POS SDK

POS Intellect 5.3 (english)

Last update 10/24/2022

Table of Contents

1 POS SDK. Introduction 5
1.1 Purpose of the Guide 5
1.2 Structure of the Guide 5
1.3 General information about POS SDK 5
2 XML-protocol 7
2.1 Description of XML protocol 7
2.1.1 General information about XML protocol 7
2.1.2 Control package 8
2.1.3 Format of xml_titles.xml settings file 8
2.1.4 Interaction with POS server 12
2.2 Configuring the POS-Intellect system when working with XML data communication protocol 12
2.3 Configuring xml_titles.txt settings file 14
2.4 Description of XML protocol packages for interaction with POS-Intellect system 19
2.4.1 Fields and their values 19
2.4.2 Format of data exchange packages with AxxonSoft video monitoring system 21
3 Using a .prl parser 29
3.1 General information about creating .prl parsers in the POS-Intellect software 29
3.2 Regular expressions 29
3.2.1 General information about regular expressions 29
3.2.2 Quantifiers 29
3.2.3 Symbol sorting 30
3.2.4 Special symbols 31
3.2.5 Select condition 31
3.3 Description of the parser_designer.exe utility interface 32
3.4 Creating a parser 33
3.5 Checking a created template 34
3.6 Configuring parser in the POS-Intellect software 36
4 Libraries for working with POS-Intellect software package 38
4.1 COM. Library for working with TCP/IP connection 38

4.2 ActiveX component for working with TCP/IP connection 39
4.3 DLL. Library for working with TCP/IP connection 39
5 Interaction with 1S program 41
6 Appendix 1. The tpcgen.exe utility for sending text data to the POS-Intellect system 42
6.1 General information about the tcpgen.exe utility 42
6.2 Working with the tcpgen.exe utility 42

Download POS SDK

1 POS SDK. Introduction

On this page:

- Purpose of the Guide
- Structure of the Guide
- General information about POS SDK

1.1 Purpose of the Guide

POS SDK utilities and libraries package document is a property of the AxxonSoft company. This document is a reference manual for developers of POS software.

This document contains information about POS SDK utilities and libraries package designed for configuring operation of the *POS-Intellect* software with POS terminals of different manufacturers.

1.2 Structure of the Guide

This document includes the following:

1. Description of the XML protocol;
2. Information about parsers usage;
3. Description of external libraries for working with the *POS-Intellect* software package.

1.3 General information about POS SDK

POS SDK utilities and libraries package is available for loading on the *page*.

Utilities package in POS SDK allows creating of parsers to get information from POS receipts and input this information to a receipts database.

Database is loaded in one of the following ways depending on communication protocol of receipt data from POS-terminal to POS-server:

1. Using the xml_titles.txt parser
2. Using parser with .prl resolution

> **Note.**
>
> Detailed information about parsers configuring in the POS-Intellect software package is given in the POS-Intellect. Administrator's Guide document.

Libraries package is designed for sending data to the POS-Intellect from external programs.

Attention!

Libraries of the POS SDK are applied for example. AxxonSoft company is not responsible for their operation.

2 XML-protocol

2.1 Description of XML protocol

2.1.1 General information about XML protocol

XML protocol allows adding connectivity with *POS-Intellect* operation control system to POS software developers if data of POS operations is sending as XML packages. This format allows transferring of any number of significant parameters to a system and organizing their displaying on a video image and saving in transaction database.

XML package has the **<TransactionBlock>** basic tag. Separate xml package is sent to the Intellect for each POS operation. Each package should have the **<TransactionBlock>** start tag and the **</TransactionBlock>** end tag . So each POS operation has its presentation as XML package. Example of sent package is follows.

```
<TransactionBlock>
   <FunctionNumber>1003</FunctionNumber>
   <FunctionName>User registration</FunctionName>
   <UserNumber>7850</UserNumber>
   <UserName>John Smith</UserName>
   <TransactionTime>10:53:59</TransactionTime>
</TransactionBlock>
```

All parameters are presented in a text view. It's possible to use point or comma to separate fractional part of a number. Converting to the required type is performed automatically.

POS program should send data to TCP, UDP or RS232 port (is specified in settings). Separate port is in use for each POS.

It's required to foresee an automatic connection repairing in case of disconnecting when the TCP protocol is in use.

Required tags of xml-package:

1. **FunctionNumber** – number of function.
2. **TransactionTimestamp** (conditionally required tag) – event time specified in **yyyy-mm-ddThh:nn:ss.fff** format. If the **TransactionTimestamp** tag is missing in the package, than time of package receipt is recording to the database except of event time.
   The following symbols are in use:
   yyyy - year as four-digit number.
   mm- month in range from 01 to 12.
   dd - day of the month in range from 01 to 31.
   hh - hour in 24-hours format from 00 to 23.
   nn - minutes in range from 00 to 59.
   ss - seconds in range from 00 to 59.
   fff - second thousandths (millisecond) in value of date and time.

Separate elements of XML package by line separators (consecutive 0xD, 0xA symbols). It's more convenient to do when system debugging (when using the xml_test.exe utility, see the Configuring xml_titles.txt settings file section).

Total number of parameters is not limited. Parameter names are without spaces. Received data are displaying in the screen as captions and recording to a database for further analysis.

The **Intellect\Modules\Pos\xml_titles.txt** file is intended for settings (see the *Format of xml_titles.xml settings file* section).

All received data are recording to **POS_LOG_MASTER** and **POS_LOG_DETAIL** tables.

Default name of receipts database is pos, MS SQL Server 2008 is in use.

2.1.2 Control package

If there is no any pos activity, but connection with *POS-Intellect* is established, it's required to send control package confirming TCP/IP connection. Recommended interval is 30 seconds. Control package is sending with the **FunctionNumber=77777** parameter.

```
<TransactionBlock>
<FunctionNumber>77777</FunctionNumber>
<DateTime>2014-04-16T17:37:18.516</DateTime>
</TransactionBlock>
```

Time of data reading in seconds is specified on the site of POS server. System considers that connection with software is lost if none of XML packages was not received during the specified time. In this case connection is closing and establishing again.

2.1.3 Format of xml_titles.xml settings file

Sample view of xml_titles.txt file is shown in the figure.

```
[2]
New receipt №<CheckNumber>,
Cashier : <UserName> , № <UserNumber>

[3]
     <ItemName>(<ItemNumber>)  x <ItemQuantity>  <ItemPrice> rub

[9]
Print receipt №<ChackNumber> , <TransactionDate> <TransactionTime>

[TRANSFORM]
UserName=cashier_name
UserNumber=cashier_number
ItemName=item_name
ItemQuantity=item_count
ItemPrice=item_total
5:ItemPrice=item_amount
6:ItemPrice=item_price
ItemNumber=item_code
CheckNumber=check_number

[PROLOG]
2
1
[BODY]
3
8
[EPILOG]
9

[FUNCTIONNAME]
2=Open receipt
3=Sale of goods
9=Print receipt
```

The numbers in square brackets represent the numbers of functions that come in the **FunctionNumber** field of the package. Directly below the number the template is specified in brackets — a description of how the specified function will be displayed on the video image. The variables in angle brackets "**<>**" will be replaced with the corresponding values. If there is at least one such parameter in the string, and there is the data incoming from the POS terminal with this parameter missing, then the entire string will be ignored and will not be displayed on the screen.

It's possible to specify single functions (e.g. **[1]** or **[2]**) and combination of several functions if they have the same template (e.g. **[1,2,5,10]**).

Configuring of file sections and fields is follows:

1. Parameters of fields formatting are specified in the **[PARAM_TEXT_FORMAT]** section.

   ```
   [PARAM_TEXT_FORMAT]
   ItemName=25,left,upper
   ```

   In example the **ItemName** parameter is displayed. Its maximal length is 25 symbols (spaces will be added automatically if length is less), left alignment, all symbols are displayed in capitals.

Available alignment values: **left, right, center**.
Available values of letters size: **upper, lower**.

2. Numbers of sections corresponding to new receipt entry are specified in the **[PROLOG]** section. Function, number of which is specified in this field, automatically increases (per one) internal counter of receipts (transactions). Parameters corresponding to these functions should be column names of the **POS_LOG_MASTER** table.
3. Function numbers corresponding to receipt body are specified in the **[BODY]** section. Parameters corresponding to these functions should be column names of the **POS_LOG_DETAIL** table.
4. Function numbers corresponding to receipt end are specified in the **[EPILOG]** section. Parameters corresponding to these functions should be column names of the **POS_LOG_MASTER** table.
5. Function numbers which should not display on the screen in text view. But data of these functions are saving to POS base – if they are specified in **[PROLOG],[BODY],[EPILOG]** sections (see. **PROLOG,BODY,EPILOG**).
6. Parameters and output string are specified in the **[PROLOG_EVENT_PARAM]** section. The specified string is displayed if this parameter is in the received xml-package. This section is applied for systems without concrete determination of receipt entry.

```
[PROLOG_EVENT_PARAM]
check_number, NEW RECEIPT!
```

7. Matches of sending parameters and column names of pos database tables depending on function number are specified in the **[TRANSFORM]** section.

```
[TRANSFORM]
UserName=cashier_name
UserNumber=cashier_number
ItemName=item_name
ItemQuantity=item_count
ItemPrice=item_total
5:ItemPrice=item_amount
6:ItemPrice=item_price
ItemNumber=item_code
CheckNumber=check_number
```

In the above example, the **item_total** column will match to the received **ItemPrice** parameter if function number is not 5 or 6. If function number is 5 – the **item_amount** column, if function number if 6 – **item_price** column.
It's possible to send the following string:

```
<item_total>12,34</item_total>
```

instead of the string:

```
<ItemPrice>12,34</ItemPrice>
```

In this case, description in the **[TRANSFORM]** field is not required.

