
   var matches = /TOTAL: \s+(\w*)/.exec(data);
   if (matches != null) {
   var totalCount = parseFloat(matches[1]);
   listener.pushSaleProductsCount(totalCount);
   return true;
```

```
    }
    return false;
}
```

Regular expressions are used to calculate total amount (see line 2 in the code above). Line 5 describes a total amount registration, if a line coincides with parser template. The way of discount definition (in percents) is the same.

Listing 9. Discount definition

```
self.testDiscount = function(data) {
    var matches = /Discount: \s+(\w+)\s+\%?\s+(\d+\.?\d*)/.exec(data);
    if (matches != null && matches.length >= 3) {
        var saleDiscount = parseFloat(matches[1]);
        listener.pushReceiptDiscount(saleDiscount);
        return true;
    }
    return false;
}
```

6. Add calling of all the defined functions into the script:

Listing 10. Script text. Line processing consists of line shift, printing line symbol on titles and applying all created function to received cash voucher line for data extraction.

```
function createComponents() {
    var self = new Object();
    ... //list of parser functions
    self.dispatcher = function(data) { //dispatcher method for
        out.println(data); //processing received line
        listener.pushFilteredString(data);
        this.testReceiptBegin(data) ||
        this.testReceiptEnd(data) ||
        this.testCashierName(data) ||
        this.testSale(data) ||
        this.testTotalSale(data) ||
        this.testDiscount(data);
    }
    return self;
}
```

Below is the final program code.

Listing 11. Final code

```
function createComponents() {
    var self = new Object();
    self.testReceiptBegin = function(data) {
        if (/\* "Avalon Company" \*/.test(data)) {
            listener.pushReceiptBegin();
            listener.pushFilteredString("-----------------------------");
            listener.pushFilteredString(" Beginning ");
            listener.pushFilteredString("-----------------------------");
            return true;
        }
```

```
    return false;
}

self.testReceiptEnd = function(data) {
    if (/\* Have a good day! \*/.test(data)) {
        listener.pushReceiptEnd();
        listener.pushFilteredString("------------------------------");
        listener.pushFilteredString(" Finished! ");
        listener.pushFilteredString("------------------------------");
        return true;
    }
    return false;
}

self.testSale = function(data) {
    var matches = /\+\s(\w+)\s+(\d+)\s*\w*\s*(\d+\.?\d*)/.exec(data);
    if (matches != null && matches.length >= 3) {
        var productName = matches[1];
        var productCount = parseFloat(matches[2]);
        var amount = parseFloat(matches[3]);
        listener.pushNewSale(productName);
        listener.pushSaleProductsCount(productCount);
        listener.pushSaleAmount(amount);
        listener.pushSaleEnd();
        listener.pushFilteredString(data);
        return true;
    }
    return false;
}

self.testTotalSale = function(data) {
    var matches = /TOTAL: \s+(\w*)/.exec(data);
    if (matches != null) {
        var totalCount = parseFloat(matches[1]);
        listener.pushSaleProductsCount(totalCount);
        return true;
    }
    return false;
}

self.testDiscount = function(data) {
    var matches = /Discount: \s+(\w+)\s+\%?\s+(\d+\.?\d*)/.exec(data);
    if (matches != null && matches.length >= 3) {
        var saleDiscount = parseFloat(matches[1]);
        listener.pushReceiptDiscount(saleDiscount);
        return true;
    }
    return false;
}

self.testCashierName = function(data) {
    var matches = /Cashier: (\w+\s*\w*)/.exec(data);
    if (matches != null && matches.length >= 2) {
        listener.pushCashierName(matches[1]);
        return true;
    }
    return false;
}

self.dispatcher = function(data) {
```

```
            out.println(data);
            listener.pushFilteredString(data);
            this.testReceiptBegin(data) ||
            this.testReceiptEnd(data) ||
            this.testCashierName(data) ||
            this.testSale(data) ||
            this.testTotalSale(data) ||
            this.testDiscount(data);
        }
        return self;
}
```

5 Appendixes

5.1 Appendix A. Technical Support Information

Current section contains service information that is necessary on addressing to Intelligent Security Systems Technical Support.

Note. Collected data have to be send to the Intelligent Security Systems Technical Support Team (see **Getting Technical Support**).

To ensure quick technical support, prepare the following technical information:

Warning! Data in items marked by "*" are necessary to report.

1. (*) User (customer) name to address to.
2. (*) Organization name.
3. (*) User (or organization) contacts: phone, e-mail.
4. Name of a personal Intelligent Security Systems manager (on Intelligent Security Systems authorized partner case). Otherwise, give the following data:
   - Company where the hardware and software components were purchased.
   - Actions proposed to solve the problems announced by a partner from whom the product was purchased.
5. (*) Problem description.
6. (*) Actions results in the problem.
7. List of changes which result to the problem in case of applying after some changes in system settings/configuration.
8. System and diagnostic information on computer and SecurOS system configuration obtained from the **SystemInfo** utility (see **SecurOS Administration Guide** for detailed information about utility).

   If it is impossible to run the utility provide the following information:
   - (*) Guardant keys identifier and Dallas code;

   Note. Equipment Dallas code can be found by the **ISS Hardware Report** utility (see **SecurOS Administration Guide** for detailed information about utility).

   - (*) name and version of the installed Intelligent Security Systems company software.
   - total number of video servers and monitoring (operator) workstations in the system;
   - operating system (name and service pack version).
9. Another useful information, if possible. For example:

- computer equipment configuration.
- central processors load.
- main and virtual memory used volumes.
- network load.
- network and network neighborhood configuration.

Index

P

S

T