Customer Panel - Developer Guide

In this book you will find all the information you need to develop your own widgets for the Intalio Customer Panel

- [Modules](#)
  - [Introduction](#)
  - [System Configuration](#)

- [Widgets](#)
  - [XML Configuration](#)
  - [Template](#)
  - [Multilanguage translation support](#)
  - [Installation file .sopm](#)
  - [Writing a first customer-panel widget](#)

Modules

Introduction

The chapter focuses strictly on the development section. For administrative information check out an [administration tutorial](#).

Main assumptions

The customer panel comes with an easy way of developing and deploying new modules to the panel. It provides you an out-of-the-box configuration with a few steps needed for proper running.

System Configuration

XML Configuration File

Configuration determines the main aspects of the module. Each one regardless of future different configurations has the same core model:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<otrs_config version="2.0" init="Application">
<Setting Name="OTRSFrontendModule::<NAVIGATION_NAME>" Required="0" Valid="0">
        <Description Translatable="1">Module random description</Description>
        <Navigation>CustomerFrontend</Navigation>
        <Value>
            <Hash>
                <Item Key="id">LatestActivity</Item>
                <Item Key="type">[module/redirect/hidden]</Item>
                <Item Key="text">Latest Activity</Item>
                <Item Key="to">/module/LatestActivity</Item>
                <Item Key="icon">timeline</Item>
                <Item Key="priority">110</Item>
                <!--OPTIONAL-->
                <Item Key="mode">[newWindow/tab]</Item>
                <Item ValueType="Textarea" Key="widgets">
                    [
                        {
                        "props": {
                        },
                        "id": "5",
                        "widgetFile": "WIDGET_Timeline.js",
                        "compiled": false,
                        "view" : "Dashboard",
                        "slot" : "slot3",
                        "name": "Latest Activity"
                    }
                    ]
                </Item>
            </Hash>
```

```
          </Value>

       </Setting>

  </otrs_config>
```

`<Hash>` specifies main configurations settings:

- *<id>*
  An **identification key** for your module. Should be unique for every module**.**
- *<type>*
  Determines the type of the module. For further information check out the administration guide for modules.
- *<text>*
  Display label for frontend panel.
- *<to>*
  Specify the URL path to the module. If redirect was chosen as a type this field specifies redirection link.
- *<icon>*

  Specify the name of the icon to display. Available icons can be found in this icon library.
- *<priority>*
  The higher the priority the higher the module is on the menu.
- *<widgets>*
  Set an array of **widget configurations** that should be placed in the module. More on

  widget configuration here.
- **[Optional]** *<mode>*
  Active if type set to **redirect**. Specifies the way in which the system will display the redirection page.
  Available options are:
  - *newWindow*
  - *tab*

Available modules

Another step is to include a module name in the `CustomerOTRSFrontendModule::AvailableModules` in system configuration.

It's important to pass the module's **id** as a configuration key.

Permissions

The last step is to set visibility of the module for predefined CustomerUser groups in the `CustomerOTRSFrontendModule::Permissions`

CustomerOTRSFrontendModule::Permissions   OTRSFrontend.xml ≡

CUWCommitments
→ users

CUWLedger
→ users

CUWReceivables
→ users

CustomerDashboard
→ users

CustomerForm
→ users

CustomerUserSettings
→ users

ITDepartment
→ users

LatestActivity
→ users

NewTicket
→ users

Tickets
→ users

Modules available to customers.

Add new entry

If your module is not visible in the menu try to initiate the configuration rebuild from within the OTRS console.
1. Login on your server using SSH
2. Change user to otrs
3. Run config rebuild command -> `/opt/otrs/bin/otrs.Console.pl Maint::Config::Rebuild`

After completion of these three steps, your module is ready to go!

Widgets

XML Configuration

Widget's development process consists of a few simple steps to let your widget start work for you.

XML configuration file:

The core of the whole widget. This one specifies visibility for specific groups, determines in which module should widget be located and how should it be called in the panel. The syntax and layout configuration is similar to every XML OTRS configuration.

There are two ways to configure widgets in the module:

- **Specifying the widgets array directly in the module's configuration file:**

```
<Setting Name="OTRSFrontendModule::LatestActivity" Required="0" Valid="0">
    <Description Translatable="1">Customer user latest activity menu module.</Description>
    <Navigation>CustomerFrontend</Navigation>
    <Value>
        <Hash>
            <Item Key="id">LatestActivity</Item>
            <Item Key="type">module</Item>
            <Item Key="text">Latest Activity</Item>
            <Item Key="to">/module/LatestActivity</Item>
            <Item Key="icon">timeline</Item>
            <Item Key="priority">110</Item>
            <Item ValueType="Textarea" Key="widgets">
                [
                    {
                    "props": {
                    },
                    "id": "5",
                    "widgetFile": "WIDGET_Timeline.js",
                    "compiled": false,
                    "view" : "Dashboard",
                    "slot" : "slot3",
```

