



# Filtered Views for Microsoft Dynamics CRM

Version 4.2.13, March 5, 2010  
Copyright 2009-2010 Stunnware GmbH

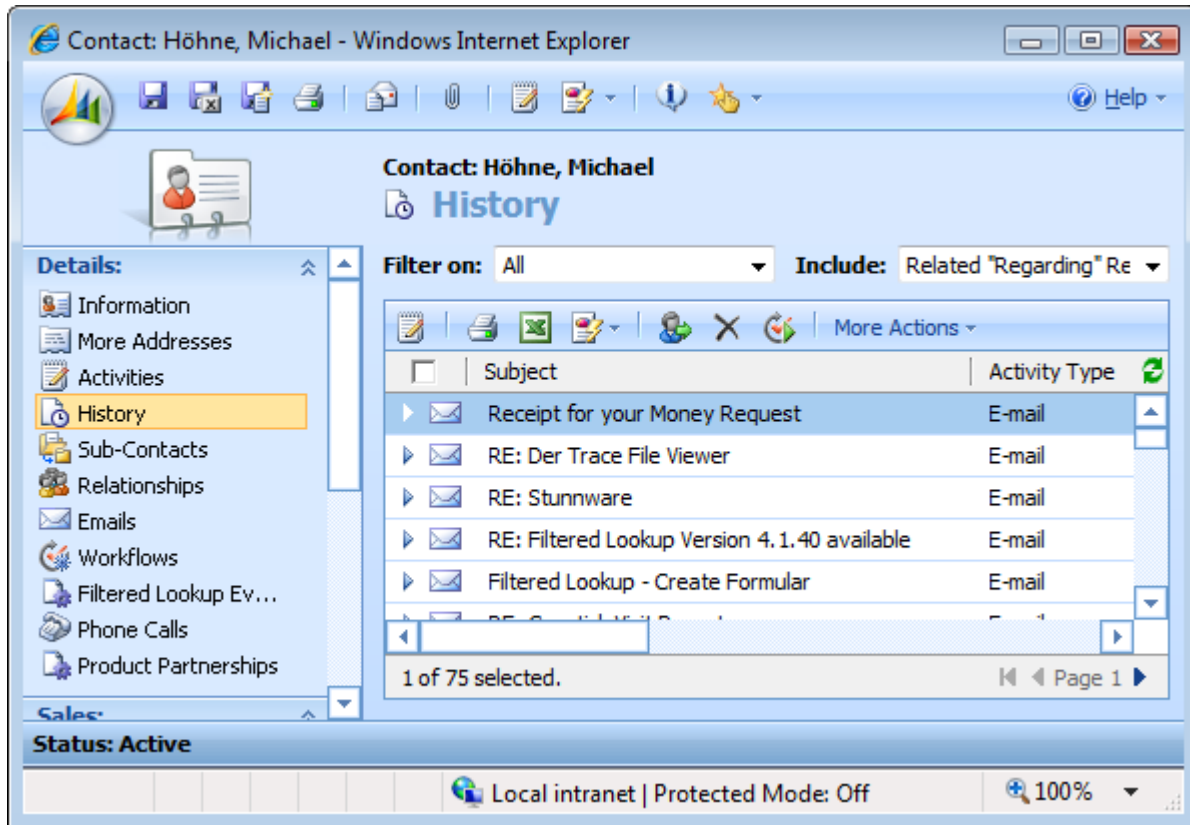
## Contents

---

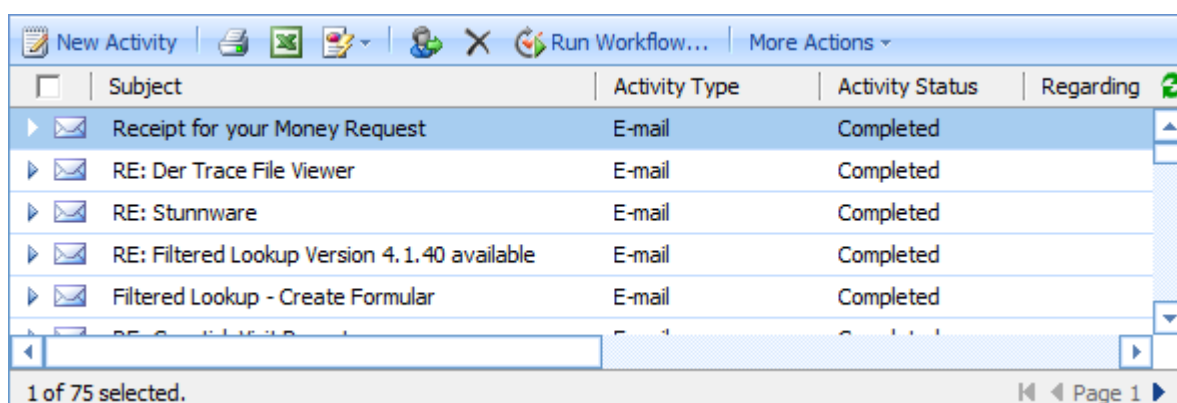
|   |           |
|---|-----------|
| <b>Overview .....</b>                         | <b>3</b>  |
| How it works.....                             | 4         |
| <b>Setup.....</b>                             | <b>5</b>  |
| Contents of the download package.....         | 5         |
| Installing the application.....               | 5         |
| Copying the license file .....                | 5         |
| Configuration.....                            | 6         |
| <b>Customization.....</b>                     | <b>10</b> |
| Defining the view .....                       | 10        |
| Testing the view.....                         | 16        |
| Including a View selection dropdown .....     | 17        |
| View Translations (v4.2) .....                | 18        |
| Implementing the JavaScript Code.....         | 19        |
| Support methods .....                         | 21        |
| Specifying an entity context (v4.2).....      | 22        |
| View OnLoad Event (v4.2).....                 | 23        |
| UI Filters (v4.2).....                        | 24        |
| Using Filtered Views in the CRM Sitemap ..... | 29        |
| <b>Deploying Filtered Views .....</b>         | <b>31</b> |
| Exporting Data.....                           | 31        |
| Importing Data.....                           | 32        |

## Overview

Filtered Views are a concept to show data in a view that is filtered on some other data. For instance, let's look at the default history view of a contact:



To see more of the view definition, here's the same view without the form:



| Subject                                      | Activity Type | Activity Status | Regarding |
|--|---------------|-----------------|-----------|
| Receipt for your Money Request               | E-mail        | Completed       |           |
| RE: Der Trace File Viewer                    | E-mail        | Completed       |           |
| RE: Stunnware                                | E-mail        | Completed       |           |
| RE: Filtered Lookup Version 4.1.40 available | E-mail        | Completed       |           |
| Filtered Lookup - Create Formular            | E-mail        | Completed       |           |

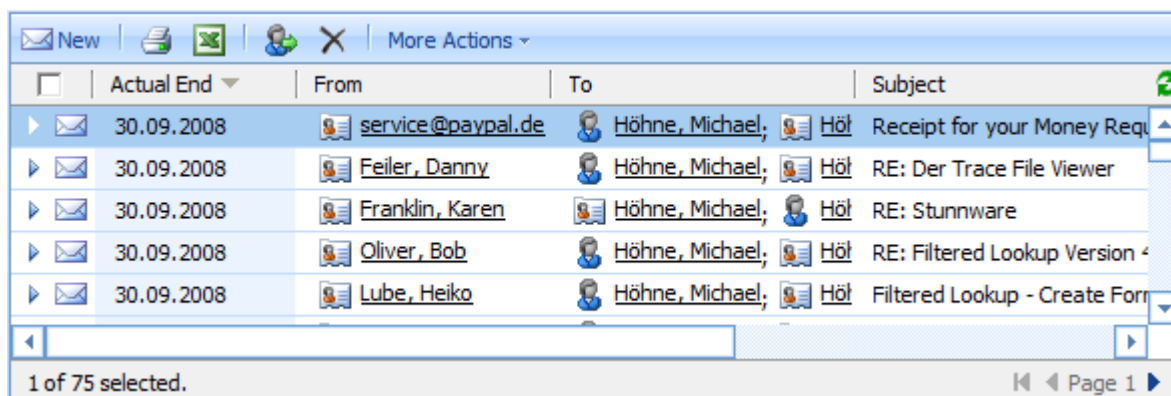
The above history view displays all kinds of activities, but let's assume that you want to have an email view similar to what you know from your Outlook client. How do you do it? Though you can modify the history view in the activity

entity, you can only select fields common to all activity types. So you need a pure email view and though there is a relationship between the contact and the email entities, you cannot change its definition.

As a workaround you could create a new relationship between contacts and emails and write a plug-in that automatically associates contacts and emails whenever an email is modified. While that should work, it seems quite a bit overhead to just view data.

An alternative is to create a report in SQL Server Reporting Services and display it in an IFRAME. Or you can create your own solution and try to mimic the CRM UI. All of this will work, but whenever you need another view, you have to repeat these steps.

The Filtered Views add-on is based on a different idea. Instead of using workarounds to mimic a behavior that already is an integral part of Microsoft Dynamics CRM, it uses the existing advanced find functionality to assemble a view filter and layout, and lets Microsoft Dynamics CRM render it. To give you an example how it could look like, here's an email view based on the Filtered Views add-on:



This looks more like an Outlook view, though not exactly due to the limited capabilities of CRM views in general. The view above does display emails associated to the contact record where it is displayed in. However, it did not require you to write a custom solution, create a report, or otherwise develop a solution.

## How it works

The solution is based on an article available at <http://www.stunnware.com/crm2/topic.aspx?id=AdvancedFind1>, meaning that you can create this solution on your own. However, you may also decide to simply use the Filtered Views and the price of this product is fair enough that it shouldn't be hard to decide what direction to go.

The Filtered View uses a Fetch XML query and a view layout, assembles it and uses the Advanced Find functionality of Microsoft Dynamics CRM to execute the query and have it been rendered. The result can then be shown in an IFRAME on the form or as an entry in the navigation bar. It could be even used in the sitemap, but as you cannot easily pass parameters to a link in the sitemap and don't have an entity context where such parameters could come from, it's certainly easier to use a standard view in the sitemap.

Because the add-on uses the Advanced Find functionality and because this functionality is not documented by Microsoft, the Filtered Views add-on is an unsupported extension. However, it cannot do anything bad to your data, because it only reads information from CRM and does not change anything.

## Setup

---

The Filtered Views add-on is installed through a setup program and the most recent version is always available for download from the Stunnware website at [www.stunnware.com](http://www.stunnware.com). Before starting the installation, please check if there is a newer version available.

### Contents of the download package

The download is an archive in zip format containing the setup package that installs the add-on on either a CRM Server, the CRM Offline Client or both of them, depending what software it finds on your machine. The directory contains the following files:

- **Stunnware Filtered Views 4.0.msi**: This is the installer package containing the Filtered Views binaries and web source files.
- **Setup.exe**: This is the setup program used to install the software on a 32-bit system.
- **Setup64.exe**: This is the setup program used to install the software on a 64-bit system.

Running Setup64.exe is only required when installing the software on a CRM Server x64 edition. Use Setup.exe in all other cases, including a 64-bit client, because there is no 64-bit version of Microsoft Office Outlook available today.

### Installing the application

Run Setup.exe or Setup64.exe, whichever is appropriate, and follow the instructions on the screen. Simply accept all default values, though you can change the installation folder if you want. The installation first installs the Filtered Views binaries to the installation folder, which by default is [Program Files]\Stunnware\Filtered Views.

After installing the application, which merely is a file copy process, a custom installer action is executed. This action searches the registry for the existence of a CRM Server and CRM Offline Client installation. If it finds such entries, then the previously installed files are copied into the following locations in the CRM web of the server or client respectively:

- Assemblies (\*.dll) are copied into the **bin** folder of the CRM web.
- All other files are copied into the **ISV/Stunnware/Filtered Views** folder.