8. Matches of receiving function numbers (**FunctionNumber**) and their names are specified in the **[FUNCTIONNAME]** section. It's using for matching function numbers and their names in reports forming in accordance of operator search requests.
9. When parameter specified in the **[IGNORE_ZERO]** section equals zero, the corresponding POS event will be ignored. For example, package is received from the POS, and the following text is specified in the xml_titles.txt file, than section with the specified event won't be added to the POS_LOG_DETAIL table.

```
<TransactionBlock>
   <FunctionNumber>1003</FunctionNumber>
   <TerminalID>001</TerminalID>
   <TransactionDate>26.05.10</TransactionDate>
   <TransactionTime>10:53:59</TransactionTime>
   <CashierNumber>005</CashierNumber>
   <CashierName>John Smith</CashierName>
   <CheckNumber>0063</CheckNumber>
   <Article>  4902210219202</Article>
   <ProductName>Milk 0.5l</ProductName>
   <ProductQty>1</ProductQty>
   <ProductPrice>50.00</ProductPrice>
   <RowSum>50.00</RowSum>
 </TransactionBlock>
```

Package with zero IDCAmount parameter

```
[IGNORE_ZERO]
IDCAmount

[BODY]
12
```

Fragment of the xml_titles.txt file specifying on requirement to ignore the package

A system defines the way of adding parameter values to the database basing on functions specified in **PROLOG,BODY,EPILOG** sections. *Parameters are adding only when function numbers in which the specified parameters are sending are added to one of three section (**PROLOG,BODY,EPILOG).***

Values are loading to the database as follows:

1. Record is adding to the **[POS_LOG_MASTER]** table using the **INSERT** SQL operation when receiving function specified in the **[PROLOG]** section. Program automatically increases a **[check_id]** counter and stores it.
2. Record is adding to the **[POS_LOG_DETAIL]** table using the **INSERT** SQL operation where the stored value of the **[check_id]** table is specified in the **[id]** field, when receiving function specified in the **[BODY]** section.
3. Record is updating in the **[POS_LOG_MASTER]** table using the **UPDATE** SQL operation for the string where the **[check_id]** value corresponds to the stored value, when receiving function specified in the **[EPILOG]** section.

   So the **[check_id]** field is using within the program and should not be in use as name of sending parameter. Sequence order of sections can be optional. Single-line comments can be written using // symbols.

   For example, the package is received when adding a good to receipt:

```
<TransactionBlock>
       <FunctionNumber>1</FunctionNumber>
<FunctionName>Add goods</FunctionName>
<ItemName>Bag</ItemName>
<ItemCount>12</ItemCount>
<ItemPrice>12,34</ItemPrice>
<TransactionTimestamp>2016-01-04T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

   and the following description is corresponding to this package:

```
[1]
<FunctionName> :
Name: <ItemName>
Quantity: <ItemCount>
Price: <ItemPrice>
 Cashier: <UserName>
 ID: <UserNumber>

[BODY]
1

[TRANSFORM]
ItemName=item_name
ItemCount=item_count
ItemPrice=item_price
```

In this case the following captions will be displayed:

```
Adding of goods:
Name: Bag
Quantity: 12
Price: 12,34
```

Record containing values of **item_name, item_count, item_total** fields and the ***FunctionNumber*** value will be added to the **POS_LOG_DETAIL** table of the **pos** database.

2.1.4 Interaction with POS server

Integration with POS server which can receive events from all pos terminals (or filling stations) and resend them to the video surveillance system using one logic connection is allowable.

The separate camera (cameras) should be added to the each POS (filling station).

In this case the**StationNumber** tag is required in the package.

