Openbravo POS Scripting Tutorial

Languages: **English** | Français | Translate this article...

Contents [hide]

Introduction

This guide refers to the Openbravo POS 2.10 version and next versions. In this version Openbravo POS has extended the scripting capabilities available in previous versions to give developers more power to create rules and to adapt Openbravo POS to their business rules and country regulations.

The scripting capabilities this release offers has the following benefits:

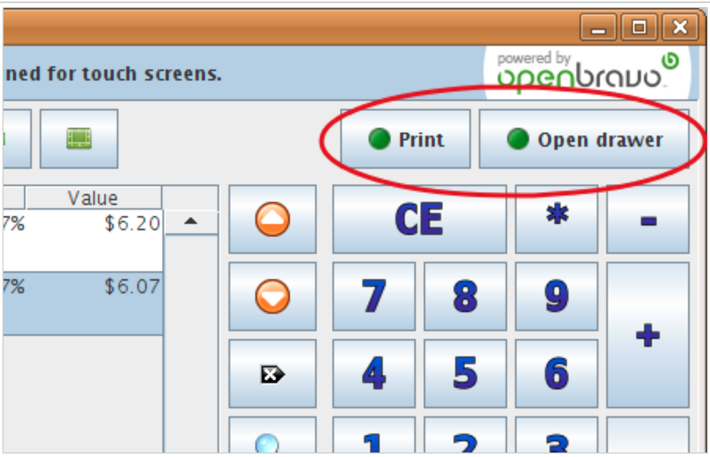
- Simple scripting language very similar to Java and easy to learn.
- No need to compile Openbravo POS. You only need the binary package.
- Powerful object model. When writing a script for Openbravo POS you have access to all Openbravo POS objects methods and properties, and access to all the Java API.
- Automatic deploy to all POS terminals. You only have to create the scripting functionalities in one terminal and all terminals connected to the same database will automatically use the scripting functionalities created.
- Security. The execution of scripts is role based. You can allow or deny to execute scripting functionalities based on the user's role.

Execution of scripts

There are two different moments where script code is executed: Buttons of the sales panel and events of the sales panel. All the execution moments are defined in the resource *Ticket.Buttons*.

Buttons

Scripting buttons appear the sales panel top right and can execute a template or a script when the user press the button.



Openbravo POS script buttons.

A button is defined as a new node of type *button* created in the resource *Ticket.Buttons*. This is an example of two buttons:

```
<button key="button.print" name="button.print" template="Printer.TicketPreview"/>
<button key="button.discount" name="button.discount" code="Script.Discount"/>
```

The attributes of a button are:

- **key**. Is a string that uniquely identifies this button. This key must be added to the permissions list of the role definition in the roles panel to allow users to execute this button
- **name**. Is the key of the localized label to show. See the OpenbravoPOS Localization document.
- **image**. Is the name of the image to show in the button. By default is a green spot.
- **template**. This attribute defines the resource template to execute when the user press the button. This attribute is used to print custom receipts or display custom messages to the customer display. One typical usage is to print a preview of the current receipt or to open the drawer.
- **code**. This attribute defines the scripting resource to execute when the user press the button. This code is used to create discounts, modify taxes assigned to the receipt, edit receipt line attributes, print templates... You only can have one of the two attributes in one button, *template* or *code*.

Events

Events are scripts that are executed when an action is executed in the sales panel.

An event is defined as a new node of type *button* created in the resource *Ticket.Buttons*. This is an example of one event executed when a new receipt line is added to the receipt:

```
<event key="ticket.addline" code="event.addline"/>
```

The event attributes are:

- **key**. Is the key of the event and defines the moment when Openbravo POS executes it. It can be one of the following: *ticket.change*, *ticket.addline*, *ticket.removeline* or *ticket.setline*.
- **code**. This attribute defines the scripting resource to execute when the event is launched.