Note that if you run Setup.exe on a 64-bit server, the custom installer action will not find the information of the CRM Server setup in the registry. Though installation will succeed, the Filtered Views files will not be copied to the CRM Server web. You can manually copy the files into their proper locations though.

### Copying the license file

You need a license file to run the Filtered Views Add-On on a CRM server machine. The same license file is not required on the Microsoft CRM Outlook Offline client though. If you don't have a license file, please request one by following the instructions on <http://www.stunnware.com/?area=products&group=fv4&subarea=fv4-download>.

The license file is always named "license.lic" and has to be copied into the /isv/stunnware.com/FilteredViews subdirectory of the CRM web. Please make sure that the file security of the license.lic file is set to inherit its permissions from the parent folder. License files are always sent in a zip file and when moving the license directly from the zip file to the above stated directory, the file permissions may be set differently, preventing the add-on from accessing it.

## Configuration

Besides the application files installed during setup, you also have to import the Filtered Views customizations. There are no changes made to any existing entity or the sitemap. You have to modify the ISV.Config though to add the toolbar buttons to the Filtered Views custom entity. The necessary configuration files are installed during setup. You find them in the [Program Files (x86)]\Stunnware\Filtered Views for Microsoft Dynamics CRM 4.0\Customizations folder, unless you have chosen a different installation path:

- **Filtered Views Customizations.zip:** Contains the customizations required to run the add-on.
- **Filtered Views ISV.Config Extensions:** Contains the extensions to the ISV.Config needed to create and test Filtered Views.

### Importing the Filtered Views customization file

The following instructions assume that you are familiar with importing and publishing customizations. Please refer to the Microsoft Dynamics CRM documentation otherwise.

In the Microsoft CRM web client, navigate to the Settings page, select Customizations and then click on Import Customizations. Select the **Filtered Views Customizations.zip** file and upload it. The CRM client will display the customizations included in the customization file. There are four entities in the package:

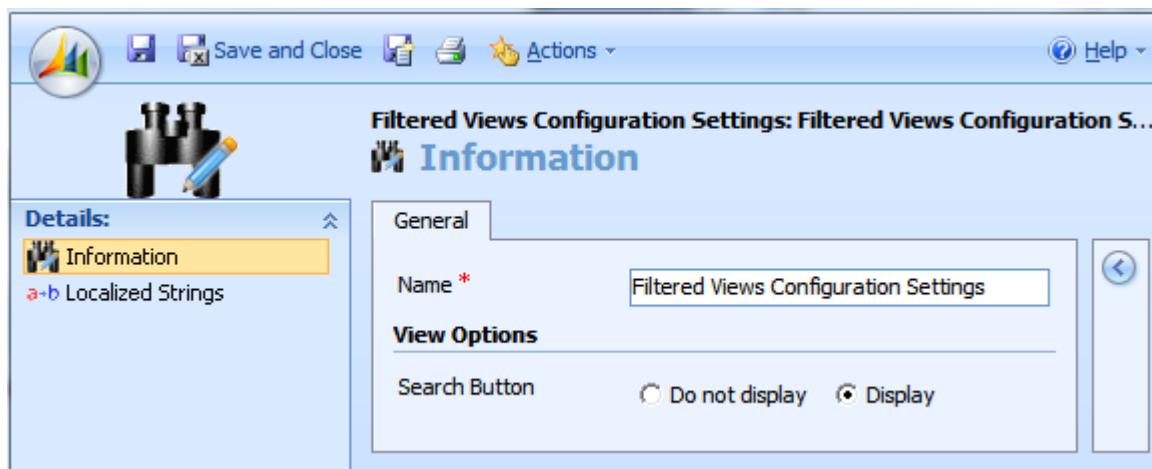
- The Filtered Views Configuration Settings store organization-wide settings used in every view.
- The Filtered View entity is used to define the query and layout of a filtered view.
- Filtered Views Search Parameters allow adding search controls to the view. They are referred to as "UI Filters" in the UI and this documentation.
- Localized Strings are used to localize view names and parameter names.

Select all four entities and import them into your organization. After the import has finished, select Customize Entities from the customization page and open the Filtered View entity. At the bottom of the General tab, you will see options defining where to display the entity. You should include it in the Settings area, but can choose any area you prefer. If you don't select any area, then you will find it difficult to create a new Filtered View record later on.

Repeat the same steps for the Filtered Views Configuration Settings entity to display it in the Settings area. You may have to press CTRL-F5 in your browser to let it become visible.

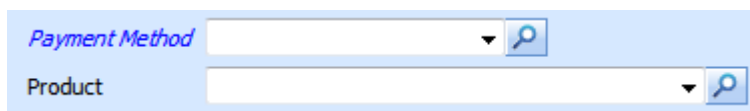
## Configuring organization settings

Once you see the new entities in the Settings area, create a new Filtered Views Configuration Settings record:

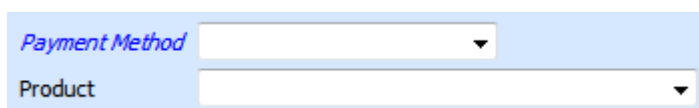


The Name is not relevant but initialized to Filtered Views Configuration Settings. Only one Filtered Views Configuration Settings record should exist in the system. If multiple records are stored, then the values from the first record being found are used.

The **Search Button** option controls if a search button is displayed next to UI filters. UI filters are explained in detail in a separate chapter. When setting it to "Display", UI parameters look like this:

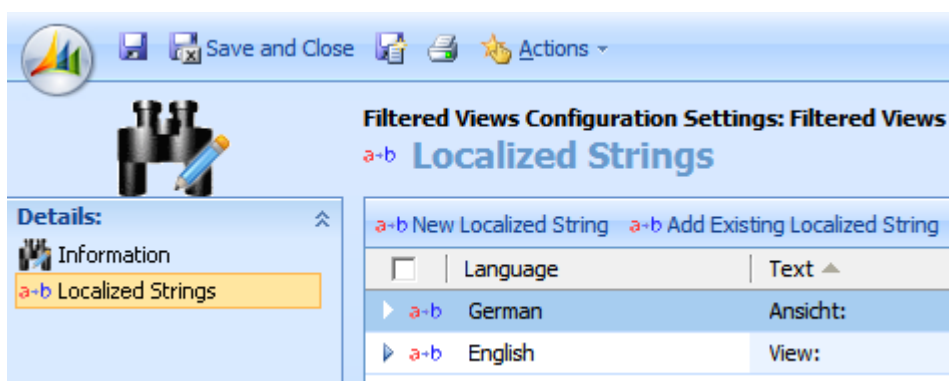


Setting it to "Do not display" results in the same filter controls but without the search button:



A user can always initiate a new search by hitting the Enter key, which is much faster especially in text fields. However, many users will miss the search button if it's not available, so it's your choice whether to use it or not.

The **Localized Strings** view is used to specify the "View:" label in different languages.



"View:" is used as the label for the view selection dropdown. Please refer to the "Including a View selection dropdown" chapter for more information about view selections.

## Adding the ISV.Config extensions

The Filtered View entity uses two toolbar buttons that have to be declared in the ISV.Config. To add these extensions, export your existing ISV.Config and add the content of the **Filtered Views ISV.Config Extensions.xml** file to your ISV.Config:

```
<configuration version="3.0.0000.0">
  <Entities>
    <!-- Copy anything between the two comments into the Entities section of your ISV.Config file. -->
    <Entity name="sw_filteredview">
      <Grid>
        <MenuBar>
          <Buttons>
            <Button Icon="/isv/stunnware.com/FilteredViews/img/Export.png" JavaScript="var selectedRecords = crmGrid.InnerGrid.SelectedRecords;&#xD;&#xA;&#xD;&#xA;if ((selectedRecords != null) &amp;&amp;(selectedRecords.length &gt; 0)) {&#xD;&#xA;&#xD;&#xA; var ids = null;&#xD;&#xA;&#xD;&#xA; for(var index in selectedRecords) {&#xD;&#xA;&#xD;&#xA; var row = selectedRecords[index];&#xD;&#xA;&#xD;&#xA; if (ids == null) {&#xD;&#xA; ids = row[0];&#xD;&#xA; }&#xD;&#xA; &#xD;&#xA; else {&#xD;&#xA; ids = ids + ';' + row[0];&#xD;&#xA; } &#xD;&#xA; } &#xD;&#xA; &#xD;&#xA; window.navigate(prependOrgName('/isv/stunnware.com/FilteredViews/export.aspx') + '?data=' + ids, '_self'); &#xD;&#xA;}">
              <Titles>
                <Title LCID="1033" Text="Export"/>
              </Titles>
              <ToolTips>
                <ToolTip LCID="1033" Text="Export the selected records"/>
              </ToolTips>
            </Button>
            <Button Icon="/isv/stunnware.com/FilteredViews/img/Export.png" JavaScript="window.navigate(prependOrgName('/isv/stunnware.com/FilteredViews/export.aspx?all=1'))">
              <Titles>
                <Title LCID="1033" Text="Export All"/>
              </Titles>
              <ToolTips>
                <ToolTip LCID="1033" Text="Export all records"/>
              </ToolTips>
            </Button>
            <Button Icon="/isv/stunnware.com/FilteredViews/img/Import.png" JavaScript="window.navigate(prependOrgName('/isv/stunnware.com/FilteredViews/import.aspx'))">
              <Titles>
                <Title LCID="1033" Text="Import"/>
              </Titles>
              <ToolTips>
                <ToolTip LCID="1033" Text="Import records"/>
              </ToolTips>
            </Button>
          </Buttons>
        </MenuBar>
      </Grid>
      <ToolBar>
        <Button ValidForCreate="1" ValidForUpdate="1" Icon="/_imgs/ico_18_answer.gif" JavaScript="SwSelectView()" Client="Web,Outlook">
          <Titles>
            <Title LCID="1033" Text="Select View" />
          </Titles>
          <ToolTips>
            <ToolTip LCID="1033" Text="Copy the query and layout from an existing view" />
          </ToolTips>
        </Button>
        <Button ValidForCreate="0" ValidForUpdate="1" Icon="/_imgs/ico_16_1039_advFind.gif" JavaScript="SwPreview()" Client="Web,Outlook">
          <Titles>
            <Title LCID="1033" Text="Preview" />
          </Titles>
          <ToolTips>
            <ToolTip LCID="1033" Text="Preview the filtered view, if possible." />
          </ToolTips>
        </Button>
      </ToolBar>
    </Entity>
  </Entities>
</configuration>
```



```

</Button>
</ToolBar>
</Entity>
<!-- Copy anything between the two comments into the Entities section of your ISV.Config file. -->
</Entities>
</configuration>

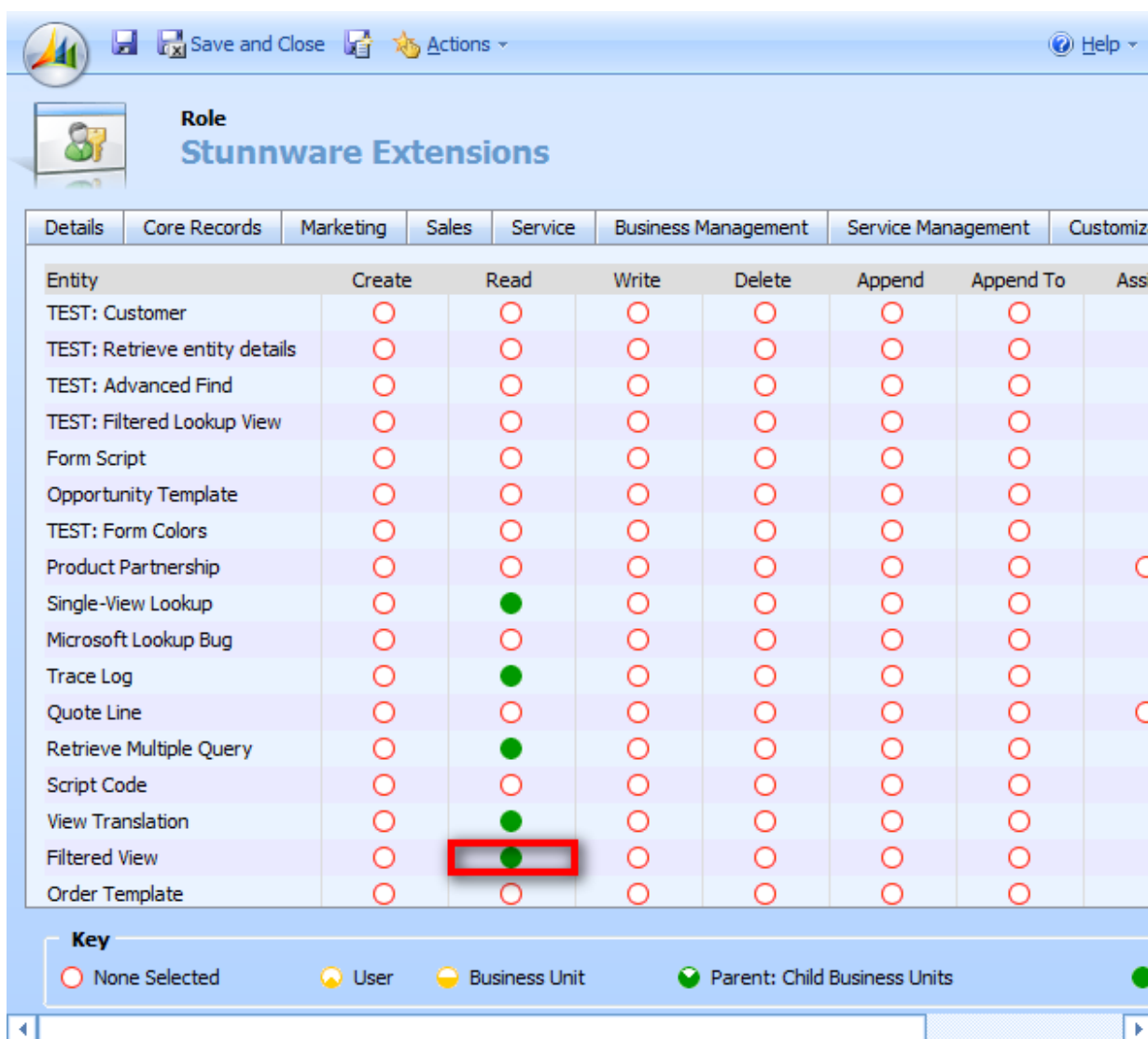
```