```
<StationNumber>ID</ StationNumber >
```

Where ID - unique identifier of POS (filling station).

Example of interaction with POS server:

One POS is installed in the filling station and data are sending to the video surveillance system using this POS. 3 filling stations are connected to this POS. The separate camera is added to the each filling station.

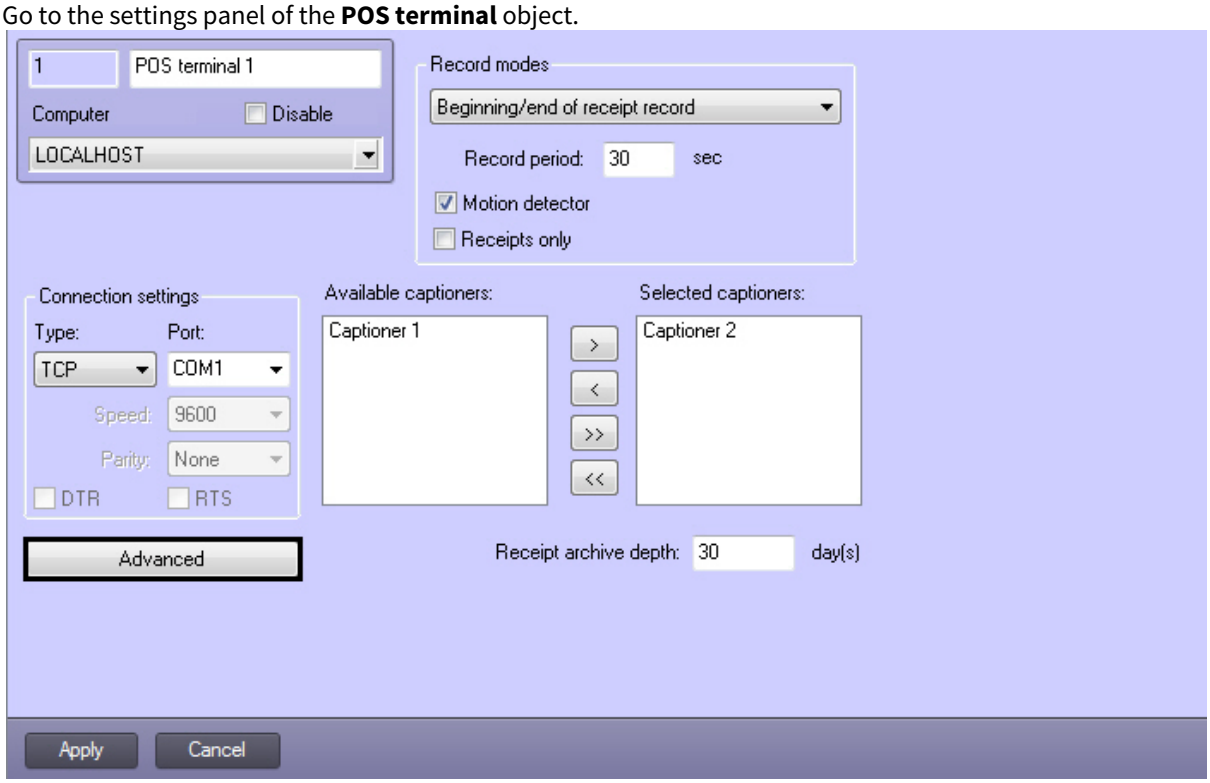
To realize video monitoring in the POS-Intellect software it's required to create 3 POS-terminal objects in the objects tree, assign cameras to the each **POS-terminal** object and assign **POS-terminal** to the corresponding POS.

Then the POS will send data to the mixforward.exe utility embedded to the POS-Intellect software which will analyze data and resend them to created system objects.

2.2 Configuring the POS-Intellect system when working with XML data communication protocol

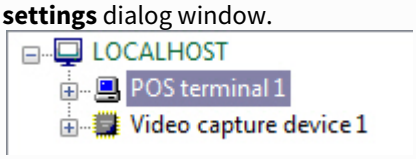
To configure the *POS-Intellect* system when working with XML data communication protocol, do the following:

1. Go to the settings panel of the **POS terminal** object.



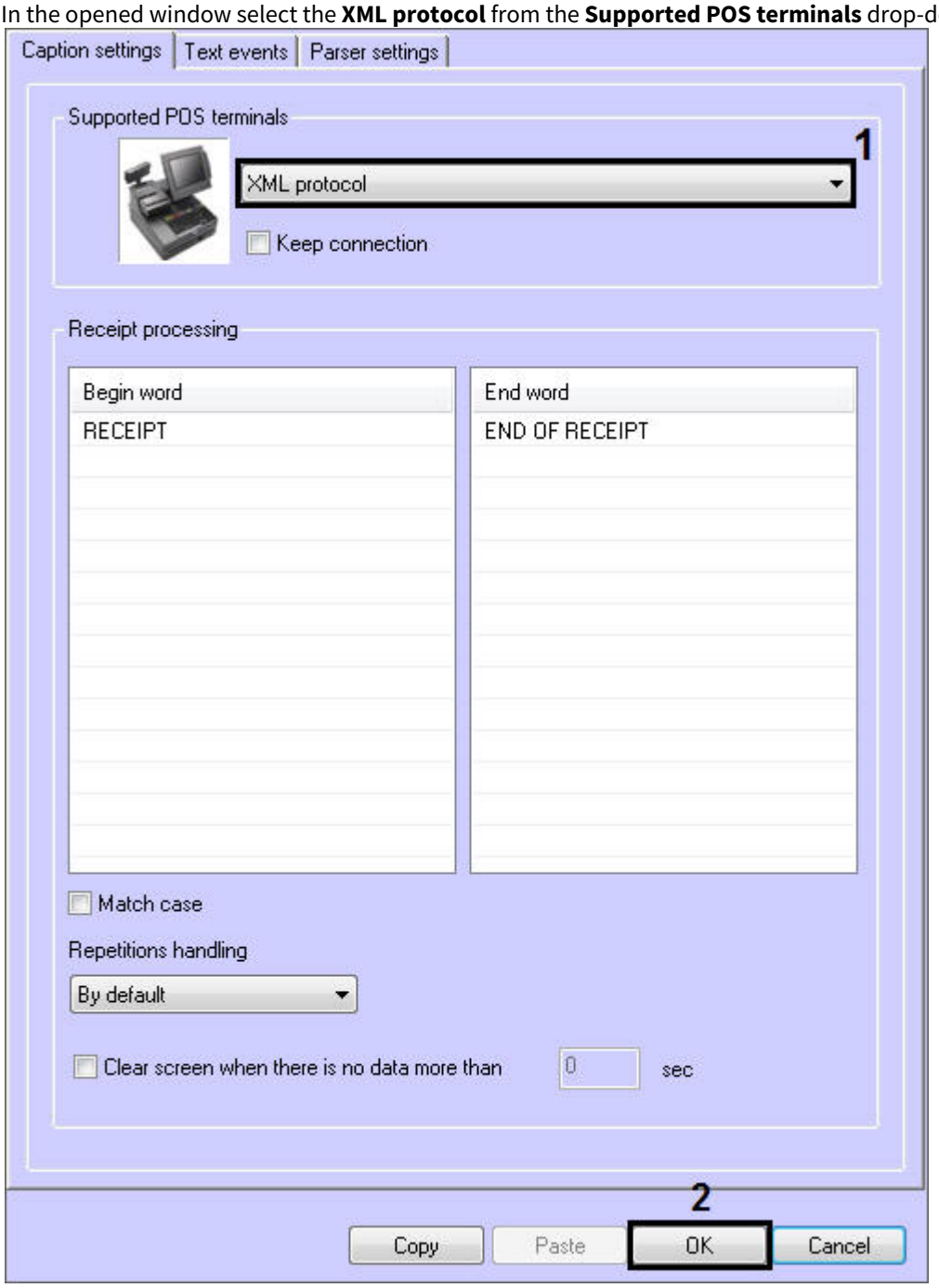
Note.

The **POS-terminal** object is based on the **Computer** object on the **Hardware** tab of the **System settings** dialog window.



2. Click the **Advanced** button.

3. In the opened window select the **XML protocol** from the **Supported POS terminals** drop-down list (**1**).



4. Click **OK** (**2**).

Configuring the *POS-Intellect* system is completed.

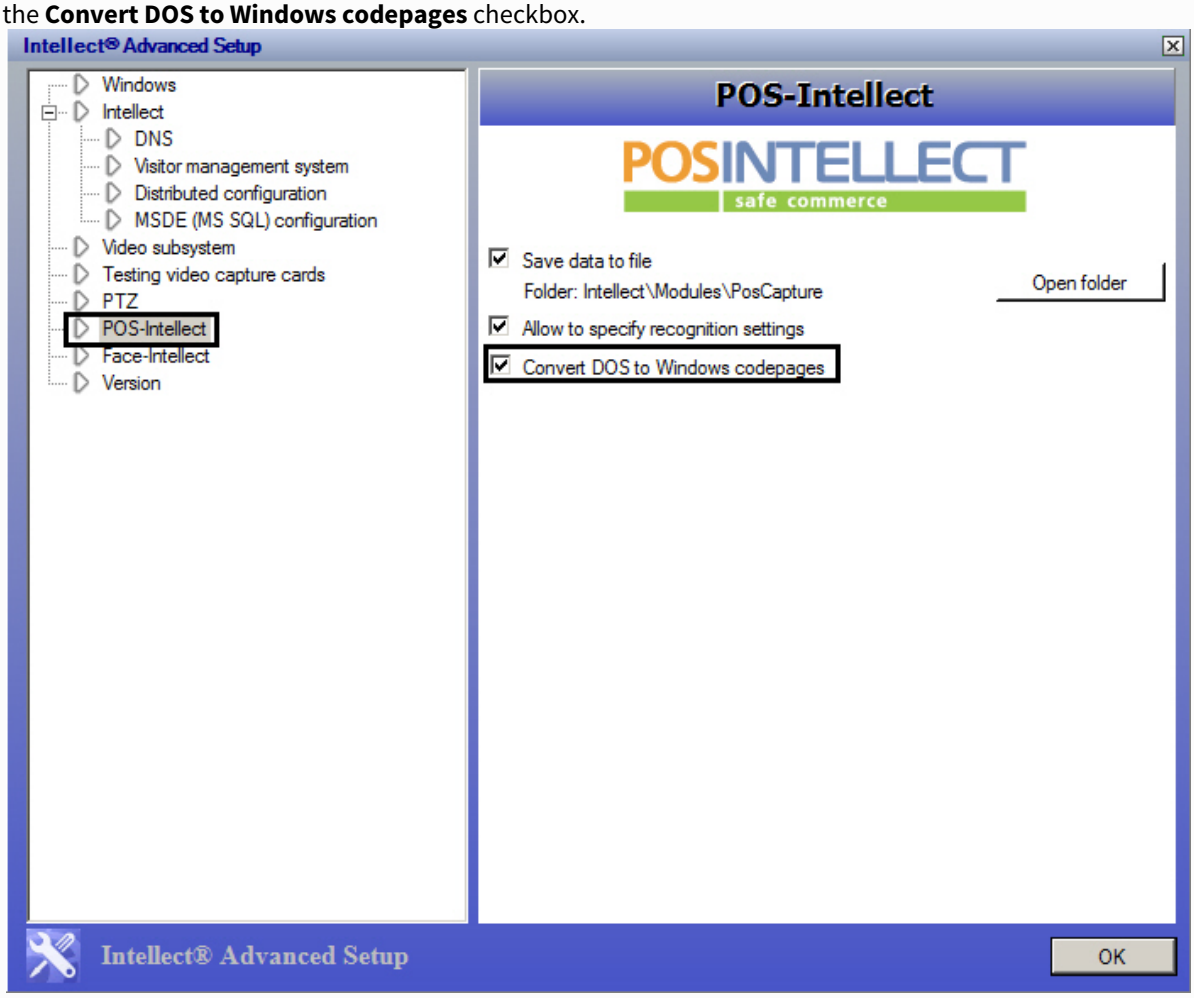
2.3 Configuring xml_titles.txt settings file

To simplify configuring of the *xml_titles.txt* file use the specialized utility. The *xml_test.exe* utility is designed to debug captions above the video image when working with XML protocol of the POS-terminal module. The

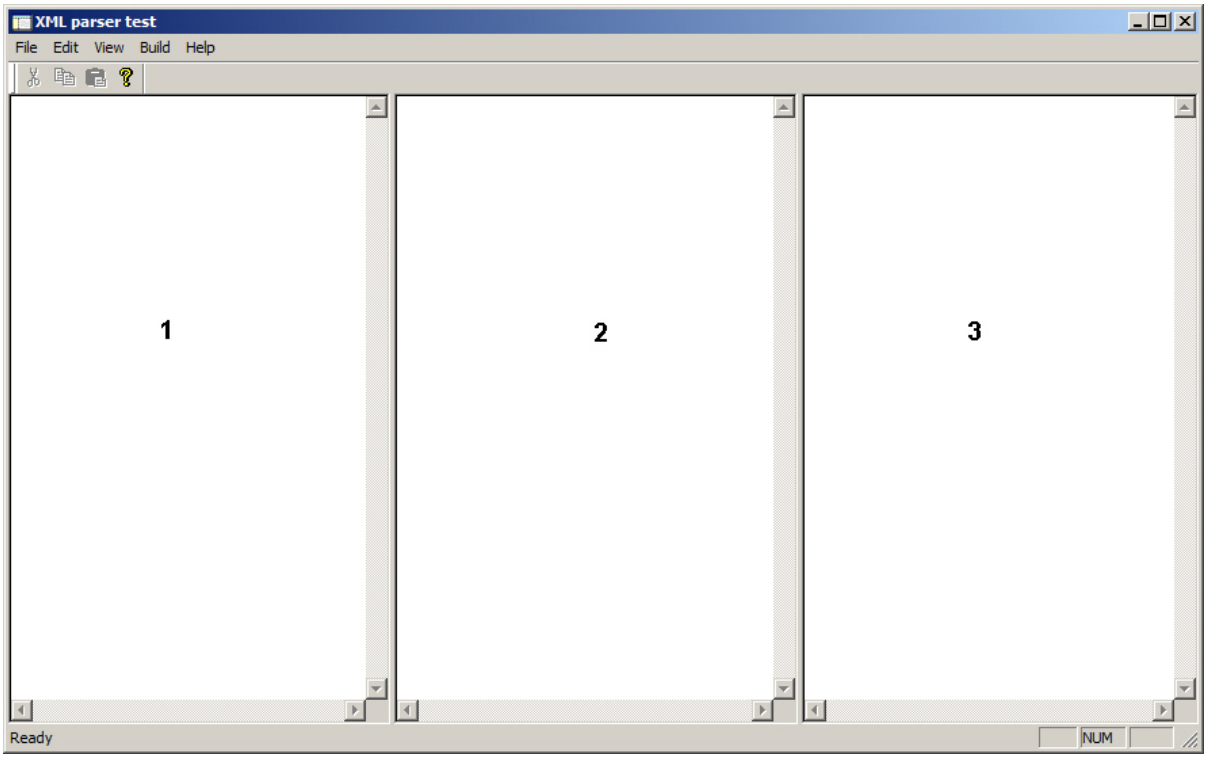
xml_test.exe utility is a part of the POS SDK utilities and libraries package. To launch the utility download the POS SDK archive, extract files and launch the POS_SDK\XML_Protocol\Xml_test.exe executable file.

Note.

If sending data are in DOS coding, it's required to launch the advanced settings utility tweaki.exe and set the **Convert DOS to Windows codepages** checkbox.

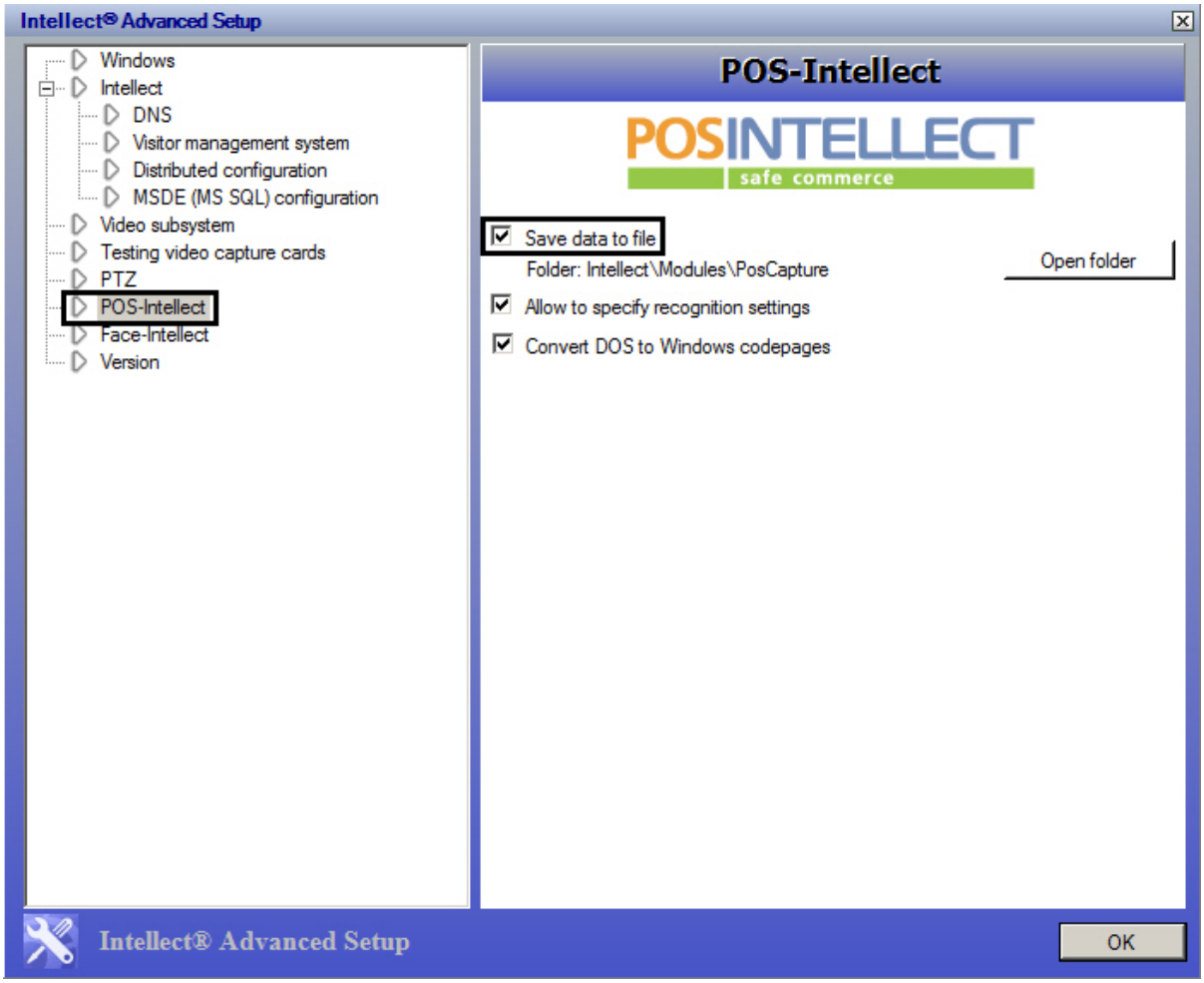


The *xml_text.exe* utility contains three windows:



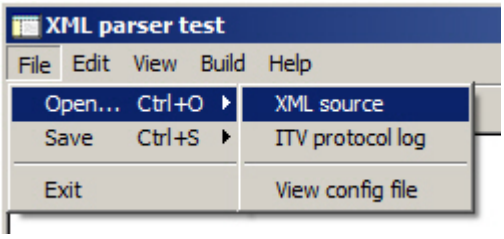
XML-log received from the POS is loaded to the first window. Logs are saving in the <Intellect software installation directory>\Modules\PosCapture folder. Name of log file is pos_N.log, where N - number of POS-terminal corresponding to POS from which log was received.

Ensure that the **Save data to file** checkbox is set in the **POS-Intellect** section of the tweaki.exe utility if there are no logs in this file.

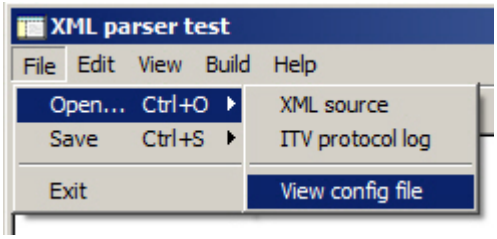


To open log select **File -> Open -> XML source** menu.

> **Note.**
>
> The **ITV protocol log** menu item is not in use.

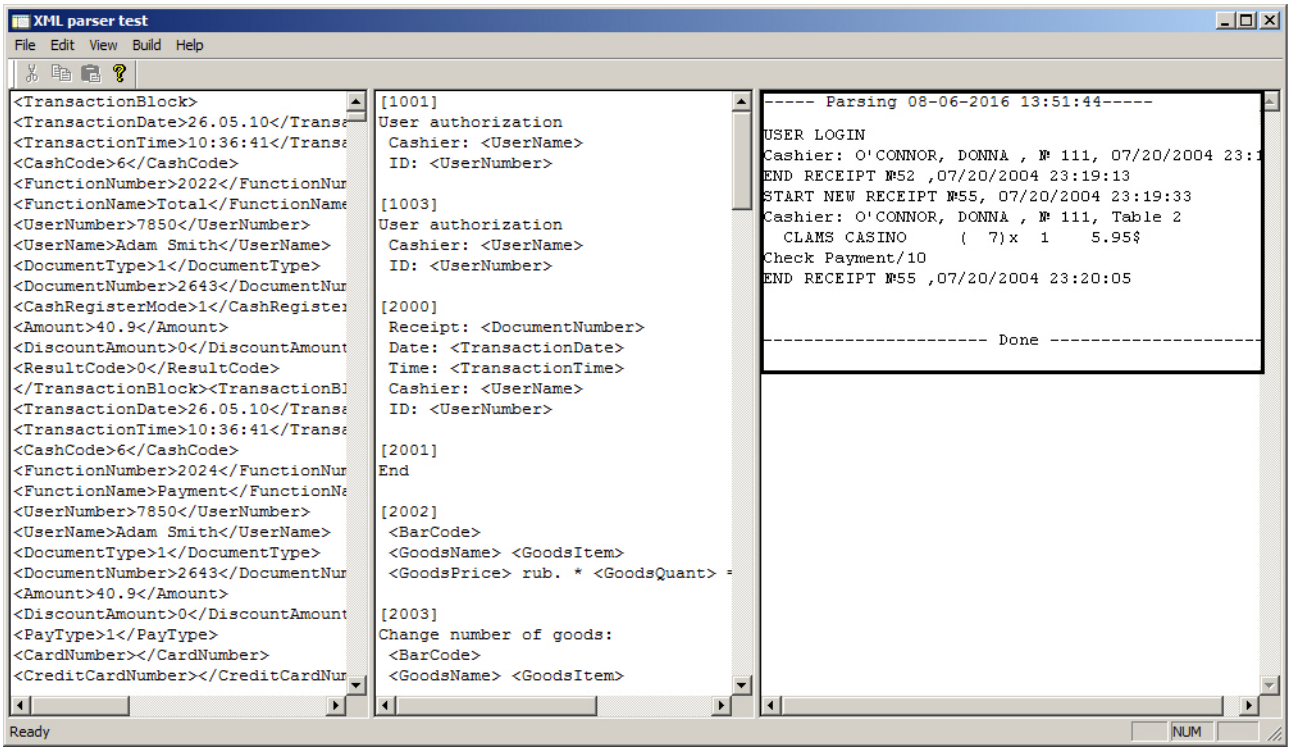


The xml_titles.txt parser file replying for displaying is loaded to the second window. To open xml_titles.txt file select the **File -> Open ->View config file** menu.



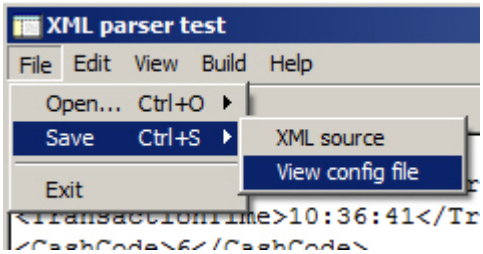
It's possible to load existed file or form it singly.

Displayed captions above the video image while *POS-Intellect* working with connected devices will display in the third window when clicking the F7 key.



So the main operation is editing a content of the second window.

To save the xml_titles.txt file select the **File -> Save -> View config file** menu.



Then it's required to copy the file to the **<** Intellect software installation directory **>\Modules\Pos\xml_titles.txt**

2.4 Description of XML protocol packages for interaction with POS-Intellect system

2.4.1 Fields and their values

List of fields and their values is given in the table. This list is recommended. Functions can be added or removed.

Function parameters, their names and number can be optionally changed except of the **FunctionNumber** parameter (it's required parameter).

Changing of functions or parameters is requiring to configure the xml_titles.txt file.

Field	**Value**
FunctionNumber	Number of function
FunctionName	Name of function
cash_number	Number of POS
card_number	Number of card
date	Date of document creation
discount_name	Name of discount
enter_type	Way of payment card input: 1- using keyboard; 2- by magnetic reader; 3- using pin-pad; 4 – by barcode using scanner
receipt_number	Number of receipt
item_id	Item identifier
item_name	Item name
item_barcode	Item barcode
item_price	Price of item
item_quantity	Quantity of goods
item_discount	Item discount

item_amount	Anount of item in receipt
receipt_amount	Current sum of receipt
item_pos	Number of goods item
receipt_discount	Total discount
receipt_discount_number	Discount amount (%)
payment_name	Name of payment instrument
payment_amount	Total receipt sum
payment_amount_with_change	Received sum from customer
payment_change	Sum of change
result	Result of receipt completion: 1 - cancelled receipt; 2 – delayed receipt
result_str	Comment for receipt completion
shift_number	Number of shift
TransactionTimestamp	Event time specified in the **yyyy-mm-ddThh:nn:ss.fff** format
receipt_type	Type of receipt
restore	Typr of receipt restoring
type_str	Comment
type	Type of operation
user_name	Name of cashier
user_id	Cashier ID

Note.

The **type** parameter can take the following values (if they are available) in operations (7, 8, 11, 19) connecting with receipts:
<type>
0 - receipt for sale;
1 – return;
2 – return by receipt;
5 – restoring of delayed receipt;
7 – receipt for inventory;
8 – receipt for non-tax purposes;
9 – return by receipt for non-tax purposes.
In operation 15:
<type>
100 – insertion;
101 – subtraction.
Format of date-time is following: YYYY-MM-DD HH:MM:SS.MS, where YYYY – year, MM – month, DD – day, HH – hour, MM – minutes, SS – seconds, MS – milliseconds. For example, 2006-09-01 13:13:34.045.

2.4.2 Format of data exchange packages with AxxonSoft video monitoring system

1. **Cashier login**