```
                            "name": "Latest Activity"
                    }
                ]
            </Item>
        </Hash>
    </Value>
</Setting>
```

- **Specifying widget as a separate configuration:**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<otrs_config version="2.0" init="Application">



    <!--Core configuration containing module settings and additional properties-->
    <!--Setting location's name should be set in "OTRSFrontendModule::XYZ-->
      <Setting Name="OTRSFrontendModule::<MODULE_NAME>Widget#1000" Required="0" Valid="1">
        <!--Description visible in system configuration-->
        <Description Translatable="1">Additional Ticket module widget for Intalio Customer
Panel.</Description>
        <Navigation>CustomerFrontend</Navigation>
        <Value>
            <Hash>
                <!--JSON configuration file-->
                <Item ValueType="Textarea" Key="schema">
                    {
                        "props": {
                        },
                        "id": "1000",
                        "widgetFile": "WIDGET_XYZ.js",
                        "compiled": false,
                        "view" : "TicketPreview",
                        "slot" : "slot4",
                        "name": "XYZ"
                    }
                </Item>
            </Hash>
        </Value>
    </Setting>
```
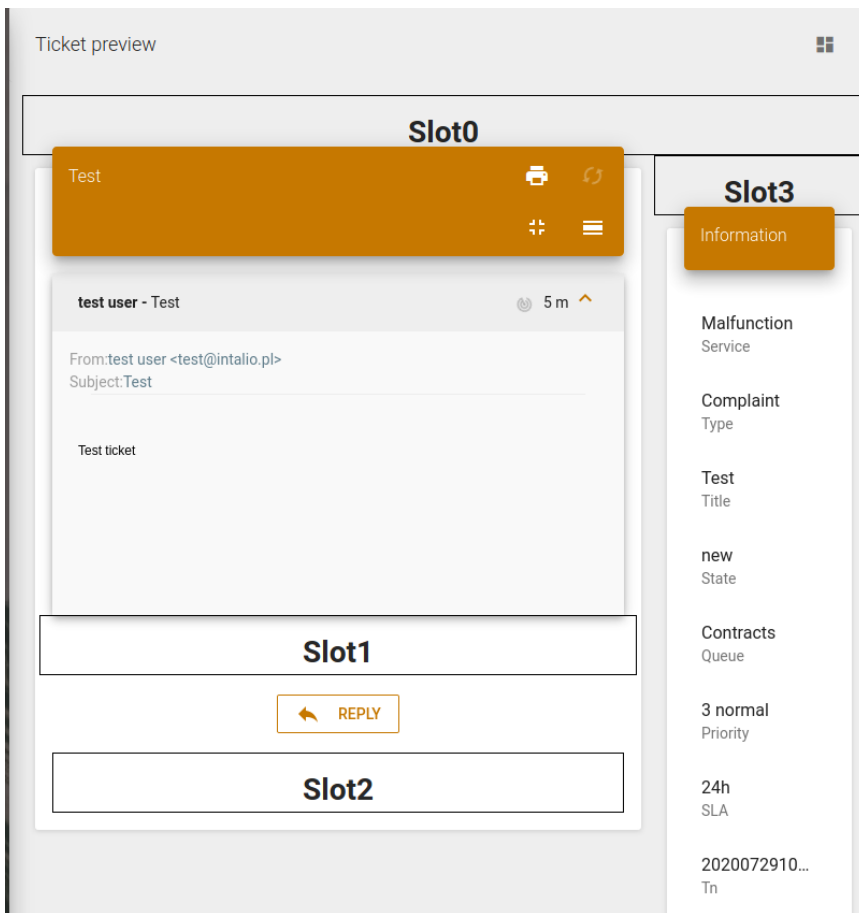
```
</otrs_config>
```

Schema

The schema file is a JSON file. It specifies the most important aspects of the whole widget.

```
{
  "props": {
    "predefined_property": true
  },
  "id": "1000",
  "widgetFile": "WIDGET_Contract.js",
  "compiled": false,
  "view" : "TicketPreview",
  "slot" : "slot4",
  "name": "Contracts"
}
```

- *<props>*
  Specify predefined properties that could be referenced in the template file
- *<id>*
  An **identification key** for your widget. Should be unique for every module**.** Id is responsible for creating unique widget settings for every user.
- *<widgetFile>*

  Template file's name - see more on the widget's template file.
- *<view>*
  Specify the primary module where should widget be located. You can choose between multiple options starting, from predefined modules ending up to your own custom

  modules. You can find a more detailed explanation on the administration section,
- *<compiled>*
  ????????,
- *<slot>*
  This label specifies a place on the layout where should widget be put in the module.

> **<Slot>** property is required while creating a widget in the TicketPreview module.