Save the modified file and import it back into CRM.

Tip: You can use the [Stunnware Tools for Microsoft Dynamics CRM 4.0](#) to easily modify the ISV.Config and Sitemap.

### Adjusting security roles

To use the Filtered Views in Microsoft Dynamics CRM, a user has to have read access to the Filtered View records:



Please change your security roles accordingly.

To create Filtered Views or change existing Filtered Views, you need create and write privileges:

| Entity        | Create | Read | Write |
|---------------|--------|------|-------|
| Filtered View | ●      | ●    | ●     |

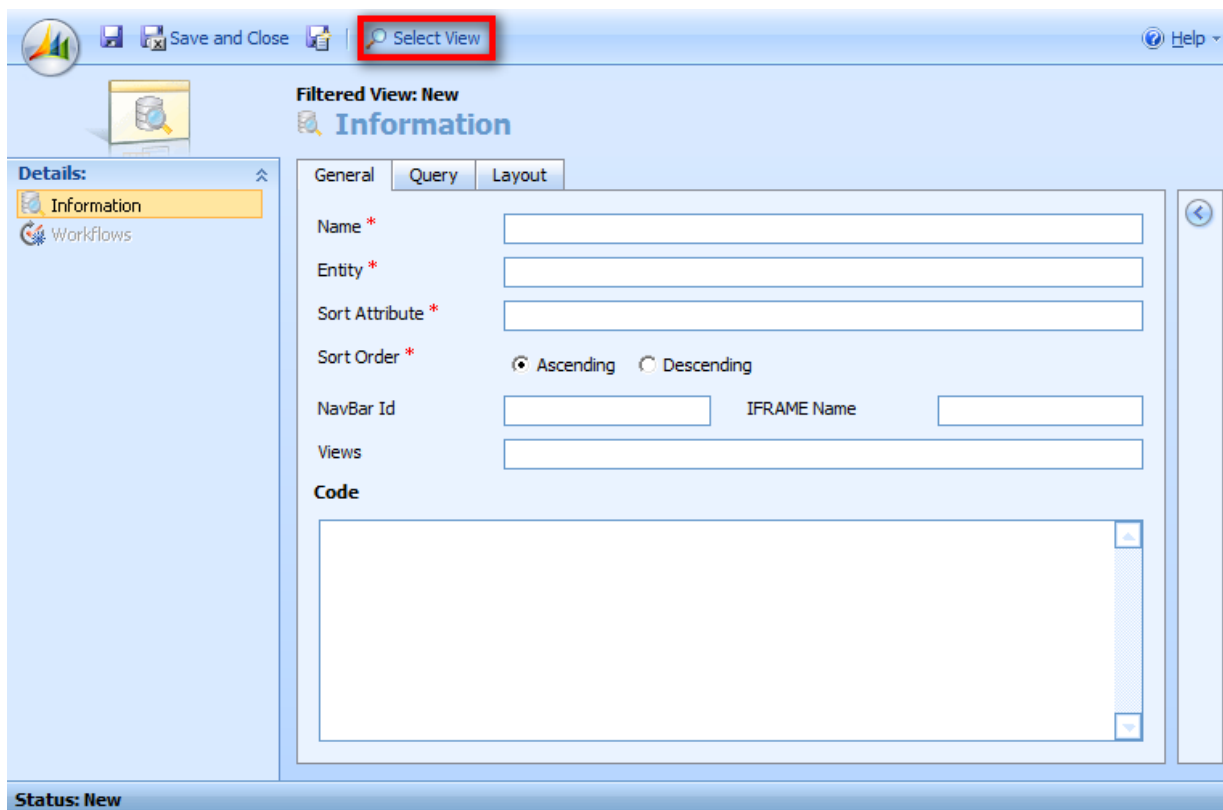
## Customization

Creating and using a Filtered View is divided into two parts. First, you create the query and the layout. Second, you implement the code needed to access this definition. Both parts are mostly automated.

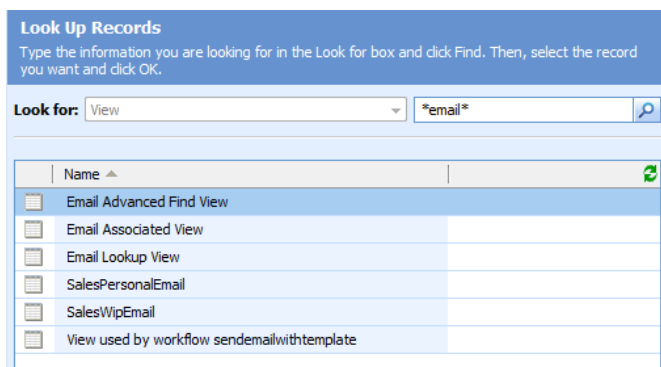
### Defining the view

#### General

Let's start with a new Filtered View and usually you are using the definition of an existing view as a starting point.

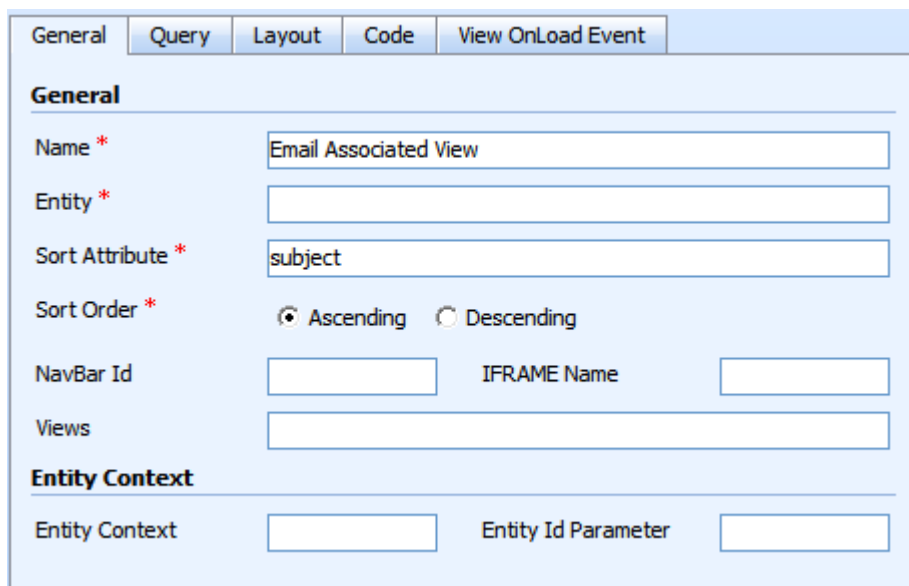


You can of course type in the data manually, but as mentioned before, the definition of a view and its later usage in code is mostly automated. So instead of trying to find out what to type in, simply click the Select View button, displaying a standard lookup dialog:



The lookup dialog displays all system views initially, but as we want to create a view displaying emails, type in \*email\* in the search box and only a few items are left.

Reusing the lookup functionality is undocumented and therefore unsupported. But again, it cannot harm your data. Select any view you like as a template and click the OK button. I have chosen the Email Associated View.



| General               | Query   | Layout              | Code | View OnLoad Event |
|-----------------------|---|---------------------|------|-------------------|
| <b>General</b>        |   |                     |      |                   |
| Name *                | Email Associated View   |                     |      |                   |
| Entity *              |   |                     |      |                   |
| Sort Attribute *      | subject   |                     |      |                   |
| Sort Order *          | <input checked="" type="radio"/> Ascending <input type="radio"/> Descending |                     |      |                   |
| NavBar Id             |   | IFRAME Name         |      |                   |
| Views                 |   |                     |      |                   |
| <b>Entity Context</b> |   |                     |      |                   |
| Entity Context        |   | Entity Id Parameter |      |                   |

After selecting a view, a lot of things happen instantly:

1. The name of the selected view is stored in the name field on the form.
2. The entity type name is stored in the entity field.
3. The sort attribute and sort order defined in the view are stored in the form as well.
4. The code (on the Code tab) is created.
5. The query (on the query tab) is populated, if there is a query stored in the view.
6. The view layout (on the layout tab) is populated as well.

The **name** obviously is the name you want to refer to this view. You can name it whatever you like, but make sure not to use the same name twice, because the code you have to add to the OnLoad event refers to a view definition by name.

The **entity** stores the type name of the entities being displayed in the view, which is "email" in this example.

The **sort attribute** and **sort order** define how the data is initially sorted. Be aware that the entity type name and sort attribute are case-sensitive. If you receive an error when using a Filtered View, then please double-check these names first.

The **NavBar Id** stores the id of a navigation bar item displayed in the CRM form showing the view. As that may be somewhat difficult to understand, let's recap what we want to do: display a view containing emails on the contact form and only these emails should be displayed, where the contact is the sender or in one of the recipient lists (to, cc, bcc).

