

Reports personalizzati

Si possono inserire cartelle e report personalizzati usando i seguenti metodi

SDK::setReportFolder(\$name, \$description);

\$name : nome della cartella

\$description : descrizione

Crea una nuova cartella nella pagina dei report. La cartella creata sarà visibile da tutti gli utenti e non può essere modificata. Il nome e la descrizione possono essere tradotti nel solito modo. Le cartelle così create si possono eliminare con

SDK::unsetReportFolder(\$name, \$delreports = true);

\$name : il nome della cartella da cancellare

\$delreports : se true elimina anche tutti i reports (creati tramite SDK) in quella cartella (non vengono eliminati files)

All'interno delle cartelle create via SDK si possono inserire reports personalizzati con:

SDK::setReport(\$name, \$description, \$foldername, \$reportrun, \$class, \$jsfunction = '');

\$name : il nome del report (che si può tradurre come sempre)

\$description : descrizione del report (traducibile)

\$foldername : il nome della cartella (SDK) in cui inserire il report

\$reportrun : il percorso del file php che contiene la classe che genera il report

\$class : il nome della classe che gestisce il report (dettagli dopo)

\$jsfunction : il nome di una funzione Javascript da avviare quando si preme "Genera report" (vedere dettagli dopo)

Analogamente si possono cancellare con

SDK::unsetReport(\$name);

\$name : il nome del report da eliminare (non verranno eliminati i files)

La classe specificata in \$class deve rispettare questa struttura, di seguito esempio:

```
<?php
require_once('modules/Reports/ReportRun.php');

class ReportRunAccounts extends ReportRun {
    []
    []var $enableExportPdf = true;
```

```

var $enableExportXls = true;
var $enablePrint = true;
var $hideParamsBlock = true;

function __construct($reportid) {
    $this->reports = Reports::getInstance($reportid); // crmv@172034
    $this->reportid = $reportid;
    $this->primarymodule = 'Accounts';
    $this->reporttype = '';
    $this->reportname = 'Account con sito';
    $this->reportlabel = getTranslatedString($this->reportname, 'Reports');
}

function getSDKBlock() {
    global $mod_strings;
    $sdblock = '<p><h2>Questo report mostra le aziende con sito</h2></p>';
    // here I can also add custom inputs to filter the report or any html I need
    return $sdblock;
}

// overridden, always hide the summary tab
function hasSummary() {
    return false;
}

// overridden, always show the total tab
function hasTotals() {
    return true;
}

// generate the report
function GenerateReport($outputformat = "", $filterlist = null, $directOutput=false) {
    global $adb;

    // compatibility, please use set them with the proper methods
    if (!empty($outputformat)) {
        $format = "HTML";
        $tab = "MAIN";
    }

    if (strpos($outputformat, 'HTML') !== false) $format = "HTML";

```

```

        if (strpos($outputformat, 'PRINT') !== false) $format = "PRINT";
        if (strpos($outputformat, 'PDF') !== false) $format = "PDF";
        if (strpos($outputformat, 'XLS') !== false) $format = "XLS";
        if (strpos($outputformat, 'JSON') !== false) $format = "JSON";
        if (strpos($outputformat, 'CV') !== false) $format = "NULL";
    }

    if (strpos($outputformat, 'COUNT') !== false) $tab = "COUNT";
    if (strpos($outputformat, 'TOTAL') !== false) $tab = "TOTAL";
    if (strpos($outputformat, 'CV') !== false) $tab = "CV";
}

$this->setOutputFormat($format, $directOutput);
$this->setReportTab($tab);
} else {
    $format = $this->outputFormat;
    $tab = $this->reportTab;
}

$format = $this->outputFormat;
$direct = $this->directOutput;
$tab = $this->reportTab;

// prepare the output class
$output = $this->getOutputClass();
$output->clearAll();

$return_data = array();

if ($tab == 'COUNT' && $this->hasSummary()) {

    // no summary for this custom report
}

elseif ($tab == 'CV') {

    // no customview for this report
}

elseif ($tab == 'MAIN') {

    $sSQL = $this->getReportQuery($outputformat, $filterlist);