> You can display slot layout by adding query parameter - "dev=1" to the URL parameters eg. '/customer-panel/ticketPreview/000014/?dev=1'

CustomRedirectQuery

Since `v1.1.41` in the configuration of the widget responsible for the list of tickets, you can define additional parameters passed to the TicketPreview module. If you add a new parameter in `props => defaultConfig => customRedirectQuery` you can easily manage them in the TicketPreview module eg.

```
[
  {
    "id" : "8",
    "widgetFile" : "ticket-list",
    "name" : "Tickets: Example queue",
    "view" : "module",
    "compiled" : true,
    "props" : {
      "defaultConfig" : {
        "customRedirectQuery" : {
          "queue" : "ExampleQueryParam"
        }
      }
    }
  }
```

```
    ]
```

In this example clicking on the specific ticket in the ticket list will result with URL like this:

```
https://<OTRS_URL>/otrs/route.pl/customer-
panel/ticketPreview/<TICKET_NUMBER>?queue=ExampleQueryParam
```

Template

Template file

The template file is the backbone of the whole widget. It is built on the simple JavaScript function returning vue Instance object. Each Vue instance has a unique id. It serves this purpose for easy saving of the user settings, user's data, and preferences. In the runtime "CustomModuleName_DynamicID" is being replaced with proper widget's id for further actions.

```
(function (internalName) {
    var internalID = internalName;
    return {
        moduleName: internalID,
        component: Vue.component(internalID, {
            props: [],
            data:,
            mounted() {},
            activated() {},
            deactivated() {},
            methods: {},
            computed: {},
            template: ``
        }),
    }

}("CustomModuleName_DynamicID"));
```

Working with database

Every Customer Panel launch system checks if the configuration for a specific customer already exists in the database. If it doesn't exist, it is being created, and then return to the frontend, or in the case, it already exists, simply return.

The system creates the configuration with the following pattern:

- <module_name>_<module_id>_<user_login>_hkrOUgHKUi_<invoking_module_name>_Settings

- <module_name>_<module_id>_<user_login>_hkrOUgHKUi_<invoking_module_name>_Data

Saving with Settings prefix is used for storing widget's configuration taken from the global widget's settings.

Saving with Data prefix is used for storing user's saved data. [settings configuration/user's actions/notes]

Set

Setting a user's data can be resolved with the following endpoint:

```
POST <OTRS_URL>/customer.pl?Action=CustomerFrontend;Subaction=SetWidgetData;ID=<moduleName>
```

```
sendNote: _.debounce(function () {
        axios.post(this.$store.getters["globalConfig/OTRS_URL"] +
"customer.pl?Action=CustomerFrontend;Subaction=SetWidgetData;ID=" + this.moduleName, {
            note: this.note
        }).then((response) => {
            console.log("Send note!")
        }).catch((error) => {
            console.log("ERROR [setWidgetConfig] ", error)
        });
    }, 300),
```

Get

Getting a user's data can be resolved with the following endpoints:

```
GET <OTRS_URL>/customer.pl?Action=CustomerFrontend;Subaction=GetWidgetData;ID=<moduleName>
```

```
getNote() {
        axios.get(this.$store.getters["globalConfig/OTRS_URL"] +
"customer.pl?Action=CustomerFrontend;Subaction=GetWidgetData;ID=" +
this.moduleName).then((response) => {
            if (response.data.success) {
                this.note = response.data.data.note
            }})}}
```

> These operations easily describe the whole idea of the template mechanism.

Global Storage

Global storage contains the most important information about OTRS and current CustomerPanel instance.

You can refer to its values by the keyword `this.$store` and retrieve properties with `getters` . Example of useful values:

```
[
 "account/getSessionId": "lkDT71THevxijoiKprX9LoHmehkoWu4emCa",
 "globalConfig/BACKEND_VERSION": "1.2.54",
 "globalConfig/FRONTEND_VERSION": "1.1.40",
 "globalConfig/OTRS_DEFAULT_LANG": "en",
 "globalConfig/OTRS_FQDN": "http://customotrsinstance.com",
 "globalConfig/OTRS_ScriptAlias": "/otrs/",
 "globalConfig/OTRS_URL": "http://customotrsinstance.com/otrs/",
 ]
```

These are very useful while working on your widgets. They allow you to connect to OTRS API our display information about currently previewed tickets.

```
getContractInfo() {
  axios.get(this.$store.getters["globalConfig/OTRS_URL"] +
`customer.pl?Action=ContractWidget;Subaction=GetContractInfo;Tn=${this.$route.params.ticketNum
ber}`).then((response) => {
      if (response.data.success === false) {
          this.contract_info = null
          this.contractLoading = false
      } else {
          this.contract_info = response.data.information
          this.contractLoading = false
      }
   })
}
```

Accessing for nested objects in global storage can be easily realized by referencing other elements eg. `this.$store.getters["ticketPreview/ticketArticles"][0]['Age']`

> You can display whole global storage content by simply printing this.$store.getters in the console.