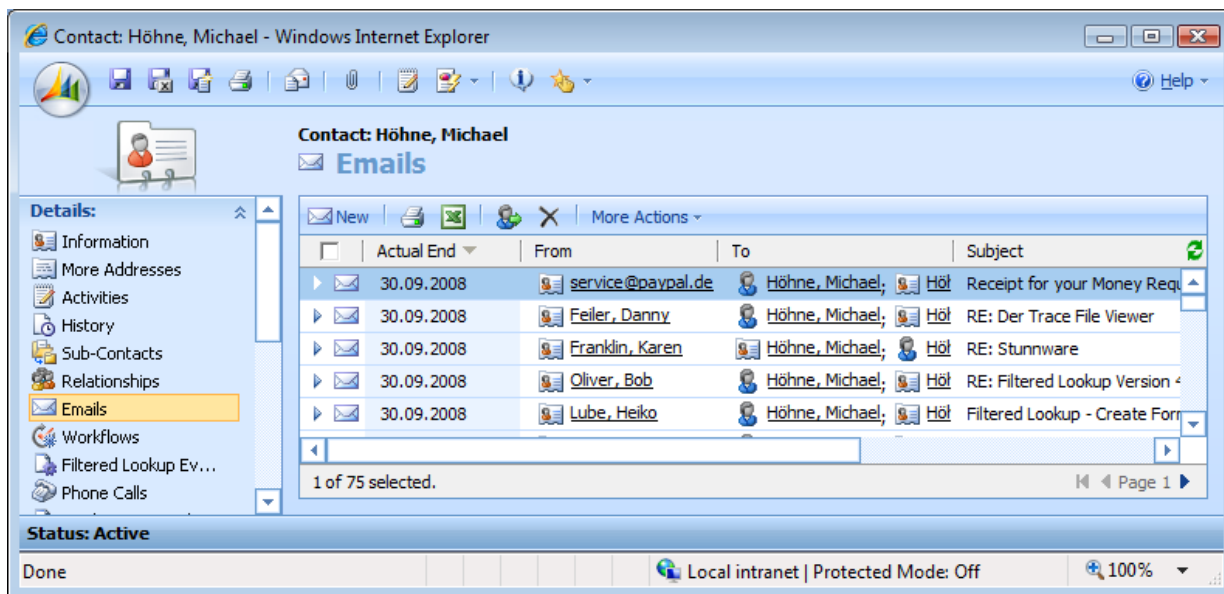
The **Views** field is used to specify multiple views. Users can then select a view from a dropdown list. Please refer to chapter "Including a View selection dropdown" for more information.

The **Entity Context** is used to control field mappings when clicking the "New" button. It is explained in the "Specifying an entity" context chapter.

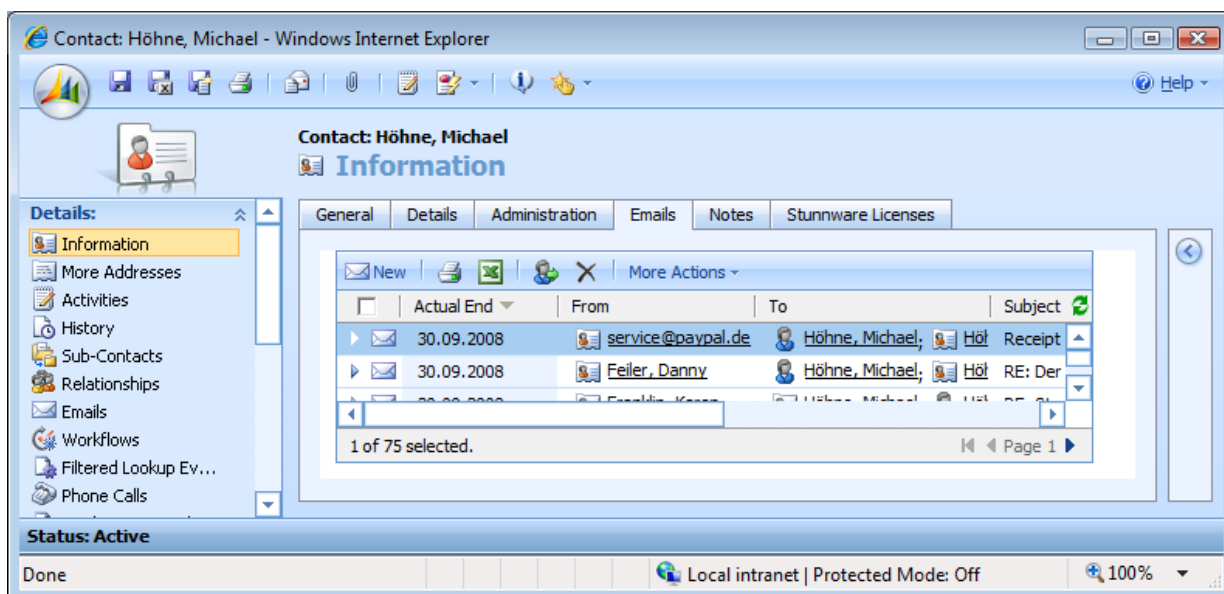
**Note:** Some view definitions do not have a Fetch XML query and when selecting such a view, the query and entity fields remain blank. You should always get the layout definition though.

There are two possible locations for the email view on the CRM form:

1. In the navigation bar:



2. In an IFRAME on the form:



When you want to display the view like an associated view, then you have to know the id of the navigation bar item you want to use and write it into the **NavBar Id** field. If you want to display it in an IFRAME, then you need to know the name of that frame - which simply is the name you have specified when creating it - and write it into the **IFRAME Name** field. If you want to display the view in both the navigation bar and an IFRAME, then simply specify both values.

**Tip:** The easiest way to add an entry to the navigation bar is by declaring a new item in ISV.Config. However, you can also override the behavior of any other entry when specifying the id of an already existing navigation bar item. If you don't know the name of the navigation bar item, then you can add the following line to the script.aspx file in the /isv/stunnware.com/FilteredViews folder:

```
loadArea = function(sArea, sParams, sUrl, bIsvMode) {
    alert(sArea);
    for(var key in _sw002NavItemArray) {
        if (key == sArea) {
            sUrl = _sw002NavItemArray[key].getUrl();
            break;
        }
    }
    _sw001LoadArea(sArea, sParams, sUrl, bIsvMode);
}
```

With the alert command in place you will see a message box popping up whenever you select an item in the navigation bar. As soon as you have written down the name of the item you need, you should remove the alert command.

## Query

Each view in Microsoft Dynamics CRM uses Fetch XML to define the query. When selecting a view from the view lookup dialog, the Fetch XML is extracted and stored in the query field, as shown in the image below.



You can modify the query manually or use tools such as the [Stunnware Tools for Microsoft Dynamics CRM 4.0](#) to change it.

## Parameters

Filtered Views are designed to be dynamic, so you will have at least one parameter in your query. Though it is possible to use a Fetch XML query without a parameter, it wouldn't make too much sense as you could simply use a system view or advanced find view to deliver the same results.

As our goal is to display all emails associated to a contact record, our email view needs one parameter, the contact id. If you have used the [Filtered Lookup for Microsoft Dynamics CRM 4.0](#) in the past, then you already know the following:

```
<fetch mapping="logical">
  <entity name="email">
    <attribute name="actualend" />
    <attribute name="cc" />
    <attribute name="prioritycode" />
    <attribute name="regardingobjectid" />
    <attribute name="sender" />
    <attribute name="subject" />
    <attribute name="torecipients" />
    <order attribute="actualend" descending="true" />
    <link-entity name="activityparty" from="activityid" to="activityid">
      <filter>
        <condition attribute="partyid" operator="eq" param="id" />
      </filter>
    </link-entity>
  </entity>
</fetch>
```

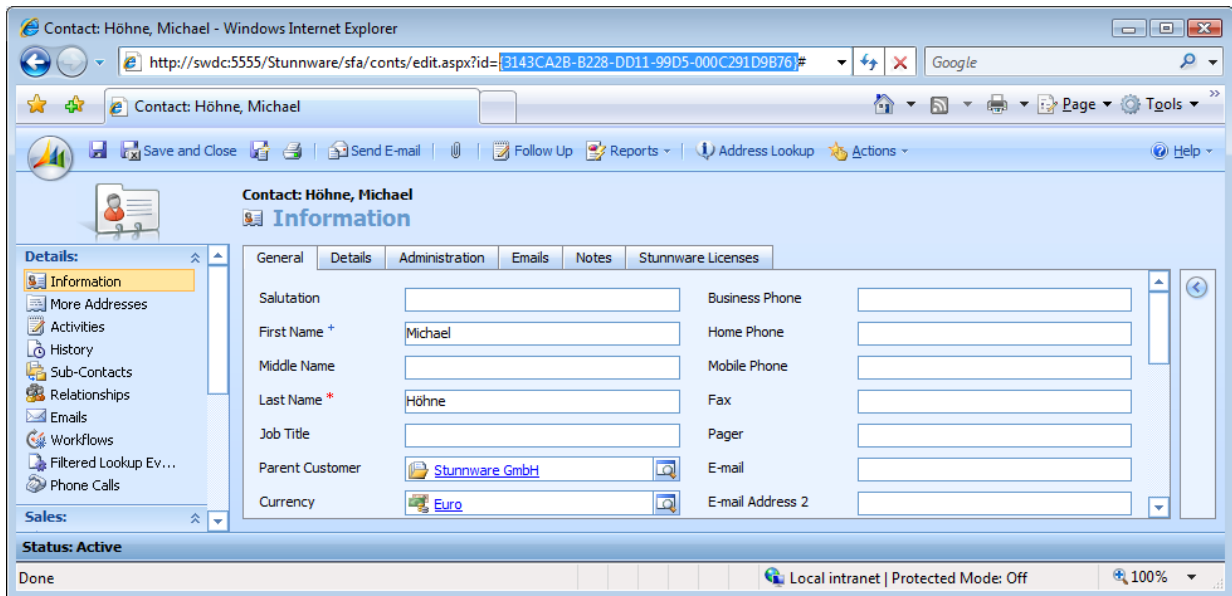
The above query returns the actualend, cc, prioritycode, regardingobjectid, sender, subject and torecipients attributes from emails that are associated to a specific record. "Associated record" means that the record is available in one of the activityparty items linked to the email, which are the sender and all recipients in the "to" and "cc" and "bcc" fields.

The param attribute in the condition (marked in yellow) does not exist in Fetch XML. It is used as a placeholder and is replaced with the Guid of a contact record at runtime, which you do in the CRM form JavaScript code. A later chapter describes this code.

Before implementing any code you of course want to test the view and for that you need to specify a test value for the id parameter. These parameters are stored in the "Test Parameters" text box and each parameter is entered on a new line. Parameter name and value are delimited by an equal sign.

When you want to pass an array of values (in, not-in, between and not-between operators) you can specify multiple values by separating them with "#;#", without quotes, e.g. **value1#;#value2#;#value3**.