```

```

    $result = $adb->query($sSQL);
    $this->total_count = $adb->num_rows($result);

    $error_msg = $adb->database->ErrorMsg();
    if(!$result && $error_msg!=""){
        // Performance Optimization: If direct output is required
        if($direct) {
            echo getTranslatedString('LBL_REPORT_GENERATION_FAILED', 'Reports') . "<br>" . $error_msg;
            $error_msg = false;
        }
        // END
        return $error_msg;
    }

    if($result) {

        $this->generateHeader($result, $output);

        while ($row = $adb->fetchByAssoc($result)) {
            $colcount = count($row);
            foreach ($row as $column => $value) {
                $cell = array(
                    'value' => $value,
                    'column' => $column,
                    'class' => 'rptData',
                );
                $output->addCell($cell);
            }
            $output->endCurrentRow();
        }

        $output->countTotal = $this->total_count;
        $output->countFiltered = $this->total_count;

        if ($format == 'XLS') {
            $head = $output->getSimpleHeaderArray();
            $data = $output->getSimpleDataArray();
            foreach ($data as $row) {
                $return_data[] = array_combine($head, $row);
            }
        }
    }

```

```

        }
    } else {
        $return_data[] = $output->output(!$direct);
        $return_data[] = $this->total_count;
        $return_data[] = $sSQL;
        $return_data[] = $colcount;
    }
}

}

}

} elseif ($tab == "TOTAL" && $this->hasTotals()) {
    //
    $output->addHeader(array('column' => 'fieldname', 'label' => getTranslatedString('Totals')));
    $output->addHeader(array('column' => 'sum', 'label' => getTranslatedString('SUM')));
    $output->addHeader(array('column' => 'avg', 'label' => getTranslatedString('AVG')));
    $output->addHeader(array('column' => 'min', 'label' => getTranslatedString('MIN')));
    $output->addHeader(array('column' => 'max', 'label' => getTranslatedString('MAX')));
    //
    // fixed totals
    $rows = array(
        array(
            array('column'=> 'fieldname', 'value' => 'Fatturato totale', 'class' => 'rptData'),
            array('column'=> 'sum', 'value' => 2000, 'class' => 'rptTotal'),
            array('column'=> 'avg', 'value' => null, 'class' => 'rptTotal'), // not used
            array('column'=> 'min', 'value' => 850, 'class' => 'rptTotal'),
            array('column'=> 'max', 'value' => null, 'class' => 'rptTotal'), // not used
        ),
    );
    //
    // add them to the output class
    foreach ($rows as $row) {
        foreach ($row as $cell) {
            $output->addCell($cell);
        }
        $output->endCurrentRow();
    }
    //
    // format for xls or html
    if ($format == "XLS") {
        //
    }
}

```

```

    // change the output array to match the expected format for XLS export
    $return_data = array();
    $data = $output->getSimpleDataArray();
    $fieldName = '';
    foreach ($data as $row) {
        $nrow = array();
        foreach ($row as $key => $value) {
            if ($key == 'fieldname') {
                $fieldName = $value;
                continue;
            }
            $klabel = $fieldName.'_'.$strtoupper($key);
            $nrow[$klabel] = $value;
        }
        $return_data[] = $nrow;
    }

    } else {
        $return_data = $output->output(!$direct);
    }

}

}

return $return_data;
}

// generate a fixed header for the report
function generateHeader($result, $output, $options = array()) {
    global $adb, $table_prefix;

    $module = 'Accounts';
    $tabid = getTabid($module);
    $count = $adb->num_fields($result);

    for ($x=0; $x<$count; ++$x) {
        $fld = $adb->field_name($result, $x);

        // get the field label from the column (if possible)
        $res = $adb->pquery("SELECT fieldlabel FROM {$table_prefix}_field WHERE columnname = ? and tabid = ?",
            array($fld->name, $tabid));