The following events are published:

- **ticket.change**. This event is launched after any modification of the lines of a receipt in the sales panel.
- **ticket.addline**. This event is launched before a new line is added to the receipt. The script code receives an extra object *line* with the details of the line being added. The script code can cancel the addition of the line returning a value distinct of null.
- **ticket.setline**. This event is launched before a line is modified in the receipt. The script code receives two extra objects *line* with the details of the line modifications and *index* with the index of the line that will be modified. The script code can cancel the modification of the line returning a value distinct of null.
- **ticket.removeline**. This event is launched before a new line is removed from the receipt. The script code receives an extra object *index* with the index of the line that will be removed. The script code can cancel the removal of the line returning a value distinct of null.
- **ticket.show**. This event is launched before the receipt is shown. This can be because is a new receipt, or is a pending receipt that comes to the sales panel again.
- **ticket.total**. This event is launched after the user press the button *equals*, after the taxes lines are calculated and before the payments dialog is shown. The script code can cancel the process of closing the sale and come back to the sales panel returning a value distinct of null.
- **ticket.save**. This event is launched after the user presses *OK* in the payments dialog, and before the receipt is stored in the database and the receipt is printed. The script code can can cancel the process and come back to the sales panel returning a value distinct of null.
- **ticket.close**. This event is launched after the receipt is stored in the database and the receipt number has been assigned. This is the preferred method to print reports based on the current receipt. A variable *print* will be passed to the script that indicates whether the print-button was selected in the payment dialog.

The object model

There are five objects passed to the scripts: *ticket* that contains the data of the current ticket, *taxes* that contains the list of available taxes record, *place* that in restaurant mode contains the table object, in other modes contains null, *user* that contains the user object and *sales* that contains utility methods of the sales panel.

- **ticket**. Contains all the information related to the ticket is an object of type *TicketInfo*
- **place**. Contains the table in restaurant mode. Is an object of type String.
- **taxes**. Contains a list of all the taxes. Is a map that contains objects of type *TaxInfo*.
- **taxeslogic**. Contains a list of methods that calculates the taxes lines of a current receipt. Is an object of type *TaxesLogic*.
- **user**. Contains the user that is currently logged in the application. Is an object of type AppUser.
- **sales**. Contains utility methods of the sales panel. Is an object of class *ScriptObject*.

Utility methods

The *sales* object available in scripts button and events contains a list of utility methods of the sales panel.

- **getInputValue()**. Returns the value typed in the sales panel using the on screen keyboard.
- **getSelectedIndex()**. Returns the index of the selected receipt line in the sales panel. If the receipt has no lines returs -1.
- **setSelectedIndex(int i)**. Sets the index of the selected receipt line in the sales panel.
- **printReport(String resourcefile)**. Prints a report using the current receipt using the included Jasper Reports engine. This command is useful if you want to print elaborated invoices from the sales panel.

- **printTicket(String resourcename)**. Prints a receipt in the receipt printer.
- **evalScript(String code, ScriptArg... args)**. Calls another script.

Ticketline Info

There are number of function calls available for working with a single ticketline:

- setTicket(String ticket, int line) set the ticketline to line
- getTicket() return the current ticket line
- setMultiply(double dvalue) set the number of units to dvalue
- getMultiply() return the number of units
- setPrice(double dvalue) set the unit price to dvalue
- getPrice() return the unit price
- setPriceTax(double dvalue) set the unit price plus tax to dvalue
- getPriceTax() return the unit price plus tax

this list isn't complete and all public functions can be found in:

src-pos/com/openbravo/pos/ticket/TicketLineInfo.java

Attributes

Receipts, receipt lines and products have a list of properties called *attributes*. These attributes can be used for any purpose the developer needs. When adding a new receipt line Openbravo POS copies the attributes of the product to the line, Openbravo POS does not manage in any other way the content and values of these attributes, and is the developer of script buttons and events the responsible of managing its values.