For the email view we need a contact id though it could be an account or lead as well in this particular query. To obtain the id, choose a contact that has some associated emails and open the record. Then press CTRL-N to display the record in a new window:



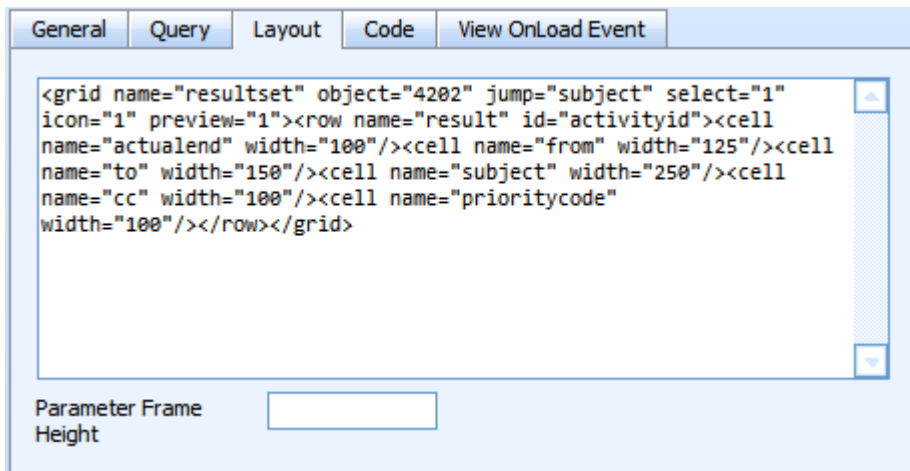
The highlighted part in the address bar is the contact id and CTRL-N (shortcut for **N**ew Window) is an easy way to get to the id of any CRM record. Another method is selecting the "Copy Shortcut" command from the "Actions" menu.

If the parameter is a numerical or text value, you simply specify a test value instead of looking for a Guid. Place the following into the Test Parameters field:

```
id={3143CA2B-B228-DD11-99D5-000C291D9B76}
```

## View Layout

The last thing to check is the view layout:

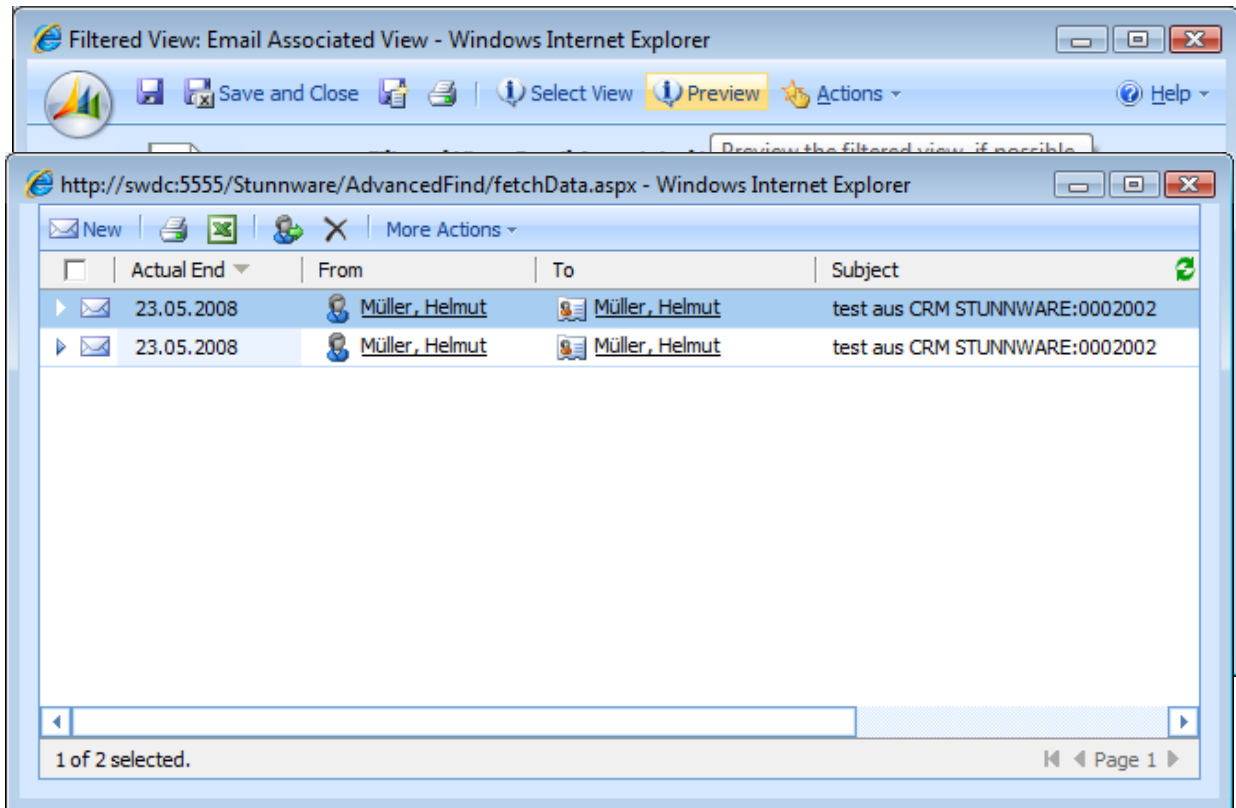


This layout XML is automatically stored when choosing a view through the "Select View" toolbar button. If you want to use the layout XML from an existing advanced find query, I again recommend a tool like the Stunnware Tools for Microsoft Dynamics CRM 4.0 to extract the layout XML.

The Parameter Frame Height defines a fixed height of the area used for UI parameters. If left blank, the height is calculated. It is described in the "UI Filters" chapter.

## Testing the view

You are now ready to test the view. Click the "Preview" tool bar button as shown in the next image:



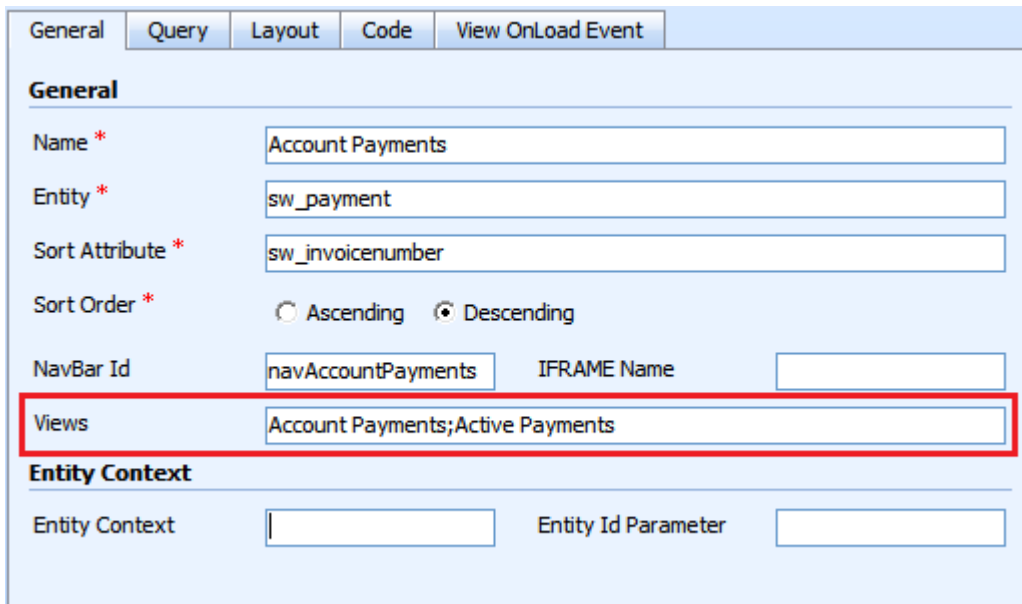
Note that in order for the preview to be available, you have to save the entity. Also make sure that the Fetch XML and Layout XML are available and that all parameters are specified in the test parameters field. Then click the Preview button:

The view is displayed in a new window and you instantly see the filter in effect.



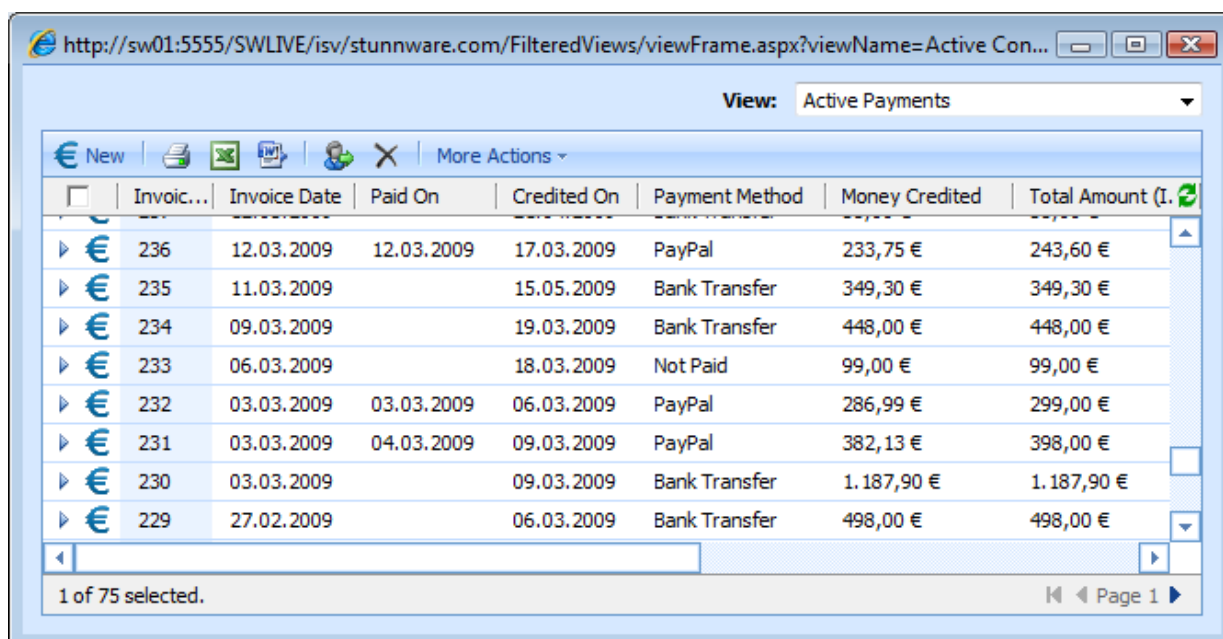
## Including a View selection dropdown

When you have created multiple views and want to let the user select a view from a list, then simply enter the view names in the Views field, separating each view with a semicolon:



The screenshot shows a configuration window with tabs for 'General', 'Query', 'Layout', 'Code', and 'View OnLoad Event'. The 'General' tab is active. Fields include: Name (\*), Entity (\*), Sort Attribute (\*), Sort Order (\*), NavBar Id, IFRAME Name, Views, Entity Context, and Entity Id Parameter. The 'Views' field is highlighted with a red border and contains the text 'Account Payments;Active Payments'.