```
<TransactionBlock>
                <FunctionNumber>1</FunctionNumber>
<FunctionName>Cashier login</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

2. **Cashier logout**

```
<TransactionBlock>
                <FunctionNumber>2</FunctionNumber>
<FunctionName>Cashier logout</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

3. **Add a position**

```
<TransactionBlock>
                <FunctionNumber>3</FunctionNumber>
<FunctionName>Add a position</FunctionName>
                <receipt_number>10</receipt_number>
                <item_id>006946</item_id>
                <item_name>MILK 0.5L</item_name>
```

```
                <item_barcode></item_barcode>
                <item_price>10.00</item_price>
                <item_quantity>1.000</item_quantity>
                <item_amount>10.00</item_amount>
                <receipt_amount>0.00</receipt_amount>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

4. **Change quantity**

```
<TransactionBlock>
                <FunctionNumber>4</FunctionNumber>
<FunctionName>Change quantity</FunctionName>
                <receipt_number>10</receipt_number>
                <item_id>013611</item_id>
                <item_name>MILK 0.5L</item_name>
                <item_barcode></item_barcode>
                <item_price>15.10</item_price>
                <item_quantity>5.000</item_quantity>
                <item_amount>75.50</item_amount>
                <receipt_amount>29.00</receipt_amount>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

5. **Cancellation and reversal of position**

```
<TransactionBlock>
                <FunctionNumber>5</FunctionNumber>
<FunctionName>Cancellation and reversal of position</FunctionName>
                <receipt_number>10</receipt_number>
                <item_id>013611</item_id>
                <item_name>MILK 0.5L</item_name>
                <item_barcode></item_barcode>
<item_price>15.10</item_price>
                <item_quantity>5.000</item_quantity>
                <item_amount>75.50</item_amount>
                <receipt_amount>29.00</receipt_amount>
                <date>27.09.2006 15:42:19</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

6. **Change price of current position**

```
<TransactionBlock>
                <FunctionNumber>6</FunctionNumber>
<FunctionName>Change price of current position</FunctionName>
                <receipt_number>10</receipt_number>
                <item_id>005393</item_id>
```

```
                <item_name>MILK 0.5L</item_name>
                <item_barcode></item_barcode>
                <item_price>29.00</item_price>
                <item_quantity>1.000</item_quantity>
                <item_amount>29.00</item_amount>
                <receipt_amount>0.00</receipt_amount>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

7. **Delayed or cancelled receipt**