These attributes are stored in the database and can be edited in the products panel as an XML text file. This is an example of a product attributes record:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>Asian salad attributes</comment>
    <entry key="notes">Only available in spring.</entry>
    <entry key="ingredients">Lettuce, and other vegetables</entry>
</properties>
```

These properties can be accessed when printing a receipt (eg in the resource Printer.Ticket) by using the expression:

```
$ticketline.getProperty("ingredients", "")
```

where $ticketline is derived from the Velocity expression:

```
#foreach ($ticketline in $ticket.getLines())
...
#end
```

The second parameter for getProperty() is optional and is a default value to be returned if the <entry> does not exist.

Examples

This is a list of functionality that can be implemented with scripts.

Add notes to receipt lines

With this functionality you create a button that when selected a line in the receipt it allows to edit notes associated to this line. The script code of the button is:

```
index = sales.getSelectedIndex();
if (index >= 0) {
    line = ticket.getLine(index);
    value = javax.swing.JOptionPane.showInputDialog("Line notes", line.getProperty("notes"));
    if (value != null) {
        // the user pressed OK
        line.setProperty("notes", value);
    }
}
```

Add a discount to the selected line

The following code when associated to a button you will be able to add a discount to the selected line. The discount amount is calculate as the percentage typed in the on screen keyboard.

```
// % discount line

import com.openbravo.format.Formats;
import com.openbravo.pos.ticket.TicketLineInfo;
import com.openbravo.pos.ticket.TicketProductInfo;

discountrate = sales.getInputValue() / 100.0;
```

```
index = sales.getSelectedIndex();
if (index >= 0) {
    line = ticket.getLine(index);
    if (line.getPrice() > 0.0 && discountrate > 0.0) {

        sdiscount = Formats.PERCENT.formatValue(discountrate);
        ticket.insertLine(index + 1,
            new TicketLineInfo(
                    "Discount " + sdiscount,
                    line.getProductTaxCategoryID(),
                    line.getMultiply(),
                    -line.getPrice () * discountrate,
                    line.getTaxInfo()));
        sales.setSelectedIndex(index + 1);
    } else {
         java.awt.Toolkit.getDefaultToolkit().beep();
    }
} else {
    java.awt.Toolkit.getDefaultToolkit().beep();
}
```

Add a discount to the total of the receipt

Important: This script only works release 2.30 and higher. It does not work in earlier versions (1.0 - 2.20).

The following code when associated to a button adds a 15% discount based on the total of the receipt. If the receipt includes items with different taxes it will add one discount line for each group of taxes. The discount amount is calculate as the percentage typed in the on screen keyboard.

```
// % Discount for the total of the receipt

import com.openbravo.format.Formats;
import com.openbravo.pos.ticket.TicketLineInfo;
import com.openbravo.pos.ticket.TicketProductInfo;
import java.util.Properties;

discountrate = 0.15;

total = ticket.getTotal();
if (total > 0.0) {
    sdiscount = Formats.PERCENT.formatValue(discountrate);

    taxes = ticket.getTaxLines();
    for (int i = 0; i < taxes.length; i++) {
        taxline = taxes[i];
        ticket.insertLine(ticket.getLinesCount(),
            new TicketLineInfo(
                "Discount " + sdiscount + " of " + taxline.printSubTotal(),
                taxline.getTaxInfo().getTaxCategoryID(),
                1.0,
                -taxline.getSubTotal() * discountrate,
                taxline.getTaxInfo()));
    }
    sales.setSelectedIndex(ticket.getLinesCount() - 1);
} else {
    java.awt.Toolkit.getDefaultToolkit().beep();
}
```

NOTE: Once the button is created and the script is associated permission to use the button must be granted. See User Roles for details.

Display prompt with change value of the sale

The following code is associated to a **ticket.close** event and will show the change value of the sale.

```
import com.openbravo.pos.payment.PaymentInfo;
import javax.swing.JOptionPane;

boolean isCash = false;
String change = "";
PaymentInfo p = ticket.payments.getFirst();

if ("cash".equals(p.getName())) {
    isCash = true;
    change = p.printChange();
}
if(isCash) {
    JOptionPane.showMessageDialog(null, "Return " + change, "Change", JOptionPane.PLAIN_MESSAGE);
}
```

To implement the show change ticket.close script do the following:

Step 1) Set Up the resource

- Open Openbravo and login as Administrator (or with a user with the role of Administrator)
- Click on Maintenance under Administration in the sidebar
- Click on the Resources button (under the heading Point of Sale)
- Click on the icon that looks like a yellow star to create a blank resource
- under the name Resource type the text: ticket.close and make sure the drop down is set to text.
- Copy and paste the code here above into the empty resource
- Click on the icon that looks like a floppy disk to save

Step 2) Set the event to happen

- In the list of resources to the left choose Ticket.Buttons
- under the <configuration> heading add the following code