The code will be updated automatically and when clicking the Preview button again you are presented with the following UI:



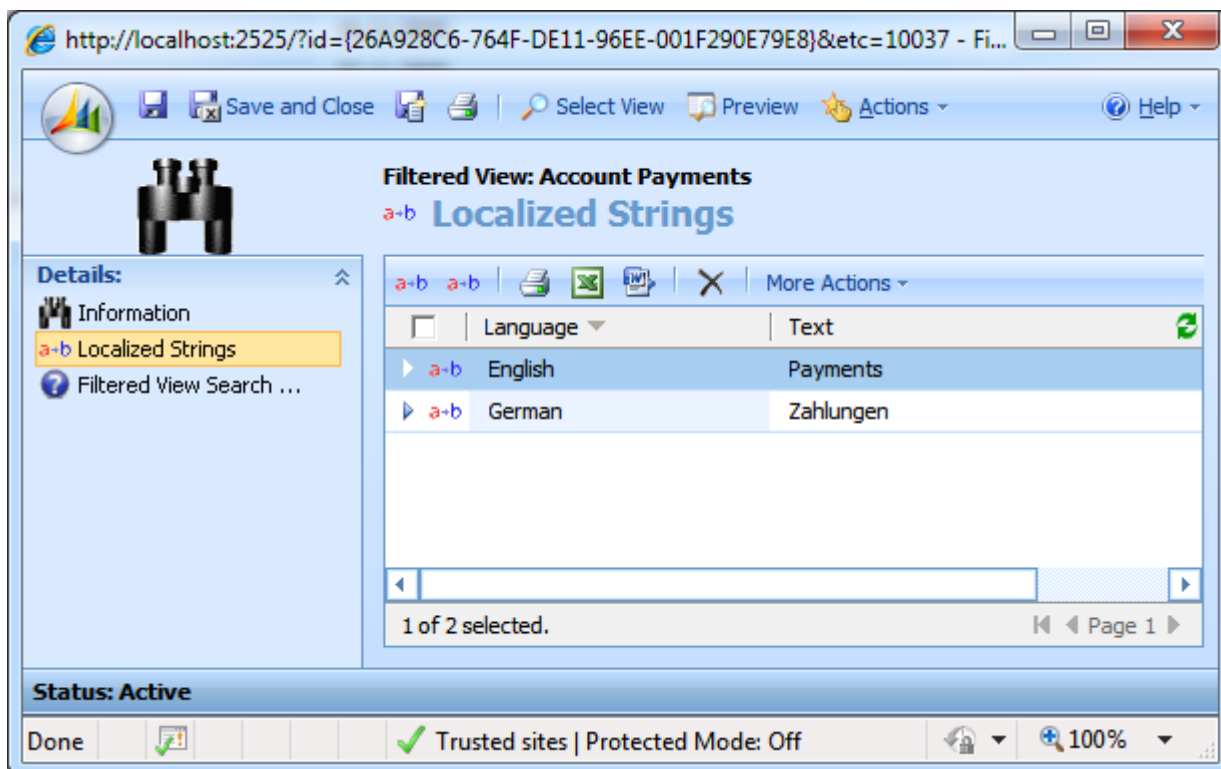
The screenshot shows a web browser window displaying a table of payment records. The browser address bar shows the URL: <http://sw01:5555/SWLIVE/isv/stunnware.com/FilteredViews/viewFrame.aspx?viewName=Active Con...>. The page title is 'View: Active Payments'. The table has the following columns: Invoic..., Invoice Date, Paid On, Credited On, Payment Method, Money Credited, and Total Amount (I. The table contains 8 rows of data. The status bar at the bottom indicates '1 of 75 selected.' and 'Page 1'.

|     | Invoic... | Invoice Date | Paid On    | Credited On | Payment Method | Money Credited | Total Amount (I. |
|-----|-----------|--------------|------------|-------------|----------------|----------------|------------------|
| ▶ € | 236       | 12.03.2009   | 12.03.2009 | 17.03.2009  | PayPal         | 233,75 €       | 243,60 €         |
| ▶ € | 235       | 11.03.2009   |            | 15.05.2009  | Bank Transfer  | 349,30 €       | 349,30 €         |
| ▶ € | 234       | 09.03.2009   |            | 19.03.2009  | Bank Transfer  | 448,00 €       | 448,00 €         |
| ▶ € | 233       | 06.03.2009   |            | 18.03.2009  | Not Paid       | 99,00 €        | 99,00 €          |
| ▶ € | 232       | 03.03.2009   | 03.03.2009 | 06.03.2009  | PayPal         | 286,99 €       | 299,00 €         |
| ▶ € | 231       | 03.03.2009   | 04.03.2009 | 09.03.2009  | PayPal         | 382,13 €       | 398,00 €         |
| ▶ € | 230       | 03.03.2009   |            | 09.03.2009  | Bank Transfer  | 1.187,90 €     | 1.187,90 €       |
| ▶ € | 229       | 27.02.2009   |            | 06.03.2009  | Bank Transfer  | 498,00 €       | 498,00 €         |

The view selection dropdown displays the views you specified in the Views field and the user can now switch between views in the same way you do in standard CRM views.

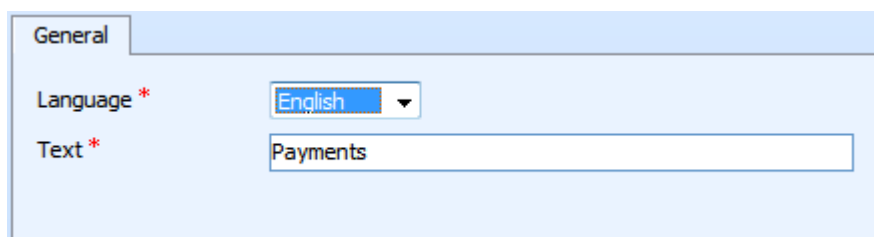
## View Translations (v4.2)

By default, the view dropdown displays the names of the Filtered Views as specified in the name field. While sufficient in a system using only one language, you most probably want to define translations when using multiple languages.



In the above example there are two translations for the „Account Payments“ view. When used on an English client, the view name is “Payments” and when used on a German client, the name is “Zahlungen”. If no appropriate string is available for the language of the user, then the original view name is used.

The „Localized String“ entity only has two fields, one for the language and another for the label:



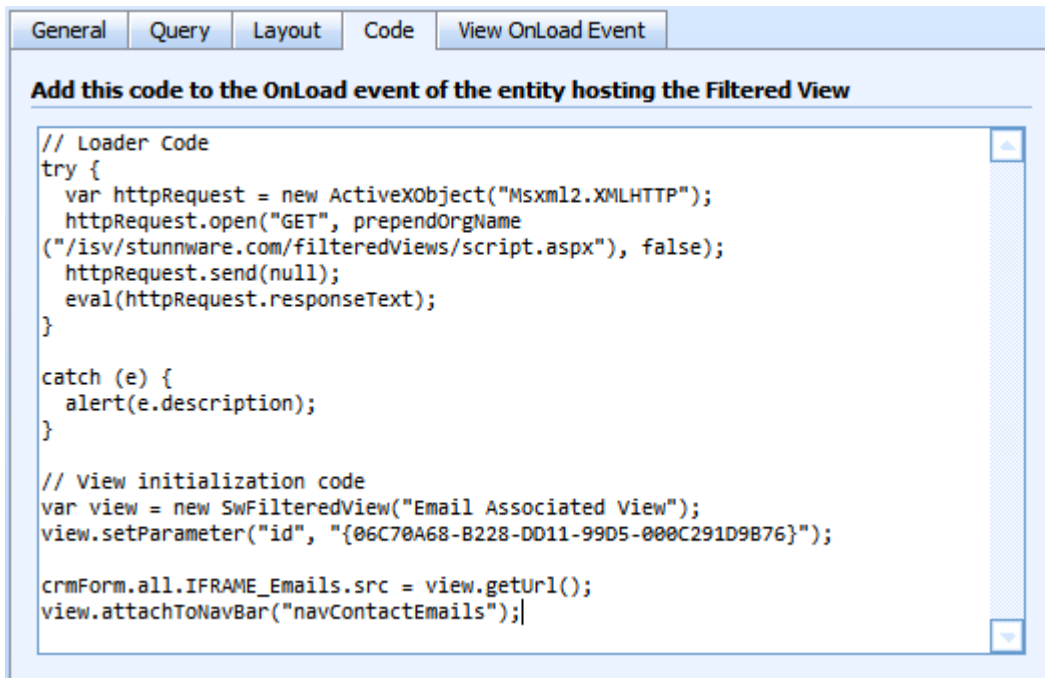
The screenshot shows a form with a 'General' tab. It contains two fields:

- Language \***: A dropdown menu with 'English' selected.
- Text \***: A text input field containing 'Payments'.

The same entity is used to localize UI parameters, which are explained in a later chapter.

## Implementing the JavaScript Code

To implement the Filtered View, you simply copy the JavaScript code from the Filtered View item into the OnLoad code of the entity where you want to display it.



In our example we want to display an email view in a contact record and therefore we copy the JavaScript code right into the contact's OnLoad event:

```

// Loader Code
try {
  var httpRequest = new ActiveXObject("Msxml2.XMLHTTP");
  httpRequest.open("GET", prependOrgName("/isv/stunnware.com/filteredViews/script.aspx"), false);
  httpRequest.send(null);
  eval(httpRequest.responseText);
}

catch (e) {
  alert(e.description);
}

// View initialization code
var view = new SwFilteredView("Email Associated View");
view.setParameter("id", "{06C70A68-B228-DD11-99D5-000C291D9B76}");

crmForm.all.IFRAME_Emails.src = view.getUrl();
view.attachToNavBar("navContactEmails");
  
```

The loader code loads a piece of JavaScript and makes the Filtered View API available in the form script. Do only include it once and always do so in the OnLoad event. Loading it multiple times won't harm, but will affect performance.

Note that the parameter defined in the Fetch XML query is passed to the Filtered View add-on by calling the setParameter method. You call setParameter once for each parameter you have to transmit. The fixed value {06C70A68-B228-DD11-99D5-000C291D9B76} is taken from the list of test parameters and you have to change the code to make it dynamic:

```
// View initialization code
if (crmForm.ObjectId != null) {

    // Loader Code
    try {
        var httpRequest = new ActiveXObject("Msxml2.XMLHTTP");
        httpRequest.open("GET", prependOrgName("/isv/stunnware.com/filteredViews/script.aspx"), false);
        httpRequest.send(null);
        eval(httpRequest.responseText);
    }

    catch (e) {
        alert(e.description);
    }

    var view = new SwFilteredView("Email Associated View");
    view.setParameter("id", crmForm.ObjectId);

    crmForm.all.IFRAME_Emails.src = view.getUrl();
    view.attachToNavBar("navContactEmails");
}
}
```

The above implementation only initializes the view when the contact record already was saved. This check is done by testing the `crmForm.ObjectId` for a null value. The entire loader code is only executed if the record was saved before, because otherwise our Fetch XML query couldn't work. Note that this is not a general setup, but totally makes sense in this example.

You usually do not have to care about most of the code and your primary task in the JavaScript code is replacing the values of the test parameters with the values used at runtime. You can of course change the view filter at any time. Say that your query filters data based on a picklist named "new\_category" and the parameter name is "category":

```
// Loader Code
try {
    var httpRequest = new ActiveXObject("Msxml2.XMLHTTP");
    httpRequest.open("GET", prependOrgName("/isv/stunnware.com/filteredViews/script.aspx"), false);
    httpRequest.send(null);
    eval(httpRequest.responseText);
}

catch (e) {
    alert(e.description);
}

// View initialization code
MyGlobals = new Object();
MyGlobals.CategoryView = new SwFilteredView("Categories");

// View OnChange implementation
MyGlobals.CategoryOnChange = function() {

    var categoryValue = (crmForm.all.new_category.DataValue == null) ?
        "-1" : crmForm.all.new_category.DataValue;

    MyGlobals.CategoryView.setParameter("category", categoryValue);
    crmForm.all.IFRAME_Categories.src = MyGlobals.CategoryView.getUrl();
}

// Initialize the view after loading the form
MyGlobals.CategoryOnChange();
```

The above code declares a new [global variable](#) named "MyGlobals". Instead of the "My" in MyGlobals you better use your customization prefix in case someone else reads this document and uses the same name, but as I don't know your customization prefix, I named it "MyGlobals". For Stunnware a good name would be "SwGlobals" or "StunnwareGlobals".

The view (SwFilteredView) is stored in the CategoryView property of the global MyGlobals object and we also add a function to MyGlobals: CategoryOnChange. It only consists of three lines that calculate the picklist value, specify it as a parameter and update the IFRAME. Finally this function is called in the last line of the OnLoad script.

To update the view whenever you are selecting a different value in the picklist, you simply add the following line to the picklist's OnChange event:

```
// new_category OnChange event implementation
MyGlobals.CategoryOnChange();
```

Note that MyGlobals is still available in the OnChange event implementation, because it was declared as a global variable. You also don't have to add the loader code, because the SwFilteredView class definition is also declared globally.