Language support

To use predefined language strings, depending on the user's default language settings you can simply reference them in the template file like this:

```
widgetTitle() {

    return this.$t('Widget name')

}
```

or directly in the HTML template:

```
<span> {{$t('Contract loading')}}</span>
```

Multilanguage translation support

To create a translation for the template file we have to create a separate file:

- XYZ_translations_strings.json
- xyzWidget.pm

In the installation process, XYZ_translations_strings.json is being merged to the global translations list. Depends on the user preferences strings a system iterate over the right prefix and look for appropriate translations. Look on the example below:

- XYZ_translations_strings.json

```
[
"Contract number",
"Contract name",
"Remaining time",
"Support start date",
"Support end date",
"Number of hours in the support",
"Service information",
"left",
"exceeded",
"h",
"No service assigned",
"Contract loading..."
]
```

- pl_xyzWidget.pm

> Be careful correctly type the module path!

```
package Kernel::Language::pl_xyzWidget;


use strict;
```

```perl
use warnings;
use utf8;
use vars qw(@ISA $VERSION);


sub Data {
    my $Self = shift;


    $Self->{Translation}->{"Contract loading..."} = "Ładowanie kontraktu...";
    $Self->{Translation}->{"Contract number"} = "Numer kontraktu";
    $Self->{Translation}->{"Contract name"} = "Nazwa kontraktu";
    $Self->{Translation}->{"Remaining time"} = "Pozostały czas";
    $Self->{Translation}->{"Support start date"} = "Początek wsparcia";
    $Self->{Translation}->{"Support end date"} = "Koniec wsparcia";
    $Self->{Translation}->{"Number of hours in the support"} = "Godziny w ramach wsparcia";
    $Self->{Translation}->{"Service information"} = "Informacje o usłudze";
    $Self->{Translation}->{"left"} = "pozostało";
    $Self->{Translation}->{"exceeded"} = "przekroczono";
    $Self->{Translation}->{"h"} = "godz.";
    $Self->{Translation}->{"No service assigned"} = "Brak przypisanej usługi";


    return 1;
}
1;
```

Installation file .sopm

Installation file

The installation process contains a standard OTRS package installation file.

Basic installation file:

> Basic installation doesn't differ much from the standard OTRS module installation if you do not plan to enable a multilanguage support mechanism.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<otrs_package version="1.0">
    <Name>Contract Widget</Name>
    <Version>1.x.x</Version>
    <Framework>6.x.x</Framework>
    <Vendor>Company</Vendor>
    <URL>https://www.xyz.pl/</URL>
    <License>GNU AFFERO GENERAL PUBLIC LICENSE Version 3, November 2007</License>
    <Description Lang="en">XYZ Widget for Intalio Customer Panel.</Description>
    <IntroInstall Type="post" Lang="en" Title="Thank you!">Thank you for choosing the XYZ
Widget for Intalio Customer Panel module.</IntroInstall>
    <BuildDate>?</BuildDate>
    <BuildHost>?</BuildHost>

    <Filelist>
        <File Permission="644" Location="Kernel/Config/Files/XML/XYZWidget.xml"></File>
        <File Permission="644" Location="Kernel/Modules/XYZWidget.pm"></File>
        <File Permission="644" Location="Kernel/Language/pl_XYZ.pm"></File>

        <!-- FRONTEND -->
        <File Permission="644" Location="var/httpd/htdocs/customer-
panel/modules/WIDGET_XYZ.js"></File>

        <!-- TRANSLATION STRINGS -->
```

```
        <File Permission="644" Location="var/intalio-customer-
panel/xyz_translation_strings.json"/>
    </Filelist>
</otrs_package>
```

Multilanguage support:

Enabling additional language support requires adding additional translation strings to the global list. Include this code below in the installation file.

> Script will look after every json file containing substring translation_strings.json!

The scipt workflow looks like that:

```
opening main translation_strings.json => looking for module's translations files => iterating over
list and appending translations which are not already in the main translation_strings.json
```

The translations are then correctly retreived in the Customer Panel.

```
<CodeInstall Type="post"><![CDATA[


        ###
        # Adding new module's translations to the global phrase list of the otrs-frontend
module
        ###
        use JSON::PP;
        use Data::Dumper;


        my $TranslationStringsDir = '/opt/otrs/var/intalio-customer-panel/';


        my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
        my $json = JSON::PP->new->pretty->allow_nonref;
        $json = $json->allow_blessed;
        $json = $json->allow_unknown;
        $json = $json->convert_blessed;


        my $MainTranslationStringsRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-panel/translation_strings.json',
        );


        my @MainTranslationStrings = @{$json->decode( ${$MainTranslationStringsRef} )};
```

```perl
        opendir(DH, $TranslationStringsDir);
        my @TranslationStringsFiles = readdir(DH);
        closedir(DH);

        foreach my $TranslationStringsFile (@TranslationStringsFiles) {
            # skip . and .. .. other than .json and other than main translation strings file
            next if($TranslationStringsFile =~ /^\.$/);
            next if($TranslationStringsFile =~ /^\.\.$/);
            next if(not $TranslationStringsFile =~ /\.json$/);
            next if($TranslationStringsFile =~ /^translation_strings\.json$/);

            print STDERR "------------------------------------\n";
            print STDERR "Opening and decoding file: $TranslationStringsFile\n";
            my $TranslationStringsRef = $MainObject->FileRead(
                Location => $TranslationStringsDir . $TranslationStringsFile,
            );

            my @TranslationStrings = @{$json->decode( ${$TranslationStringsRef} )};

            foreach my $TranslationString (@TranslationStrings) {
                my $IsTranslationStringAlreadyInMainTranslationStrings = 0;

                foreach my $MainTranslationString(@MainTranslationStrings) {
                    $IsTranslationStringAlreadyInMainTranslationStrings = 1 if
($TranslationString eq $MainTranslationString);
                }

                if (not $IsTranslationStringAlreadyInMainTranslationStrings) {
                    print STDERR "Adding string: $TranslationString\n";
                    push(@MainTranslationStrings, $TranslationString);
                } else {
                    print STDERR "String already in main translation strings:
$TranslationString\n";
                }
            }
            print STDERR "------------------------------------\n\n";
        }

        print STDERR "Final Main Translation Strings\n";
```

```perl
        foreach my $MainTranslationString(@MainTranslationStrings) {
            print STDERR $MainTranslationString ."\n";
        }


        my $MainTranslationStringsResult = $json->encode( \@MainTranslationStrings );