```
<event key="ticket.close" code="ticket.close"/>
```

- click on the icon with the floppy to save

Step 3) restart OpenbravoPOS to make the changes permanant

- click on the exit button
- reopen Openbravo and make a complete cash sale.

Success: you now see a pop up with the amount of change to be given.

Create a report to print an invoice at the end of a sale

If you need to create invoices for your customers and the receipt printer is not the solution to print the invoice you can create a complex report using the receipt data to fill in the report and print it to a regular printer. This example explains how to create a basic report using the receipt data using iReport and print it at end of every sale.

First you need to execute the script that prints the report at the end of every sale. Edit the resource *Ticket.Buttons* and add the following line:

```
<event key="ticket.close" code="code.printreport"/>
```

This line creates an event that executes the script *code.printreport* at the end of every sale.

Then you have to create the script that prints the report. To do this create a new report with the name *code.printreport* and add the following line:

```
sales.printReport("/com/openbravo/reports/ticketsample");
```

This line of code executed the report *ticketsample.jrxml* that is in the reports folder.

Then in the folder *reports/com/openbravo/reports/* of the installation of Openbravo POS create the following files:

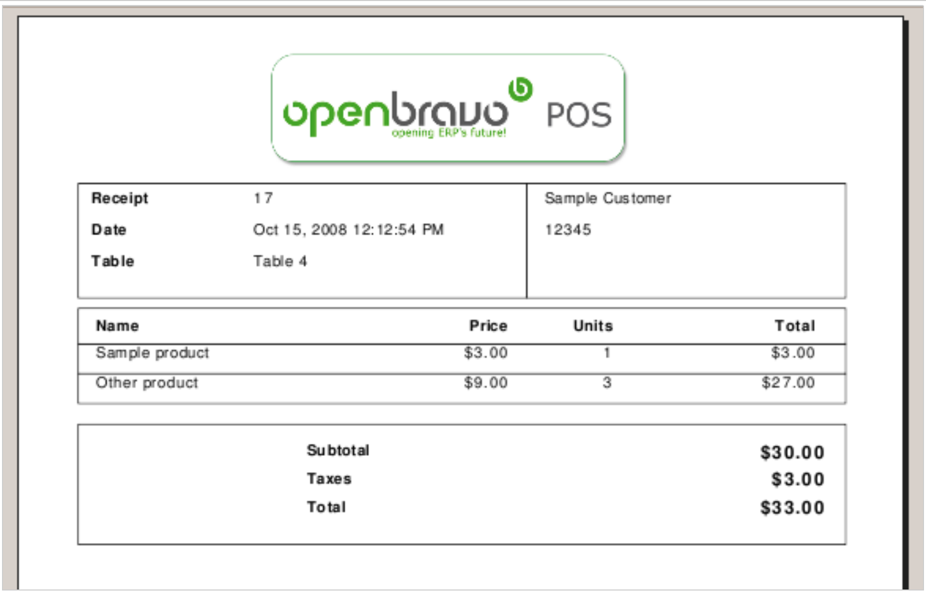
- ticketsample.jrxml
- ticketsample_lines.jrxml

The file *ticketsample.jrxml* is the report that prints the general data of the receipt and *ticket_sample_lines.jrxml* is the subreport that prints the data of the receipt lines. You can modify these files using the graphical tool iReport and adapt the layout of the report to your business requirements.

Add a logo image to *reports/com/openbravo/reports/logo.png*

To finish, in Openbravo POS, in the configuration panel select the reports printer you want to use to print the invoices report. Restart Openbravo POS and test the new report performing a sale.

This is a sample invoice using the sample report files provided.


openbravo POS
opening ERP's future!

Receipt	17	Sample Customer
Date	Oct 15, 2008 12:12:54 PM	12345
Table	Table 4	

Name	Price	Units	Total
Sample product	$3.00	1	$3.00
Other product	$9.00	3	$27.00

Subtotal	$30.00
Taxes	$3.00
Total	$33.00

Invoice sample.

Category: Development POS