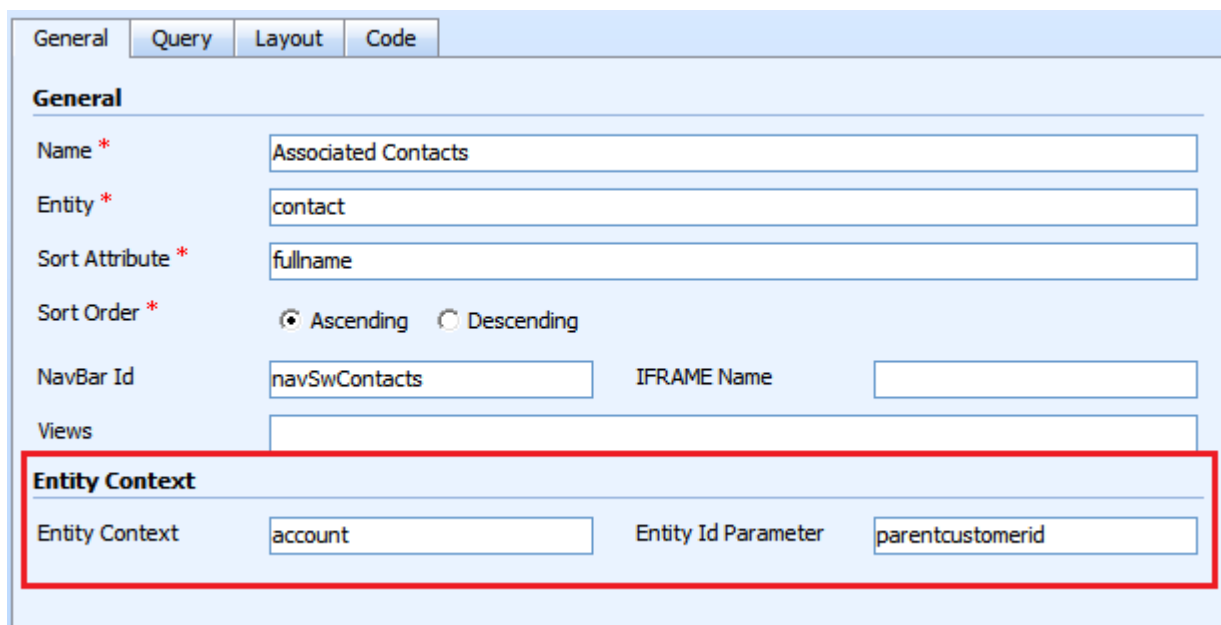
## Support methods

The Filtered View API defines the following variables and methods for easier implementation:

|  |  |
|--|--|
| SW_EMPTY_GUID                          | The string representation of an empty Guid.<br>"{00000000-0000-0000-0000-000000000000}"  |
| SW_FILTERED_VIEWS_LICENSED             | Will either be true or false. "True" means that you have a proper license and the Filtered Views add-on can be used. "False" means that you don't have a license, have more users than stated in the license or have an evaluation license that expired. |
| SwGetBooleanParameterValue(boolValue)  | Boolean values need to be passed as either "0" (false) or "1" (true). You can use the SwGetBooleanParameterValue function to convert a Boolean value to its appropriate string representation.   |
| SwGetDateTimeParameterValue(dateValue) | Date values need to be passed in the format 2007-12-31T16:45:35-02:00. Use the SwGetDateTimeParameterValue function to convert a date value to its appropriate string representation.  |

## Specifying an entity context (v4.2)

By default, when pressing the New button in a Filtered View, the new record is not related to the form hosting the view. For instance, when hosting a contact view in an account and creating a new contact from the view, you would expect that the parent customer field is set to the account and the field mappings specified on the account-contact relationship are used to populate the new contact.



The screenshot shows a configuration window with tabs for 'General', 'Query', 'Layout', and 'Code'. The 'General' tab is active. Fields include: Name (\*), Entity (\*), Sort Attribute (\*), Sort Order (\*), NavBar Id, IFRAME Name, Views, Entity Context, and Entity Id Parameter. The 'Entity Context' field is highlighted with a red border and contains the text 'account'. The 'Entity Id Parameter' field contains the text 'parentcustomerid'.

To create this context you have to tell the Filtered Views what the parent account is. In most cases it will be the hosting form, but as you will see in later chapters, it can make sense to use a different parameter.

The entity context is specified on the General tab of the Filtered View form. The above example is from a definition displaying contacts associated to a specific account. The Entity Context is set to "account", which is the entity type name of a CRM account. The Entity Id Parameter is the name of a parameter you are defining with a call to the `setParameter` method:

```
var view = new SwFilteredView("Associated Contacts");
view.setParameter("parentcustomerid", crmForm.ObjectId);
```

Using an Entity Context of "account" and an Entity Id Parameter of "parentcustomerid" tells the Filtered Views add-on that new records, which are created by clicking the New button in the Filtered View, should use the account as the parent record, which is identified by the primary key equaling the value of the parentcustomerid parameter, and to apply all appropriate field mappings.

The referenced parameter does not to be present in the Fetch XML used in the Filtered View definition, though it's very likely to be used there as well. But you can also pass parameters that are not available in the query if you have a need for it.

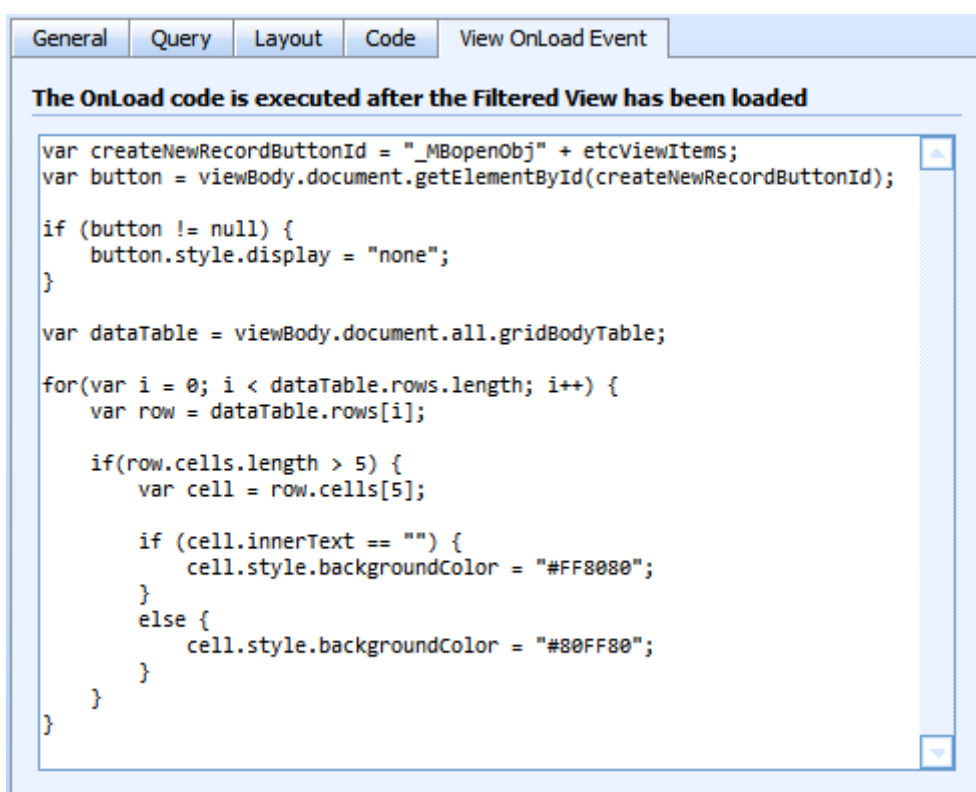
You are also not limited to pass `crmFrom.ObjectId` as in the sample above. Assume that you are displaying the contact view in an opportunity and use a query returning the contacts associated to the potential customer of this

opportunity. When clicking the new button in the contact view it wouldn't make sense to create a new contact related to the opportunity. Instead you want to create a contact associated to the account referenced in the customer field. Instead of passing `crmForm.ObjectId`, which is the primary key of the opportunity record, you would use this code instead:

```
view.setParameter("parentcustomerid", crmForm.all.customerid[0].id);
```

## View OnLoad Event (v4.2)

A big disadvantage of standard CRM views is the absence of a view event model. The Filtered Views, however, allow the execution of custom JavaScript code after a view was loaded.



Two variables are available to your script:

- `etcViewItems`: The entity type code of the items displayed in the view.
- `etcContext`: The entity type code of the context you specified in the Entity Context section (see previous chapter).

Both values are defined as numeric values. If no entity context is specified, then the value of `etcContext` is -1.

The above code hides the New button and then loops through each data row to check the value of the 6<sup>th</sup> column. If it is empty then the cell is highlighted in red, otherwise it is highlighted in green color:

|     | Invoic... | Invoice Date | Paid On | Credited On | Payment Method | Money Credited |
|-----|-----------|--------------|---------|-------------|----------------|----------------|
| ▶ € | 296       | 03.06.2009   |         |             | Not Paid       |                |
| ▶ € | 276       | 06.05.2009   |         | 04.06.2009  | Bank Transfer  | 313,60 €       |
| ▶ € | 186       | 09.01.2009   |         | 04.06.2009  | Bank Transfer  | 313,60 €       |
| ▶ € | 117       | 02.10.2008   |         | 12.12.2008  | Bank Transfer  | 223,83 €       |
| ▶ € | 111       | 25.09.2008   |         | 12.12.2008  | Bank Transfer  | 313,60 €       |

1 of 5 selected. Page 1

You can easily use view scripts to colorize data and manipulate the view in other ways. Such techniques of course must be considered unsupported, because the HTML source of views may change in the future. However, it is an easy way to enhance views with just a few lines of code.

The OnLoad code is executed once after the view has loaded and every time the view is refreshed, either through the refresh button or by navigating through the result set.

## UI Filters (v4.2)

To make your filtered views even more dynamic, you can specify one or more UI filters. A UI filter is a control accepting a parameter that you would usually set in code. For instance, to further filter the view shown in the last image by payment method, you could add a dropdown control:

|     | Invoic... | Invoice Date | Paid On | Credited On | Payment Method | Money Credited |
|-----|-----------|--------------|---------|-------------|----------------|----------------|
| ▶ € | 296       | 03.06.2009   |         |             | Not Paid       |                |
| ▶ € | 276       | 06.05.2009   |         | 04.06.2009  | Bank Transfer  | 313,60 €       |
| ▶ € | 186       | 09.01.2009   |         | 04.06.2009  | Bank Transfer  | 313,60 €       |
| ▶ € | 117       | 02.10.2008   |         | 12.12.2008  | Bank Transfer  | 223,83 €       |
| ▶ € | 111       | 25.09.2008   |         | 12.12.2008  | Bank Transfer  | 313,60 €       |

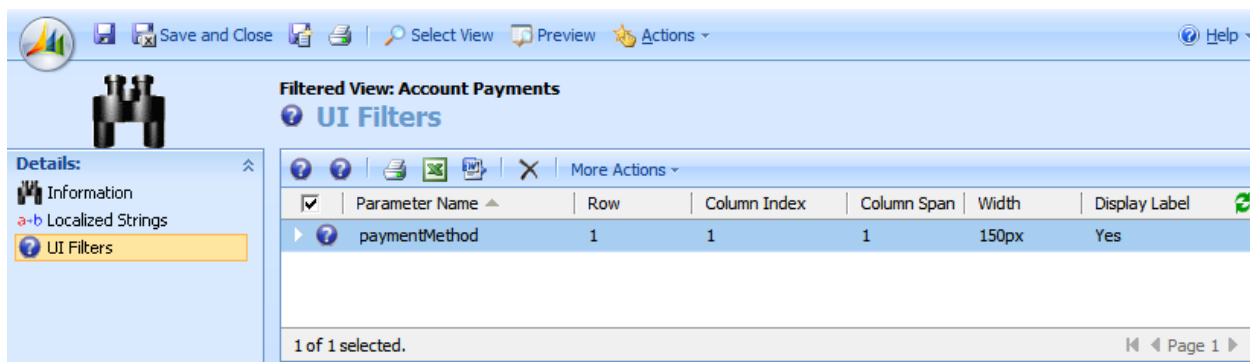
1 of 5 selected. Page 1

A UI Filter can be a text control or a dropdown control. While text controls are meant to be used for free text searches, the dropdown controls are either bound to the CRM metadata or to existing data. For instance, when filtering on the Payment Method as shown in the above picture, you want to use the picklist values as defined in the CRM metadata.