        $MainObject->FileWrite(
            Directory => '/opt/otrs/var/intalio-customer-panel/',
            Filename  => "translation_strings.json",
            Content   => \$MainTranslationStringsResult,
        );


    ]]></CodeInstall>


    <CodeReinstall Type="post"><![CDATA[


        ###
        # Adding new module's translations to the global phrase list of the otrs-frontend
module
        ###
        use JSON::PP;
        use Data::Dumper;


        my $TranslationStringsDir = '/opt/otrs/var/intalio-customer-panel/';


        my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
        my $json = JSON::PP->new->pretty->allow_nonref;
        $json = $json->allow_blessed;
        $json = $json->allow_unknown;
        $json = $json->convert_blessed;


        my $MainTranslationStringsRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-panel/translation_strings.json',
        );


        my @MainTranslationStrings = @{$json->decode( ${$MainTranslationStringsRef} )};


        opendir(DH, $TranslationStringsDir);
        my @TranslationStringsFiles = readdir(DH);
        closedir(DH);
```

```perl
        foreach my $TranslationStringsFile (@TranslationStringsFiles) {
            # skip . and .. .. other than .json and other than main translation strings file
            next if($TranslationStringsFile =~ /^\.$/);
            next if($TranslationStringsFile =~ /^\.\.$/);
            next if(not $TranslationStringsFile =~ /\.json$/);
            next if($TranslationStringsFile =~ /^translation_strings\.json$/);

            print STDERR "------------------------------------\n";
            print STDERR "Opening and decoding file: $TranslationStringsFile\n";
            my $TranslationStringsRef = $MainObject->FileRead(
                Location => $TranslationStringsDir . $TranslationStringsFile,
            );

            my @TranslationStrings = @{$json->decode( ${$TranslationStringsRef} )};

            foreach my $TranslationString (@TranslationStrings) {
                my $IsTranslationStringAlreadyInMainTranslationStrings = 0;

                foreach my $MainTranslationString(@MainTranslationStrings) {
                    $IsTranslationStringAlreadyInMainTranslationStrings = 1 if
($TranslationString eq $MainTranslationString);
                }

                if (not $IsTranslationStringAlreadyInMainTranslationStrings) {
                    print STDERR "Adding string: $TranslationString\n";
                    push(@MainTranslationStrings, $TranslationString);
                } else {
                    print STDERR "String already in main translation strings:
$TranslationString\n";
                }
            }
            print STDERR "------------------------------------\n\n";
        }

        print STDERR "Final Main Translation Strings\n";
        foreach my $MainTranslationString(@MainTranslationStrings) {
            print STDERR $MainTranslationString ."\n";
        }
```

```perl
        my $MainTranslationStringsResult = $json->encode( \@MainTranslationStrings );

        $MainObject->FileWrite(
            Directory => '/opt/otrs/var/intalio-customer-panel/',
            Filename  => "translation_strings.json",
            Content   => \$MainTranslationStringsResult,
        );
    ]]></CodeReinstall>


    <CodeUpgrade Type="post"><![CDATA[
        ###
        # Adding new module's translations to the global phrase list of the otrs-frontend
module
        ###
        use JSON::PP;
        use Data::Dumper;

        my $TranslationStringsDir = '/opt/otrs/var/intalio-customer-panel/';

        my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
        my $json = JSON::PP->new->pretty->allow_nonref;
        $json = $json->allow_blessed;
        $json = $json->allow_unknown;
        $json = $json->convert_blessed;

        my $MainTranslationStringsRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-panel/translation_strings.json',
        );

        my @MainTranslationStrings = @{$json->decode( ${$MainTranslationStringsRef} )};

        opendir(DH, $TranslationStringsDir);
        my @TranslationStringsFiles = readdir(DH);
        closedir(DH);

        foreach my $TranslationStringsFile (@TranslationStringsFiles) {
            # skip . and .. .. other than .json and other than main translation strings file
            next if($TranslationStringsFile =~ /^\.$/);
            next if($TranslationStringsFile =~ /^\.\.$/);
            next if(not $TranslationStringsFile =~ /\.json$/);
```