```
<TransactionBlock>
                <FunctionNumber>7</FunctionNumber>
<FunctionName>Delayed or cancelled receipt</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <date>2006-09-07 15:20:45.051</date>
                <receipt_number>283</receipt_number>
                < receipt_type>0</ receipt_type>
                <result>1</result>
                <result_str>RECEIPT CANCELLATION</result_str>
                <receipt_amount>240.00</receipt_amount>
                <receipt_discount>0.00</receipt_discount>
<receipt_discount_number>0</receipt_discount_number>
                <cash_number>9999999</cash_number>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

8. **Closing a receipt**

```
<TransactionBlock>
                <FunctionNumber>8</FunctionNumber>
<FunctionName>Closing a receipt</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <date>2006-09-07 15:20:45.051</date>
                <receipt_number>280</receipt_number>
                <receipt_type>0</receipt_type>
                <result>0</result>
                <receipt_discount>0.00</receipt_discount>
<receipt_discount_number>0</receipt_discount_number>
                <receipt_amount>721.90</receipt_amount>
                <cash_number>9999999</cash_number>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

9. **Total amount**

```
<TransactionBlock>
```

```
                <FunctionNumber>9</FunctionNumber>
<FunctionName>Total amount</FunctionName>
                <receipt_number>10</receipt_number>
                <receipt_amount>29.00</receipt_amount>
                <receipt_discount>0.00</receipt_discount>
<receipt_discount_number>0</receipt_discount_number>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

10. **Payment on receipt**

```
<TransactionBlock>
                <FunctionNumber>10</FunctionNumber>
<FunctionName>Payment on receipt</FunctionName>
                <receipt_number>10</receipt_number>
                <payment_name>Cash</payment_name>
                <payment_amount>721.90</payment_amount>
                <payment_amount_with_change>800.00</payment_amount_with_change>
                <payment_change>78.10</payment_change>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

11. **Opening a receipt**

```
<TransactionBlock>
                <FunctionNumber>11</FunctionNumber>
<FunctionName>Opening a receipt</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <date>2006-09-07 15:20:45.051</date>
                <receipt_number>285</receipt_number>
                <receipt_type>5</receipt_type>
                <type_str>RESTORING</type_str>
<restore>RESTORING AFTER FAILURE</restore>
                <cash_number>9999999</cash_number>
                <shift_number>5</shift_number>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

> **Note.**
>
> Value of the receipt_type parameter shows that opening a receipt was after failure. Detailed information about values of this parameter see in the Fields and their values section.

12. **Closing a shift**

```
<TransactionBlock>
                <FunctionNumber>12</FunctionNumber>
```

```
<FunctionName>Closing a shift</FunctionName>
                <receipt_number>286</receipt_number>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

13. **Discount**

```
<TransactionBlock>
                <FunctionNumber>13</FunctionNumber>
<FunctionName>Discount</FunctionName>
                <receipt_number>10</receipt_number>
                <discount_name>Hand_disk</discount_name>
                <receipt_amount>0.00</receipt_amount>
                <receipt_discount>0.00</receipt_discount>
<receipt_discount_number>0</receipt_discount_number>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

14. **Open a till**

```
<TransactionBlock>
                <FunctionNumber>14</FunctionNumber>
<FunctionName>Open a till</FunctionName>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

15. **Monetary operation**

```
<TransactionBlock>
                <FunctionNumber>15</FunctionNumber>
<FunctionName>Monetary operation</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <date>2006-09-07 15:20:45.051</date>
                <receipt_number>288</receipt_number>
                <type>101</type>
                <type_str>SUBTRACTION</type_str>
                <payment_name>Cash</payment_name>
                <payment_amount>16050.00</payment_amount>
                <shift_number>6</shift_number>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

> **Note.**
>
> Value of the type parameter shows the case of subtraction money. It's possible to **insert** money, see the Fields and their values section.

16. **Card for payment**

```
<TransactionBlock>
                <FunctionNumber>16</FunctionNumber>
<FunctionName>Card for payment</FunctionName>
                <receipt_number>10</receipt_number>
                <enter_type>1</enter_type>
                <card_number>4895</card_number>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

17. **Discount card**

```
<TransactionBlock>
                <FunctionNumber>17</FunctionNumber>
<FunctionName>Discount card</FunctionName>
                <receipt_number>10</receipt_number>
                <enter_type>1</enter_type>
                <card_number>2367</card_number>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

18. **Manual discount**

```
<TransactionBlock>
                <FunctionNumber>18</FunctionNumber>
<FunctionName>Manual discount</FunctionName>
                <receipt_number>10</receipt_number>
                <type_str>DISCOUNT FOR ITEM</type_str>
                <item_pos>1</item_pos>
                <item_discount>-15</item_discount>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

19. **Print a receipt copy**

```
<TransactionBlock>
                <FunctionNumber>19</FunctionNumber>
<FunctionName>Print a receipt copy</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <date>2006-09-07 15:20:45.051</date>
                <receipt_number>291</receipt_number>
                <receipt_type>0</receipt_type>
                <type_str>SALE</type_str>
                <cash_number>9999999</cash_number>
```

```
                <shift_number>6</shift_number>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

20. **Opening a till without transaction**

```
<TransactionBlock>
                <FunctionNumber>20</FunctionNumber>
<FunctionName>Opening a till without transaction</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

21. **Z-report**

```
<TransactionBlock>
                <FunctionNumber>21</FunctionNumber>
<FunctionName>Z - report</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <shift_number>5</shift_number>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

22. **X-report**

```
<TransactionBlock>
                <FunctionNumber>22</FunctionNumber>
<FunctionName>X - report</FunctionName>
                <user_name>Smith</user_name>
                <user_id>19</user_id>
                <shift_number>6</shift_number>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

23. **Control package**

```
<TransactionBlock>
<FunctionNumber>77777</FunctionNumber>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

24. **Manual input**

```
<TransactionBlock>
<FunctionNumber>23</FunctionNumber>
<FunctionName>Manual input</FunctionName>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

25. **Attempt of forbidden operation**

