Sviluppi SDK lato Portale

Sviluppi SDK lato Portale

Introduzione

Le seguenti specifiche indicano come eseguire implementazioni SDK lato portale con il fine di personalizzare l'aspetto e le funzionalità in maniera semplice e veloce.

Requisiti:

• VTE 18.12

Indice

1. Estendere Webservice SOAP

Premessa: L'estensione dei webservice SOAP si rende possibile tramite metodi della classe "SOAPWSManager.php".

I webservice di tipo SOAP utilizzati lato portale vengono ora salvati a database e possono essere aggiunti dinamicamente richiamando il seguente metodo e indicandone i parametri necessari:

addWebservice(ws_name, path, class, return_type, parameters)

Es.

```
$SWSMan = SOAPWSManager::getInstance();
$SWSMan->addWebservice (
	'get_field_structure',
	'modules/SDK/src/your_folder/your_file.php',
	'file_class',
	'return_parameters_type',
	array (
```

```
    array ('name' => 'module', 'type' => 'xsd:string'),
    array ('name' => 'id', 'type' => 'xsd:string'),
    array ('name' => 'language', 'type' => 'xsd:string'),
  )
);
```

Rimuovere webservice SOAP

Allo stesso modo, è possibile rimuovere un webservice già esistente richiamando il seguente metodo:
removeWebservice(ws_name)

Es.

```
$SWSMan = SOAPWSManager::getInstance();
$SWSMan->removeWebservice('get_field_structure');
```

2. Modificare Webservice SOAP esistenti

Per modificare i webservice SOAP standard di VTE è possibile estendere la relativa classe "SOAPWebservices" contenuta in "VTE_URL/soap/SOAPWebservices.php" e modificare i metodi esistenti come descritto di seguito:

SDK::setClass('ws_class', 'new_ws_class', 'path')

Es.

```
SDK::setClass('SOAPWebservices', 'NewSOAPWS', 'modules/SDK/src/your_folder/your_file_SOAPWS.php')
```

Ad esempio, per modificare il webservice create_ticket, si può procedere come segue:

```
class NewSOAPWS extends SOAPWebservices {

    function create_ticket($input_array) {
    // prepend "PORTAL:" to the ticket title
        $input_array['title'] = 'PORTAL: '.$input_array['title'];
        return parent::create_ticket($input_array);
     }


}
```

3. Modificare/sovrascrivere Template Smarty

Per modificare/sovrascrivere i template smarty lato portale basta personalizzare l'array "sdk_config" contenuto in:
"portal/SDK/config.php".
All'interno si definiscono i nomi dei nuovi template/fogli di stile. In questo modo, non saranno più richiamati i vecchi template standard ma i nuovi creati appositamente.

Indicare quindi queste informazioni come descritto di seguito:

```
$sdk_config = array (
 
    'templates' => array(
    // 'old_template' => 'new_template',
        'login.tpl' => 'customer-login.tpl',
    ),


);
```

4. Modificare le Etichette del portale

Per modificare le etichette (labels) lato portale basta personalizzare l'array "sdk_config" contenuto in:
"portal/SDK/config.php".
All'interno si definiscono i percorsi dei file, contenenti le varie labels (label originale e traduzione relativa) in base alle lingue installate nel crm.

N.B. Le lingue SDK vengono caricate successivamente le lingue standard, pertanto è possibile sovrascrivere le labels relative quest'ultime.

Indicare quindi queste informazioni come descritto di seguito:

```
$sdk_config = array(
'languages' => array(
    // add new labels or change existing ones
        'it_it' => array(
        'languages/customer-name.it_it.php',
                    // other files...
         ),
         'en_us' => array(
         'languages/customer-name.en_us.php',
                    // other files...
         ),
         // other languages...
    ),
```

```
);
```

5.Inserire Codice js per validare/modificare campi

Per inserire del codice js lato portale dedicato alla validazione di alcuni campi (es. validazione Codice Fiscale, codice IBAN, ...) o modifica degli stessi (es. prima lettera maiuscola, controllo caratteri speciali, ...) basta personalizzare l'array "sdk_config" contenuto in:
"portal/SDK/config.php".
All'interno si definiscono i percorsi dei file da includere, contenenti le funzioni necessarie per eseguire tali controlli.

Indicare quindi queste informazioni come descritto di seguito:

```
$sdk_config = array(
    'global_js' => array(
        // js to be loaded globally
        'js/my-library.js',
        'js/customer-name.js',
    ),

    'module_js' => array(
        // js to be loaded for specific module
        'HelpDesk' => array(
            'js/tickets.js',
        ),
        // other modules...
    ),

);
```

6. Includere file PHP globali

Per poter utilizzare classi/metodi appropriatamente sviluppati è possibile includere dei file php globali personalizzando l'array "sdk_config" contenuto in:
"portal/SDK/config.php".
All'interno si definiscono i file da includere, contenenti i metodi necessari.

Indicare quindi queste informazioni come descritto di seguito:

```
$sdk_config = array(
```

```
'global_php' => array(
    // php files to be loaded in every page (should contain classes/functions)
        'customer-name.php',
        'MyUtilsFile.php',
     ),


);
```

7. Includere file CSS globali

Per poter utilizzare dei CSS personalizzati è possibile includere dei file css globali personalizzando l'array "sdk_config" contenuto in:
"portal/SDK/config.php".
All'interno si definiscono i percorsi dei file da includere, contenenti gli stili definiti.

N.B. Questi file css globali, essendo caricati successivamente quelli standard, possono sovrascrivere i css relativi quest'ultimi.

Indicare quindi queste informazioni come descritto di seguito:

```
$sdk_config = array (


'global_css' => array(
    // css to be loaded globally
        'css/customer-name.css',
     ),


);
```