```perl
            next if($TranslationStringsFile =~ /^translation_strings\.json$/);

            print STDERR "-------------------------------------\n";
            print STDERR "Opening and decoding file: $TranslationStringsFile\n";
            my $TranslationStringsRef = $MainObject->FileRead(
                Location => $TranslationStringsDir . $TranslationStringsFile,
            );

            my @TranslationStrings = @{$json->decode( ${$TranslationStringsRef} )};

            foreach my $TranslationString (@TranslationStrings) {
                my $IsTranslationStringAlreadyInMainTranslationStrings = 0;

                foreach my $MainTranslationString(@MainTranslationStrings) {
                    $IsTranslationStringAlreadyInMainTranslationStrings = 1 if
($TranslationString eq $MainTranslationString);
                }

                if (not $IsTranslationStringAlreadyInMainTranslationStrings) {
                    print STDERR "Adding string: $TranslationString\n";
                    push(@MainTranslationStrings, $TranslationString);
                } else {
                    print STDERR "String already in main translation strings:
$TranslationString\n";
                }
            }
            print STDERR "-------------------------------------\n\n";
        }

        print STDERR "Final Main Translation Strings\n";
        foreach my $MainTranslationString(@MainTranslationStrings) {
            print STDERR $MainTranslationString ."\n";
        }

        my $MainTranslationStringsResult = $json->encode( \@MainTranslationStrings );

        $MainObject->FileWrite(
            Directory => '/opt/otrs/var/intalio-customer-panel/',
            Filename  => "translation_strings.json",
            Content   => \$MainTranslationStringsResult,
```

```
        );
]]></CodeUpgrade>
```

Writing a first customer-panel widget

Main assumptions

Our goal is to create a simple widget that can be added to the Dashboard view. It will include the number of new tickets assigned to the specific customer user.

Module Structure

The widget will be built as a standard OTRS package, easy to build and deploy. It will contain all necessary configuration needed for proper running. We will create a separate hidden module in which we will put our widget. Our whole package's structure will look like this:

```
otrs-ticket-information/
├── install.sopm
├── Kernel
│   ├── Config
│   │   └── Files
│   │       └── XML
│   │           └── TicketInformationWidget.xml
│   ├── Language
│   │   └── pl_TicketInformationWidget.pm
│   └── Modules
│       └── TicketInformation.pm
└── var
    ├── httpd
    │   └── htdocs
    │       └── customer-panel
    │           └── modules
    │               └── WIDGET_NewTickets.js
    └── intalio-customer-panel
        └── ticket_widget_translation_strings.json
```

XML Configuration

What we do here is registering the OTRS module - TIcketInformation, which will be referenced when calling our API for retrieving the ticket's information. Furthermore, we register our CustomerPanel module and assign 'Open tickets counter' to him.

```
<otrs_config version="2.0" init="Application">
    <Setting Name="CustomerFrontend::Module###ContractWidget" Required="1" Valid="1">
        <Description Translatable="1">FrontendModuleRegistration for TicketInformation
module.</Description>
        <Navigation>Frontend::Customer::ModuleRegistration</Navigation>
        <Value>
            <Item ValueType="FrontendRegistration">
                <Hash>
                    <Item Key="Group">
                        <Array>
                            <Item>users</Item>
                        </Array>
                    </Item>
                    <Item Key="Description" Translatable="1">TicketInformation.</Item>
                    <Item Key="Title" Translatable="1">TicketInformation</Item>
                    <Item Key="NavBarName">TicketInformation</Item>
                </Hash>
            </Item>
        </Value>
    </Setting>
    <Setting Name="OTRSFrontendModule::TicketInformation" Required="0" Valid="1">
        <Description Translatable="1">This is module for a small ticket's informations
 widgets</Description>
        <Navigation>CustomerFrontend</Navigation>
        <Value>
            <Hash>
                <Item Key="id">TicketInformation</Item>
                <Item Key="type">hidden</Item>
                <Item Key="text">Contract</Item>
                <Item Key="to">/module/TicketInformation</Item>
                <Item Key="priority">110</Item>
                <Item ValueType="Textarea" Key="widgets">
                    [
                        {
                            "props": {},
                            "id": "20",
```

```
                        "widgetFile": "WIDGET_NewTickets.js",

                        "compiled": false,

                        "view" : "Dashboard",

                        "name": "Open tickets counter"

                    }

                ]

            </Item>

        </Hash>

    </Value>

</Setting>

</otrs_config>
```

OTRS Module

In the previous subthread, we defined OTRS module ['TicketInformation'] which will be our bridge to OTRS API. We will define there whole logic connecting to the database and returning it to the web caller. In this case, we will retrieve number of new tickets.

```perl
package Kernel::Modules::TicketInformation;


use strict;
use warnings;
use utf8;
use JSON::PP;
use Data::Dumper;
use Kernel::System::VariableCheck qw(:all);


our $ObjectManagerDisabled = 1;


sub new {
    my ( $Type, %Param ) = @_;


    # allocate new hash for object
    my $Self = {%Param};
    bless( $Self, $Type );
    $Self->{ParamObject} = $Kernel::OM->Get('Kernel::System::Web::Request');


    return $Self;
}


sub Run {
```

```perl
my ( $Self, %Param ) = @_;

if($Self->{Subaction} eq 'GetNewTicketsCount') {

        my $UserLogin = $Self->{UserLogin};

        my $ResultJSON = {
            success => 0,
            message => "An unknown error occurred. Please contact the administrator.",
        };


        my $DBObject = $Kernel::OM->Get('Kernel::System::DB');