```
<TransactionBlock>
<FunctionNumber>25</FunctionNumber>
<FunctionName>Attempt of forbidden operation</FunctionName>
<type_str>Название операции</type_str>
                <date>2006-09-07 15:20:45.051</date>
<TransactionTimestamp>2009-01-01T04:04:33.004</TransactionTimestamp>
</TransactionBlock>
```

3 Using a .prl parser

3.1 General information about creating .prl parsers in the POS-Intellect software

Parsers with .prl file extension define the rules of loading receipts database if data from the POS-terminal are sending to the POS-server by protocol differing from XML. Regular expressions are used when parser creating.

To create parser, do the following:

1. Create templates using parser_designer.exe and ParserTest.exe utilities.
2. Add created templates to the parser using the *POS-Intellect* software.

> **Note.**
>
> There are a lot of ways of recording templates to the same fragment. So examples given in this section are not the only possible.

3.2 Regular expressions

> **Note.**
>
> The following description of regular expressions based on text from the http://phpclub.ru web-site.

3.2.1 General information about regular expressions

To analyze POS operations it's required to review data flow received from pos terminal and select the required data. Regular expressions are used for this purpose.

Regular expressions - template languages.

Template is processed symbol-by-symbol. For example, to find a "d" letter in the "stadium" word, it's required to analyze all letters of the "stadium" word and estimate them with required letter.

Template – is a unique indicator what to search in the string. It's possible to search digits, letters, invisible symbols (space, tabbing, etc.).

To search the "d" or "m" letter in the "stadium" word unless the first coincidence, use the method of sorting: analyze each letter and compare it with the required symbol. i.e. with "d" and "m". Such set of search conditions is called a symbol sorting and it's writing as follows: [dm] - it says that d or m letters are searching for.

To specify that any letter is searching for, it's required to list them all [abc...xys] or specify the interval [a-z].

It's allowable for Russian letters [а-я], digits [0-9] and for capital letters [A-Z], i.e. to get the symbol sorting with all Latin letters specify the [a-zA-Z] in a template.

This symbol sorting describes only one symbol. To describe all symbols in the string use quantifiers.

3.2.2 Quantifiers

For example, there are two strings:

abcd12345efg
fghi56789qwe

Condition: Find parts in strings which contain four latin letters and then 5 digits.

How to describe a symbol coincident with some Latin letter (a symbol sorting is in user for it: [a-z] (don't pay attention to capitals)), and a symbol of coincident with a digit: [0-9] was described in the previous section.

It's required to find strings with four letters at first and 5 digits straight after them.

It's allowable to write: [a-z][a-z][a-z][a-z][0-9][0-9][0-9][0-9][0-9]

Such form of search condition will work. One symbol of search condition is described by symbol sorting. Such construction is too massive. It's recommended to use quantifiers in such cases. Quantifier determines number of symbols in a search condition. So the search condition will be simplified be quantifiers: [a-z]{4}[0-9]{5}

Number of symbols described in symbol sorting and going one after another in a string is specified in curved brackets.

So 5 digits are going after 4 Latin letters. Each symbol sorting described only one symbol, number of similar sequential symbols are described by quantifiers.

This specifying of symbol numbers in a search condition is not unique. Quantifiers can be different: [a-z]{1,3} means that from one to three Latin letters can go one after another. [a-z]{2,} means that minimum two Latin letters can go one after another.

Quantifier in curved brackets is not unique way to set number of symbols going one after another. [a-z]* means that any number of Latin letters can go one after another, like [a-z]{0,}. [a-z]+ means that at least one Latin letter can go one after another, but maximal number is not specified, like [a-z]{1,}. [a-z]? means that number of Latin letters is not more than 1, letter also can be missed, like [a-z]{0,1}.

Condition search of similarities which is described by nonspecial symbol is a literal.

Some of quantifiers given above can be applied to literals.

If the abcd{1,4}efg condition search will be applied for abcefg string, the similarities won't be found because a quantifier {1,4} means that from one to four digits "d" will be after abc and before efg.

3.2.3 Symbol sorting

Symbol sorting can contain some literal or literal intervals. To describe literal intervals the '-' symbol is used which is staying between the first and last symbol in an interval. Examples of different intervals of one symbol sorting are as follows: [1-5] - digits from 1 to 5, [a-f] - Latin letters from a to f, [a-fq-x] - Latin letters from a to f and from q to x, two ranges are in use in the last symbol sorting.

If some of symbols: a or g or 7 or 4 can be in the determined positions, than the symbol sorting will be as follows: [ag47].

It's possible to specify allowable literals in a symbol sorting. Specifying of literals can be combined with specifying intervals: [14a-kz] means that a symbol in a string can be similar with 1, 4, Latin letters from a to k and with z letter. Letters, digits, punctuation marks, mathematical signs can be a literal. For example, ',' (coma), '!' (exclamation point), '+' (plus). It's possible to use '-' (minus) symbol even if it used for interval specifying. If specify '-' between a and z, it will be interval, but if specify it after opened square bracket - it will be minus. For example, [-,a-z] means that symbol sorting contain minus, coma and Latin letters from a to z.

To specify symbol sorting containing all symbols except of the specified, e.g. all symbols except of a,b,c, use the special negation symbol: ^ (cover). It's required to specify: [^abc] - all symbols (not letters, but symbols) except of a,b,c Latin letters.

It's possible to train to write templates here.

3.2.4 Special symbols

To specify, for example, space in a search condition, it's required to make it visible, i.e. enter some symbol or set of symbols which will be identified as invisible.

\s - letter s after the backslash describe space or tabulation character. Space also can be specified the general way, but [a-z\s] template will be more clear than [a-z]. Use this symbols carefully because it's the same as space, tabulation and a symbol of new string.

\S - visible symbols. i.e. all symbols which are not coincide with \s

\w - special symbol exchanging the whole symbol sorting and containing all symbols which can be specified in a word, mostly it is [a-zA-Z_], but it depends on installed localization, Unicode supporting, etc.

\W - all symbols which are not coincide with \w or [^a-zA-A_]

\d - all digits, i.e. [0-9]

\D - non-digit symbols

Often special symbols describe commonly used character sets, but these sets have boundaries, i.e. digits, letters and underlining or invisible symbols. To describe all symbols use a point. For example,

.{5} - five different symbols.

To describe a point, set the backslash before it: \.

To search a backslash after which a point is going, enter the double slash as literal in a search condition:

Similarly, to enter two backslashes, it's required to double them: \\\\

3.2.5 Select condition

It's possible to create a search condition using regular expressions. For example, there are two groups of literals: one of them contain two literals b and e going one after another, the other group contain nine literals going one after another: n, o, t, \s, t, o,\s, b, e.

\s is a space. Group of literals - is a sequence of symbols describing by symbol sorting or by literals. Group of literals is presenting in a brackets. They also save registered group of literals to special variables.

Examples of literal groups are follows:

(be)
(not\sto\sbe)

Select from two literal groups: (be)|(not\sto\sbe), "|" symbol between literals is a selection condition "or". Regular expression of checking:

(be)|(not\sto\sbe)

Regular expression is a similar if:

- string is not equal "be"
- or if string is not equal "not to be"

It's possible to select between literals and between groups of literals. Literal groups are combining in brackets. If it's required to select between two single literals, than two literals between there is a vertical line are to be combined in brackets.

Example of selection from two literals: s(o|u)n is son and sun.

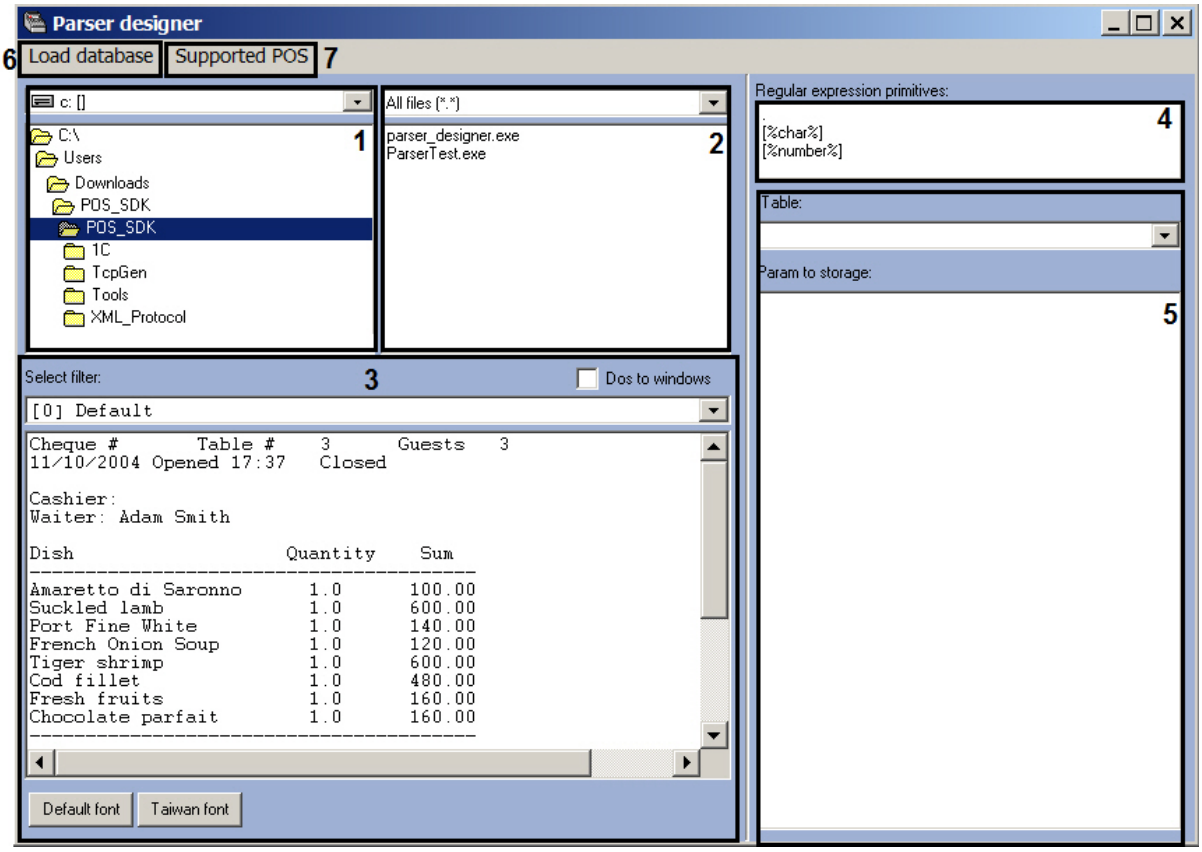
Example of selection from two literal groups: (son)|(sun) is also son and sun.

In case of selection between single literals or between literal groups, literals are to be combined in brackets.

To enter "|" literal, it's require to enter a special symbol behind it which will mean that vertical line is a literal. This special symbol is a backslash: \|

3.3 Description of the parser_designer.exe utility interface

View of the parser_designer.exe utility window is shown in the figure.



Areas designed for the following functions are located in working area of the utility:

1. Select folder with log from a POS-terminal (1).
2. Select log (2).
3. Display log text (3).
    a. The **Select filter** drop-down list is for protocol selection.
    b. The **Dos to windows** checkbox is for recoding symbols if log text is in the DOS coding.
    c. **Default font** and **Taiwan font** buttons are for selecting fonts.
4. Templates of regular expressions.
5. List of table fields selected in the **Table** field from the connected database (5). POS_LOG_MASTER and POS_LOG_DETAIL tables are used.
    a. The POS_LOG_MASTER table stores data about title and end receipt part: receipt number, name and code of cashier, total sum, exchange sum, etc.
    b. The POS_LOG_DETAIL table stores data of receipt "body", i.e. information about good from this receipt.

Utility main menu allows access to the following functions:

1. **Load database** – settings of connection to POS database (6).

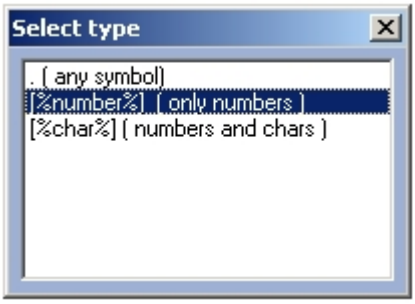
> **Note.**
>
> For correct connection to the POS database use Windows authorization because functionality of password saving when connecting to database using login and password was disabled at the moment of writing this documentation.

2. **Supported POS** - displaying list of supported POS-terminals in the current utility version (7).

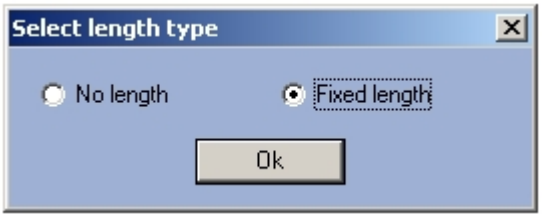
3.4 Creating a parser

To create a parser, do the following:

1. Select the required text fragment in the window of log text. In the window 5 select a field in a table in which text fragment is to be written. Click the left mouse button and replace field to window with log text.
2. In the opened window select a format of template. The following formats are available:
   a. . - some symbol
   b. [%number%] - only digits and decimal point
   c. [%char%] - digits and symbols



3. Select type of length. The following types of length are available:
4. No length - not limited length
5. Fixed length - fix length which is counting automatically by the selected fragment.



6. String in the log field takes a form of a template with specifying a field name. Construction like ? P<item_name> specified a table field to which the recognized value will be loaded - e.g. it's a check_number field. If required. it's possible to change the regular expression going after this text.

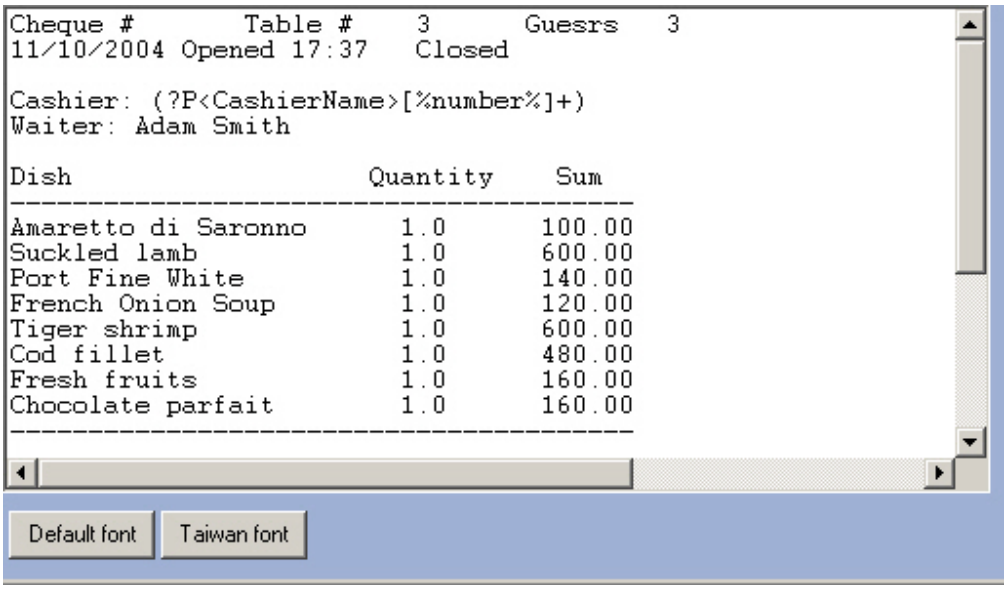
```
(?P<CashierName>[%number%]+)Cheque #        Table #     3
11/10/2004 Opened 17:37   Closed