```

```

    if ($res && $adb->num_rows($res) > 0) {
        $fieldlabel = $adb->query_result_no_html($res, 0, 'fieldlabel');
        $headerLabel = getTranslatedString($fieldlabel, $module);
    } else {
        $headerLabel = $fld->name;
    }

    $hcell = array(
        'column' => $fld->name,
        'label' => $headerLabel,
        'orderable' => false,
        'searchable' => false,
    );

    $output->addHeader($hcell);
}

}

// generate the report query
function getReportQuery($outputformat, $filterlist) {
    global $table_prefix;

    $query = 'SELECT accountid, accountname, website, phone FROM '.$table_prefix.'_account WHERE website <>
    ""';

    return $query;
}
}

```

La funzione Javascript specificata in \$jsfunction deve già essere dichiarata e restituisce una stringa da aggiungere alla request

```

function preRunReport(id) {
    var params = "";
    var select = getObj('picklist1');

    if (select) {
        params += "&picklist1="+select.options[select.selectedIndex].value;
    }

    var selectuser = getObj('picklist2');
    if (selectuser) {
        params += "&picklist2="+selectuser.options[selectuser.selectedIndex].value;
    }
}

```

```
}  
return params;  
}
```

Quando si aggiorna un VTE alla versione 16.09 (o successiva), alcune funzionalità dei report SDK vanno verificate in quanto sono cambiate alcune funzioni.

Se si aggiorna ad un VTE con revisione minore di 1576 è necessario riportare la patch identificata da crmv@140813 (files ReportRun.php e SDKSaveAndRun.php).

Una delle parti cambiate è la gestione dei filtri temporali, che in precedenza si poteva modificare estendendo le funzioni *getPrimaryStdFilterHTML* e *getSecondaryStdFilterHTML*, che ora non vengono più usate.

Per ottenere lo stesso risultato bisogna estendere la funzione *getStdFilterFields* che restituisce un array di campi disponibili, ad esempio:

```
<?php  
function getStdFilterFields() {  
    // See the method Reports::getStdFilterFields for the standard implementation  
      
    // this example just loads standard fields for the Potential module  
    $list = $this->reports->getStdFiltersFieldsListForChain(0, array('Potentials'));  
      
    return $list;  
}
```

Quando il report viene generato, non va più letta la variabile *\$filterlist*, bensì *\$this->stdfilters*, che contiene il filtro da usare.

Se la query del report è completamente personalizzata, va gestita manualmente anche la generazione del filtro temporale, ad esempio usando una funzione come questa:

```
<?php  
function addStdFilters() {  
    global $current_user;  
    $sql = '';  
      
    if (is_array($this->stdfilters)) {  
        foreach ($this->stdfilters as $flt) {  
            if ($flt['fieldid'] > 0) {  
                // get field informations
```



```

        $finfo = $this->getFieldInfoById($flt['fieldid']);
        $table = $finfo['tablename'];
        $qgen = QueryGenerator::getInstance($finfo['module'], $current_user);
        $operator = 'BETWEEN';
        if ($flt['value'] == 'custom') {
            $value = array($flt['startdate'], $flt['enddate']);
        } else {
            $cv = CRMEntity::getInstance('CustomView');
            $value = $cv->getDateforStdFilterBytype($flt['value']);
        }
        // adjust for timezone
        $value[0] = $this->fixDateTimeValue(
            $qgen, $finfo['fieldname'], $value[0]
        );
        $value[1] = $this->fixDateTimeValue(
            $qgen, $finfo['fieldname'], $value[1], false
        );
        // add the condition
        $sql .= ' AND '.$table.'.'.$finfo['columnname'].' ' .
            $operator.' '.$value[0].' AND '.$value[1];
    }
}
}
return $sql;
}

```

Hooks

[modules/Reports/Listview.php](#)
[modules/SaveAndRun.php](#)
[modules/Reports/Reports.php](#)
[modules/Reports/CreatePDF.php](#)
[modules/Reports/CreateXL.php](#)
[modules/Reports/PrintReport.php](#)
[Smarty/templates/ReportContents.tpl](#)
[Smarty/templates/ReportRunContents.tpl](#)
[Smarty/templates/ReportRun.tpl](#)

Revision #1

Created 26 February 2020 12:50:41 by ddalmaso

Updated 14 May 2020 14:04:05 by ddalmaso