        $DBObject->Prepare(
            SQL => "SELECT
                        COUNT(t.id)
                    FROM
                        ticket t
                    JOIN
                        ticket_state ts ON ts.id = t.ticket_state_id
                    JOIN
                        ticket_state_type tst ON tst.id = ts.type_id
                    WHERE
                        customer_user_id = ?
                    AND
                        tst.id = 1",
            Bind => [\$Param{UserLogin}],
        );

        while( my @Row = $DBObject->FetchrowArray()) {
            $ResultJSON->{data} = $Row[0];
        }

        if (exists $ResultJSON->{data}) {
          $ResultJSON->{success} = \1;
          $ResultJSON->{message} = "OK";
        }

        return $ResultJSON;
```

```
        }
    }
    1;
```

You can read more on creating frontend modules in [the official OTRS developer guide](#). Basically, based on the passed UserLogin we retrieve tickets with the 'new' state and return a simple JSON response.

Template File

After adding backend logic we are ready to create our template file. This contains a simple syntax built on vue instance.

```
(function (internalName) {
    var internalID = internalName;
    return {
        name: internalID,
        component: Vue.component(internalID, {
            props: [],
            data: function () {
                return {
                    moduleName: internalID,
                    count: "0",
                    lastUpdate: "",
                }
            },
            mounted() {
                console.log("WIDGET_NumberOfNewTickets loaded!")
                axios.get(this.$store.getters["globalConfig/OTRS_URL"] +
 `customer.pl?Action=TicketInformation;Subaction=GetNewTicketsCount`).then((response) => {
                    this.count = response.data.data.toString()
                })
                this.lastUpdate = new Date().toLocaleTimeString().slice(0,5)
            },
            activated() {
            },
            deactivated() {
            },
            template: `
                <v-flex xs12 md3>
                    <material-stats-card
```

```
                                  style="margin-top:0px !important;"
                                  color="success"
                                  icon="new_releases"
                                  :title="$t('Number of new tickets')"
                                  :value="count"
                                  sub-icon="calendar_today"
                                  sub-icon-color="secondary"
                                  :sub-text="$t('Last update')+' '+lastUpdate"
                                  sub-text-color="text-primary"
                        />
                </v-flex>
          `

        }),
    }
}("CustomModuleName_DynamicID"));
```

What we do here is returning **vue instance** object with its own ID, which is assigned to every CustomerUser. It allows us to easily manipulate and save the user's own personal settings (If we provide such options).

Here, we used a simple call to our earlier defined frontend module and retrieve the new ticket number. Then, we displayed it in the material-stats-card template as a simple card.

Language support

Note, that we used here a mechanism to include a multilanguage translation. This will depend on the user's default language's setting. We will create now two different files needed to be included in the module to properly translate your labels.

ticket_widget_translation_strings.json

```
[
 "Number of new tickets",
 "Last update"
]
```

pl_TicketInformationWidget.pm

```
package Kernel::Language::pl_TicketInformationWidget;

use strict;
use warnings;
```

```perl
use utf8;

use vars qw(@ISA $VERSION);

sub Data {
    my $Self = shift;

    $Self->{Translation}->{"Number of new tickets"} = "Liczba nowych zgłoszeń";
    $Self->{Translation}->{"Last update"} = "Ostatnia aktualizacja";