Cashier:
Waiter: Adam Smith

Dish                    Quantity     Sum
-----------------------------------------
Amaretto di Saronno        1.0       100.00
Suckled lamb               1.0       600.00
Port Fine White            1.0       140.00
French Onion Soup          1.0       120.00
```

7. Create templates for all required fields of database the similar way.

For example, a template for cashier name is shown in the figure (length is not limited).

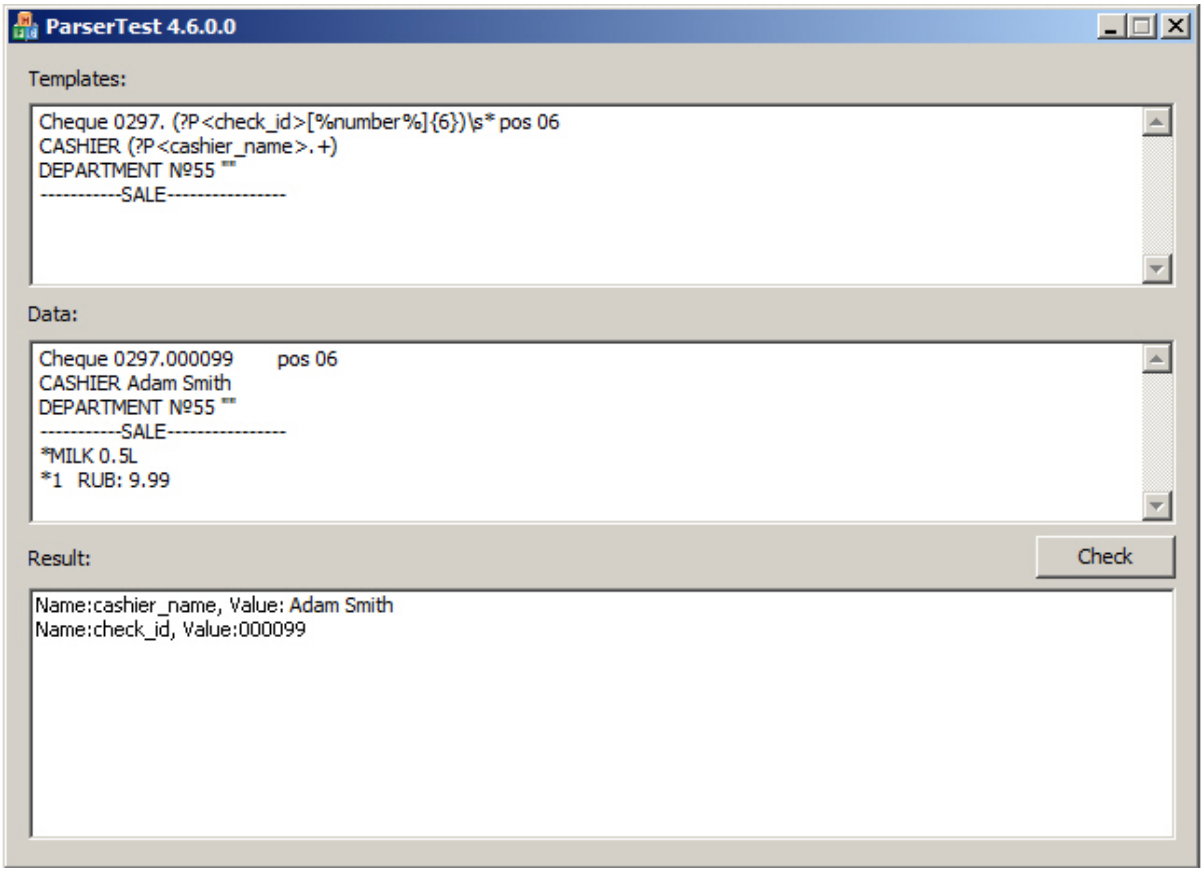


Selected fragment is a template of title receipt part.

> **Note.**
>
> Data from body receipt part are saving in the table different from table in which data from title and end parts are saving. So before creating a parser select the corresponding table from the **Table** drop-down list.

3.5 Checking a created template

To check working of the created template use the ParserTest.exe utility.



Template is checked as follows:

1. Copy fragment of parser to the **Templates** field.
2. Put fragment of receipt text to the **Data** field.
3. Click the **Check** button.
4. If template is correct, than names of fields which will be loaded to database and their values will display in the **Result** field. Otherwise, the template is failed.

   > **Note.**
   >
   > Unnecessary spaces in begin of strings, signs of string end, etc. can be errors. Template from example in the figure:
   > **«(?P<check_id>[%number%]{6}) pos 06 »**
   > from the figure in the Creation a parser section is transformed the following way:
   > **«(?P<check_id>[%number%]{6})\s*pos 06».**
   > Spaces are changed to "\s" to avoid errors of spaces number.

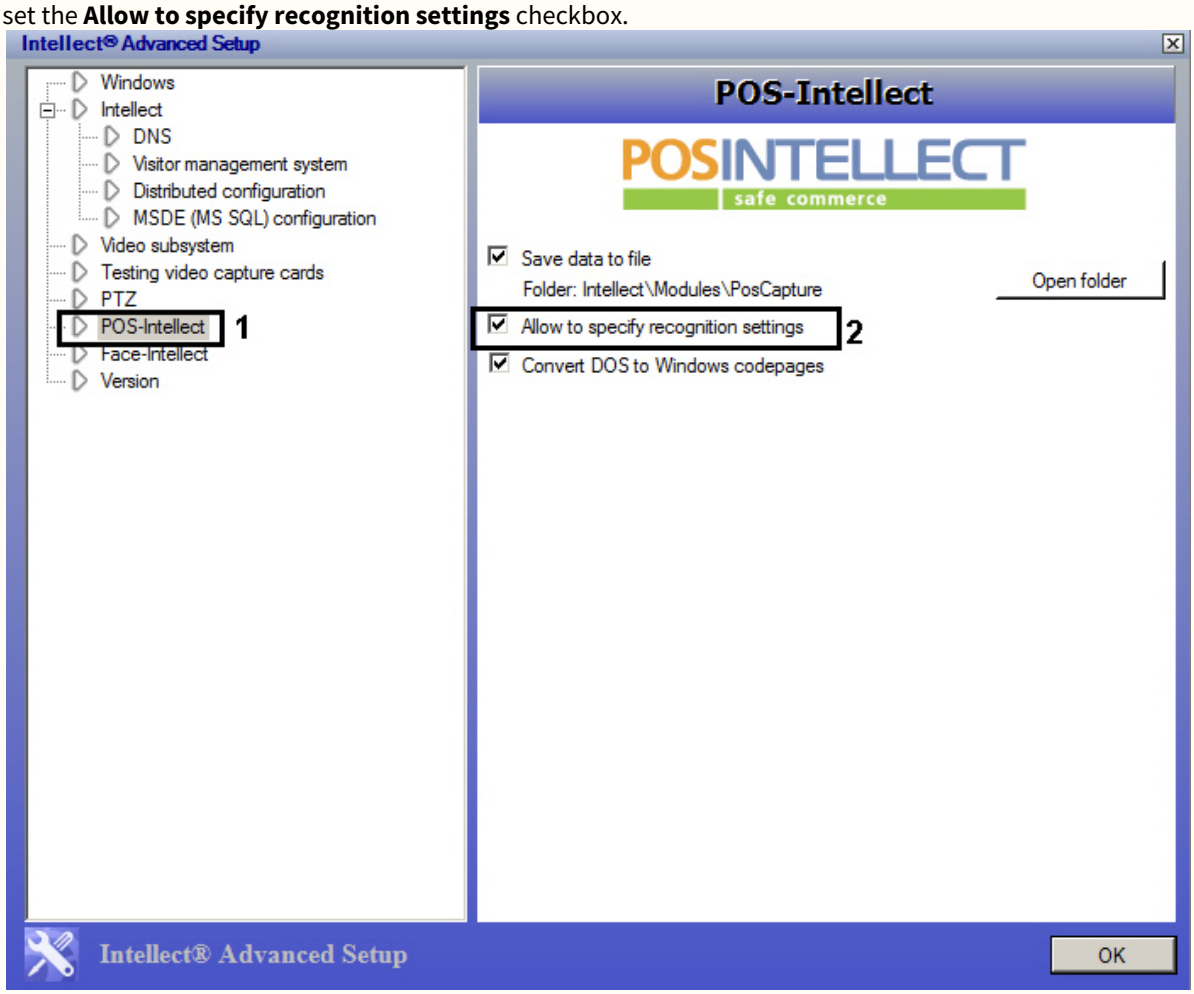
5. If data in the **Result** field are correct, replace template to parser settings of the POS-terminal object using functions of copying and pasting text (see the Configuring parser in the POS-Intellect software section).

> **Note.**
>
> For examples from figures, replace the template to the **Receipt prolog** section. Other receipt sections are adding in the same way.

3.6 Configuring parser in the POS-Intellect software

Attention!

To get possibility of templates editing, run the tweaki.exe utility , go to the **POS-Intellect** section (1) and set the **Allow to specify recognition settings** checkbox.



When templates for all needed fields are created and checked, it's required to create a parser file. Do the following:

1. Run the *POS-Intellect* software. Go to the **Hardware** tab of the **System settings** dialog window and select the required **POS-terminal** object for which parser is configuring. Click the **Advanced** button on the settings panel of this object.
2. In the opened window go to the **Parser settings** tab.
3. Configure prolog/end/body of receipt:
    a. Using the + button add a rule of structuring (1).
    b. Copy test of structuring rule formed using the parser_designer.exe utility and checked using the ParserTest.exe utility (2).
    c. Repeat steps 3.a-3.b for all required structuring rules of prolog/end/body of receipt.
4. Specify name of a parser in the **Template name** field.
5. Click the **Export** button to save parser as file with .prl resolution.

Note.

Detailed process of parser creation is given in the POS-Intellect software package. Administrator's Guide document.

4 Libraries for working with POS-Intellect software package

> **Attention!**
>
> Libraries from the POS SDK package are given as example. AxxonSoft company is not responsible for their operation.

4.1 COM. Library for working with TCP/IP connection

The library is designed for integration of POS software (working on Windows) with the **POS-Intellect** system. It hides code of providing TCP/IP communication from programmers. For example, it's possible to resend data to network using TCP/IP protocol by some modifications of program code if there is code of data sending to receipt printer. Library provides COM object which is possible to use from some space supporting COM technology (e.g. Visual Basic, Delphi, 1S).

> **Attention!**
>
> It's required to register the regsvr32.exe poslib.dll library previously.

Application (for VB.NET):

Create object:

```
Dim pos As Object
pos = CreateObject("Poslib.Net")
```

Start process of connection setup:

```
Dim port As System.UInt32
port = Convert.ToUInt32("5000")
pos.Open(ip, port)
```

Used method:

HRESULT Open(BSTR ip_address,DWORD port) - is called at the beginning of working with library, initiates connection setup

- ip_address – server address
- port – connection port

Send text:

```
pos.Send("Test!" & vbNewLine)
```

Used method:

```
HRESULT Send(BSTR str);
```

- **-** str – sending message to system

Close connection:

```
pos.Close()
```

Used method:

- HRESULT Close() – is called at the end of working with library

When calling the **Open** method, the library is connecting using TCP/IP and restoring connection if it was lost. All callings are asynchronous and they don't have an impact to the main flow. Data are sending from a separate flow. All used methods are thread-safe.

When calling the **Send** method, data will be send if connection established and won't be send if connection lost. There is no any confirmation of successful/failed data sending.
It's required to specify type of TCP/IP connection and enter a port number specified as method of **Open** function on the settings panel of the **POS-terminal** object of the POS-Intellect system.

4.2 ActiveX component for working with TCP/IP connection

The library is designed for integration of POS software (working on Windows OS) with the **POS-Intellect** system. It hides code of providing TCP/IP communication from programmers. For example, it's possible to resend data to network using TCP/IP protocol by some modifications of program code if there is code of data sending to receipt printer. Library provides ActiveX component which is possible to use from some space supporting ActiveX technology (e.g. Visual Basic, Delphi, 1S).

> **Attention!**
>
> It's required to register the regsvr32.exe posx.ocx library previously.

Used methods:

1. Open(BSTR ip_address,DWORD port) - is called at the beginning of working with library, initiates connection setup
    a. ip_address – server address
    b. port – connection port
2. Send(BSTR str);
    a. str – sending message to system
3. Close() - is called at the end of working with library

When calling the **Open** method, the library is connecting using TCP/IP and restoring connection if it was lost. All callings are asynchronous and they don't have an impact to the main flow. Data are sending from a separate flow. All used methods are thread-safe.

When calling the **Send** method, data will be send if connection established and won't be send if connection lost. There is no any confirmation of successful/failed data sending.
It's required to specify type of TCP/IP connection and enter a port number specified as method of **Open** function on the settings panel of the **POS-terminal** object of the POS-Intellect system.

4.3 DLL. Library for working with TCP/IP connection

The library is designed for integration of POS software (working on Windows) with the **POS-Intellect** system. It hides code of providing TCP/IP communication from programmers. For example, it's possible to resend data to network using TCP/IP protocol by some modifications of program code if there is code of data sending to receipt printer.

Library functions:

1. **void __stdcall Open (LPCTSTR id,LPCTSTR ip_address,DWORD port)** – is called at the beginning of working with library, initiates connection setup

- ip_address – server address
- port – connection port

1. **void __stdcall Close (LPCTSTR id)** – is called at the end of working with library
2. **void __stdcall Send(LPCTSTR id,LPCTSTR str)**

- str – sending message to system.

General parameter **LPCTSTR** id for all functions is a connection identifier. It's allowable to have several connections with different servers recognizing them by this identifier.

When calling the **Open** method, the library is connecting using TCP/IP and restoring connection if it was lost. All callings are asynchronous and they don't have an impact to the main flow. Data are sending from a separate flow. All used methods are thread-safe.

When calling the **Send** method, data will be send if connection established and won't be send if connection lost. There is no any confirmation of successful/failed data sending.

It's required to specify type of TCP/IP connection and enter a port number specified as method of **Open** function on the settings panel of the **POS-terminal** object of the **POS-Intellect** system.

5 Interaction with 1S program

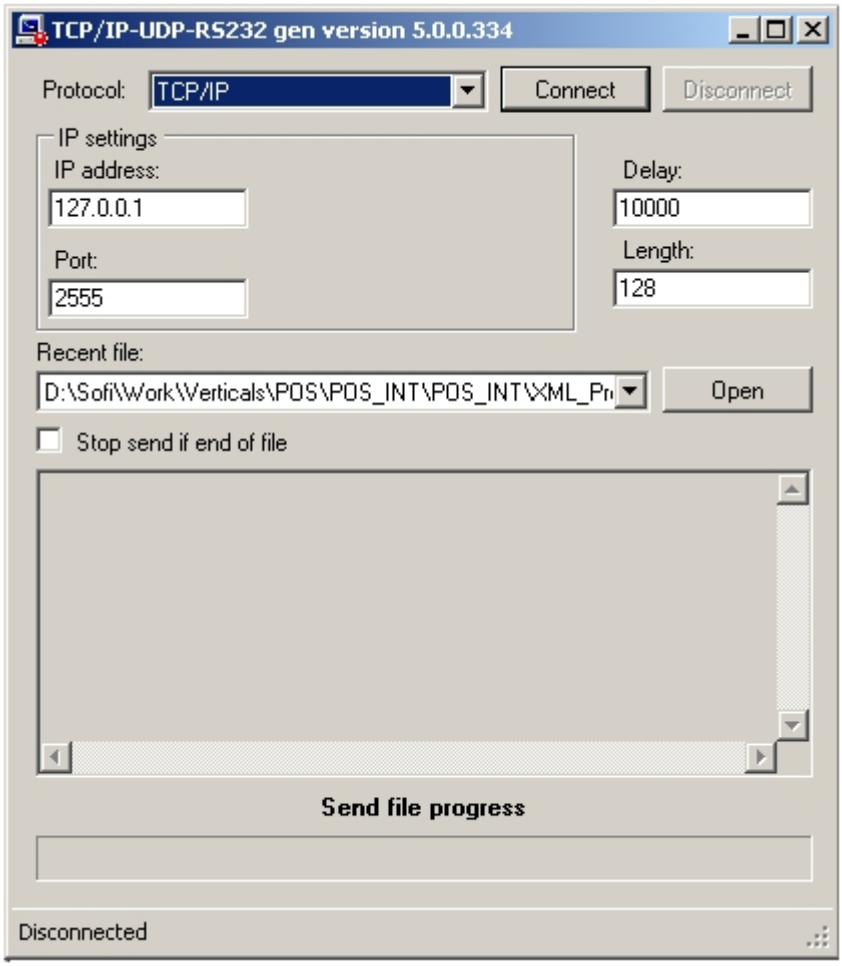
POS SDK package contains example of the 1S program enabling sending messages to the *POS-Intellect* system.

6 Appendix 1. The tpcgen.exe utility for sending text data to the POS-Intellect system

6.1 General information about the tpcgen.exe utility

The tpcgen.exe utility allows sending log from the pos-terminal to the *POS-Intellect* software package to check operation of the configured **POS-terminal** object.

Interface of the tpcgen.exe utility is shown in the figure,



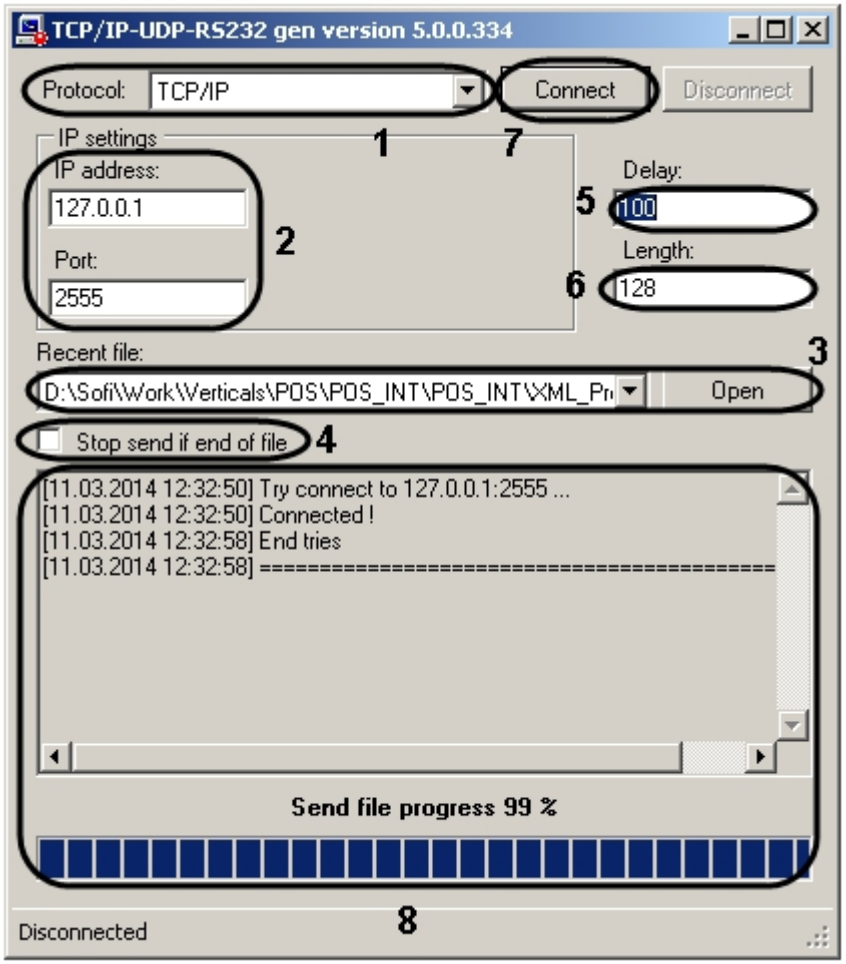
Sending data will be processed by the *POS-Intellect* system in operation mode: video titling, data recording to database, etc.

> **Note.**
>
> Download the utility as part of the POS SDK here.

6.2 Working with the tpcgen.exe utility

Working with the tpcgen.exe utility is performed as follows:

1. Run the tcpgen.exe utility.



2. From the **Protocol** drop-down list select the connection protocol on which data will send to the *POS-Intellect* (1). TCP/IP, UDP and RS232 protocols are available.
3. Configure connection parameters if default parameters are not valid (2). IP-address and port of connection to the *POS-Intellect* are specified when using the TCP/IP and UDP protocols, number and speed of port are specified when using the RS232 protocol.
4. Click the **Open** button and select a text file with pos-terminal log using the Windows standard dialog of files opening (3).
5. To stop data sending to the *POS-Intellect* when log file will end, set the **Stop send if end of file** checkbox (4). To send data cyclically, keep the checkbox deselected.
6. Specify the required delay between packages in milliseconds in the **Delay** field (5).
7. Specify the package length in bytes in the **Length** field 96).
8. Click the **Connect** button (7).
9. Process of sending will display in the status field (8). If data sending is impossible, description of problem will be specified in the field.

> **Note.**
>
> To stop data sending click the **Disconnect** button.

Working with the tcpge.exe is completed.