    return 1;
}
1;
```

> Note you can use every language you need - it's completely up to you!

Installation file

Now on the top of our project, we create installation file - **install.sopm**. It will be responsible for building our .opm package and merging our translation strings to the global list.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<otrs_package version="1.0">
    <Name>Ticket Information</Name>
    <Version>1.0.0</Version>
    <Framework>6.0.x</Framework>
    <Vendor>CustomCompany</Vendor>
    <URL>https://www.custom_company.pl/</URL>
    <License>GNU AFFERO GENERAL PUBLIC LICENSE Version 3, November 2007</License>
    <Description Lang="en">Module for Customer Panel.</Description>
    <IntroInstall Type="post" Lang="en" Title="Thank you!">Thank you for choosing the Customer
Widget.</IntroInstall>
    <BuildDate>?</BuildDate>
    <BuildHost>?</BuildHost>

     <Filelist>
        <File Permission="644"
Location="Kernel/Config/Files/XML/TicketInformation.xml"></File>
        <File Permission="644" Location="Kernel/Modules/TicketInformation.pm"></File>
        <File Permission="644"
```

```
Location="Kernel/Language/pl_TicketInformationWidget.pm"></File>

        <!-- FRONTEND -->
        <File Permission="644" Location="var/httpd/htdocs/customer-
panel/modules/WIDGET_OpenTickets.js"></File>

        <!-- TRANSLATION STRINGS -->
        <File Permission="644" Location="var/intalio-customer-
panel/ticket_widget_translation_strings.json"/>
    </Filelist>

    <CodeInstall Type="post"><![CDATA[
        use JSON::PP;
        use Data::Dumper;
        my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
        my $GlobalTranslationStringsStrRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-panel/translation_strings.json',
        );
        my $TranslationStrings = ${$GlobalTranslationStringsStrRef};

        my $json = JSON::PP->new->pretty->allow_nonref;
        $json = $json->allow_blessed;
        $json = $json->allow_unknown;
        $json = $json->convert_blessed;

        my @RequiredStrings = @{$json->decode( $TranslationStrings )};

        my $MyTranslationStringsStrRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-
panel/ticket_widget_translation_strings.json',
        );
        my $MyTranslationStrings = ${$MyTranslationStringsStrRef};
        my @TranslationStrings = @{$json->decode( $MyTranslationStrings )};

        print STDERR "RequiredStrings: ".Dumper(\@RequiredStrings);
        print STDERR "TranslationStrings: ".Dumper(\@TranslationStrings);
        @RequiredStrings = (@RequiredStrings, @TranslationStrings);

        my $TranslationStringsResult = $json->encode( \@RequiredStrings );
        print STDERR "Result: $TranslationStringsResult\n";
```

```perl
        my $FileLocation = $MainObject->FileWrite(
            Directory => '/opt/otrs/var/intalio-customer-panel/',
            Filename  => "translation_strings.json",
            Content   => \$TranslationStringsResult,
        );
    ]]></CodeInstall>


    <CodeReinstall Type="post"><![CDATA[
        use JSON::PP;
        use Data::Dumper;
        my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
        my $GlobalTranslationStringsStrRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-panel/translation_strings.json',
        );
        my $TranslationStrings = ${$GlobalTranslationStringsStrRef};


        my $json = JSON::PP->new->pretty->allow_nonref;
        $json = $json->allow_blessed;
        $json = $json->allow_unknown;
        $json = $json->convert_blessed;


        my @RequiredStrings = @{$json->decode( $TranslationStrings )};


        my $MyTranslationStringsStrRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-
panel/ticket_widget_translation_strings.json',
        );
        my $MyTranslationStrings = ${$MyTranslationStringsStrRef};
        my @TranslationStrings = @{$json->decode( $MyTranslationStrings )};


        print STDERR "RequiredStrings: ".Dumper(\@RequiredStrings);
        print STDERR "TranslationStrings: ".Dumper(\@TranslationStrings);
        @RequiredStrings = (@RequiredStrings, @TranslationStrings);


        my $TranslationStringsResult = $json->encode( \@RequiredStrings );
        print STDERR "Result: $TranslationStringsResult\n";


        my $FileLocation = $MainObject->FileWrite(
            Directory => '/opt/otrs/var/intalio-customer-panel/',
```

```
                Filename  => "translation_strings.json",
                Content   => \$TranslationStringsResult,
            );
    ]]></CodeReinstall>


    <CodeUpgrade Type="post"><![CDATA[
        ###
        # Dodanie nowych tłumaczeń tego modułu do globalnej listy fraz modułu otrs-frontend.
        ###
        use JSON::PP;
        use Data::Dumper;
        my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
        my $GlobalTranslationStringsStrRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-panel/translation_strings.json',
        );
        my $TranslationStrings = ${$GlobalTranslationStringsStrRef};


        my $json = JSON::PP->new->pretty->allow_nonref;
        $json = $json->allow_blessed;
        $json = $json->allow_unknown;
        $json = $json->convert_blessed;


        my @RequiredStrings = @{$json->decode( $TranslationStrings )};


        my $MyTranslationStringsStrRef = $MainObject->FileRead(
            Location => '/opt/otrs/var/intalio-customer-
panel/ticket_widget_translation_strings.json',
        );
        my $MyTranslationStrings = ${$MyTranslationStringsStrRef};
        my @TranslationStrings = @{$json->decode( $MyTranslationStrings )};


        print STDERR "RequiredStrings: ".Dumper(\@RequiredStrings);
        print STDERR "TranslationStrings: ".Dumper(\@TranslationStrings);
        @RequiredStrings = (@RequiredStrings, @TranslationStrings);


        my $TranslationStringsResult = $json->encode( \@RequiredStrings );
        print STDERR "Result: $TranslationStringsResult\n";


        my $FileLocation = $MainObject->FileWrite(
            Directory => '/opt/otrs/var/intalio-customer-panel/',
```

```
          Filename  => "translation_strings.json",
          Content   => \$TranslationStringsResult,
      );
    ]]></CodeUpgrade>


  </otrs_package>
```

OTRS Installation

Now we can build our package and deploy it to the system.

```
otrs> /opt/otrs/bin/otrs.Console.pl Dev::Package::Build --module-directory ./ install.sopm ./
```

The following command will generate complete installation package with .**opm** file extension

Now exerything left to do is to install generated package with one of the provided methods:

The following instruction explains how to install the package using one of the provided methods.

1. Admin Interface

Log in to your system as user with admin privileges and go to menu Admin ⇒ Package Manager. Select module file (with .opm extension) in the "Actions" panel and click "Install Package" button.

2. Command line

```
otrs> /opt/otrs/bin/otrs.Console.pl Admin::Package::Install /path/to/package/.opm
```

Summary

As You might see writiring your own widget is pretty straightforward and easy. It enables You to extend your CustomerPanel with powerful tools that highly depends on your needs.