### Defining a UI filter

UI Filters are available in the UI Filters associated view:





To create a new UI filter, click the “New” button in the view.

The screenshot shows the 'Dynamic Values' configuration window for a UI filter. It has two tabs: 'General' and 'Dynamic Values'. The 'General' tab is active and contains the following fields:

- Parameter Name \***:
- CRM Attribute**:
- Control Type \***:
- Row \***:
- Display Label \***:  No  Yes
- Column Index \***:
- Column Span**:
- Width \***:
- Options**:  Update view when selection changes (Picklist)
- Labels**:

All parameters, including parameters defined through an UI filter, are passed to the Filtered Views add-on as part of the query string. The **Parameter Name** is the name of such a parameter and therefore must not contain spaces or special characters.

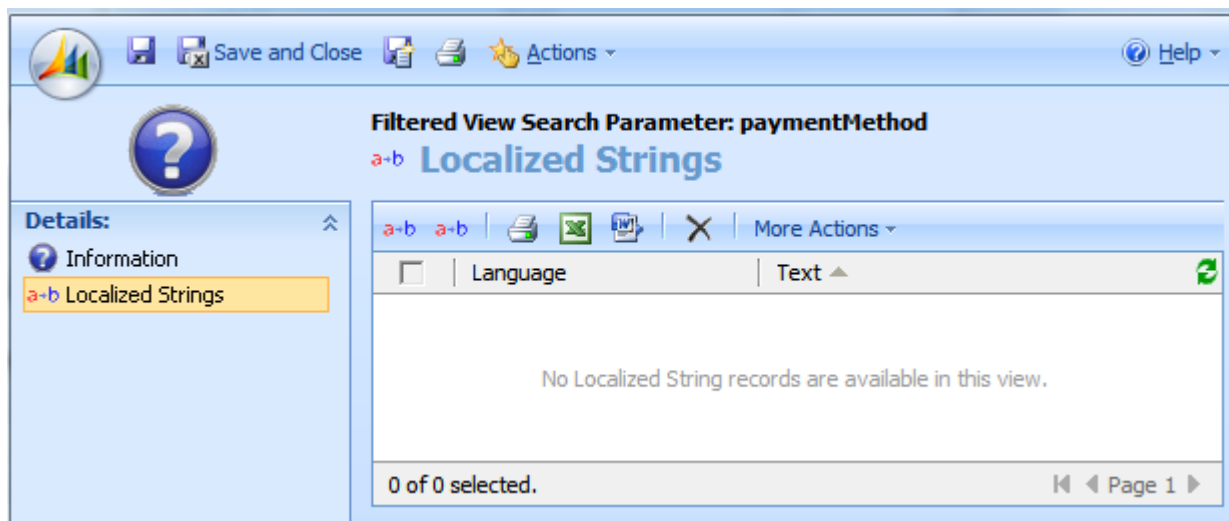
The Control Type is either a text box or a picklist (dropdown control). To have a picklist displaying the available options as defined in the metadata, you simply specify the entity and attribute name in the **CRM Attribute** field. In the above image it is set to sw\_payment.sw\_paymentmethod. Note the dot between sw\_payment and sw\_paymentmethod. The part left of the dot – sw\_payment in this example – identifies the entity name. The part right of the dot – sw\_paymentmethod – identifies the attribute name. With this setting the Filtered Views will retrieve the picklist options of the attribute sw\_paymentmethod of the entity sw\_payment from the metadata. If you would like to populate a picklist with the available industry types of the account entity, then you would specify account.industrycode in the CRM Attribute field.

The **Layout** section defines where to display the UI filter. All filter controls are placed into a table. The **Row** and **Column Index** properties specify where the UI control is shown. The **Width** and **Column Span** values are used to control the size of the filter control. You are mostly free in designing the search area and whatever you specify in

these fields will be used in the generated HTML code. For that matter, if you use inappropriate values, the output may be scrambled.

When using the picklist control you can use the **Update view when selection changes** option to apply the filter whenever the user selects a different value in the picklist. If you do not activate this option, the user has to press the Enter key to apply the new filter value.

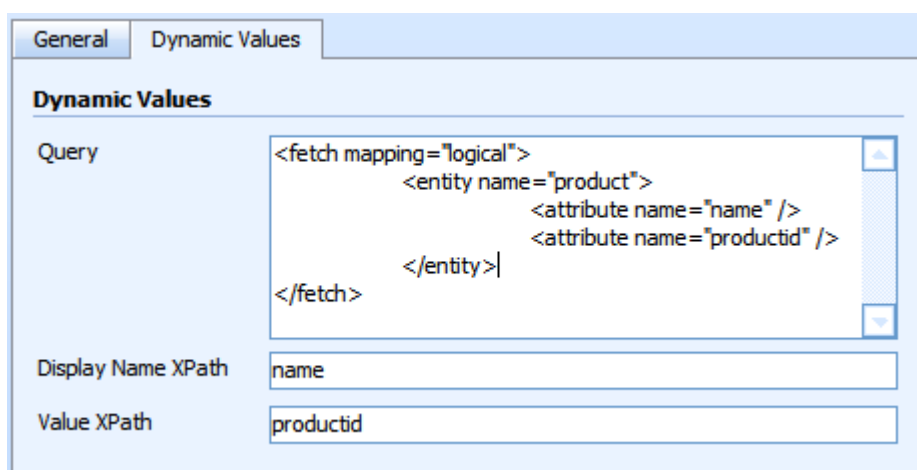
**Display Label** defines whether or not to display a label for the filter control. The text being displayed is stored in the **Default Label** field. However, you can localize the label by creating appropriate translations.



The Localized Strings are the same you use to define view translations. If no translation is found, the text in the Default Label field is used.

You can use HTML code to format a label. In this example it is set to `<font color='blue'><i>Payment Method</i></font>` and therefore the text "Payment Method" is displayed in italics and blue color.

The **Dynamic Values** tab allows populating a picklist control with values that have been already stored in the CRM database.



It again uses a Fetch query defining the data to use. The above example returns the name and id of all products in your system. This query returns an XML document, similar to the following:

```

<resultset>
  <result>
    <name>Filtered Lookup for Microsoft Dynamics CRM 4.0</name>
    <productid>{A6624E22-B2DB-DC11-AB43-000C291D9B76}</productid>
  </result>
  <result>
  <result>
    <name>Stunnware Tools for Microsoft Dynamics CRM 4.0</name>
    <productid>{06D86D0F-8B52-DD11-A739-000C29A5B586}</productid>
  </result>
  <result>
  <result>
    <name>JavaScript Factory for Visual Studio 2008</name>
    <productid>{41402C36-2FA4-DE11-8AAA-001F290E79E8}</productid>
  </result>
  <result>
    <name>Filtered Views</name>
    <productid>{E49F3C41-B81A-DE11-B036-001F290E79E8}</productid>
  </result>
</resultset>

```

The **Display Name XPath** defines which attribute to use for the display names in the picklist. The XPath you are specifying is relative to /resultset/result, so a value of "name" means that the XPath expression /resultset/result/name is used for the display name.

Accordingly the **Value XPath** defines which value to pass to the Filtered View add-on when the user select an item in the picklist. The value of "productid" means that the XPath expression /resultset/result/productid is used.

To make it obvious what happens have a look at the generated HTML code:

```

<SELECT style="WIDTH: 250px" onkeydown=searchKeyDown() id=productId onchange=ChangeView(>
  <OPTION selected value=""></OPTION>
  <OPTION value={A6624E22-B2DB-DC11-AB43-000C291D9B76}>Filtered Lookup for Microsoft
Dynamics CRM 4.0 </OPTION>
  <OPTION value={06D86D0F-8B52-DD11-A739-000C29A5B586}>Stunnware Tools for Microsoft
Dynamics CRM 4.0</OPTION>
  <OPTION value={41402C36-2FA4-DE11-8AAA-001F290E79E8}>JavaScript Factory for Visual
Studio 2008</OPTION>
  <OPTION value={E49F3C41-B81A-DE11-B036-001F290E79E8}>Filtered Views</OPTION>
</SELECT>

```

The user sees a list of product names, but the value is the product id.

### Using UI Filters in the Filtered View query

Of course the idea of having UI filters is to further narrow down the list of items displayed in the view. To use a UI filter, you do the same as for any other parameter:

```

<fetch mapping="logical">
  <entity name="sw_payment">
    <attribute ... />
    <filter>
      <condition attribute="statecode" operator="eq" value="0" />
      <condition attribute="sw_paymentmethod" operator="eq" param="paymentMethod" />
    </filter>
  </entity>

```

</fetch>

So as soon as you have defined a UI filter, you can use it as a parameter in the query of the Filtered View. Just make sure to use the parameter name as you have specified in the UI Filter record:

The screenshot shows a configuration window with two tabs: 'General' and 'Dynamic Values'. The 'Dynamic Values' tab is active. Under the 'General' section, there are two fields: 'Parameter Name \*' with the value 'paymentMethod' and 'Control Type \*' with a dropdown menu set to 'Picklist'.

If you would specify a different name, say "paymentCode", then use paymentCode in the param attribute of the query as well:

```
<fetch mapping="logical">
  <entity name="sw_payment">
    <attribute ... />
    <filter>
      <condition attribute="statecode" operator="eq" value="0" />
      <condition attribute="sw_paymentmethod" operator="eq" param="paymentCode" />
    </filter>
  </entity>
</fetch>
```

### Manually controlling the height of the UI Filter area

By default the space required to properly show the UI Filter controls is calculated at runtime. However, it might be possible that this calculation is not what you need. It could be due to different font sizes or other style changes.

The screenshot shows a configuration window with tabs: 'General', 'Query', 'Layout', 'Code', and 'View OnLoad Event'. The 'Layout' tab is active. It contains a code editor with the following XML:
 

```
<grid name="resultset" object="4202" jump="subject" select="1"
  icon="1" preview="1"><row name="result" id="activityid"><cell
  name="actualend" width="100"/><cell name="from" width="125"/><cell
  name="to" width="150"/><cell name="subject" width="250"/><cell
  name="cc" width="100"/><cell name="prioritycode"
  width="100"/></row></grid>
```

 Below the code editor is a field labeled 'Parameter Frame Height' with an empty input box.

To use a different height of the UI Filter area you specify the size in pixels in the Parameter Frame Height field in the Layout tab. To use the calculated height, simply leave the field blank.

## UI Filters in conjunction with the View selection dropdown

If you use multiple views to include a view selection dropdown, then the UI Filters of the first view are displayed. They are not changed when switching to a different view. Make sure to use UI Filters that make sense for all views in this scenario.

For instance, you can build a customer view displaying accounts in the first views and contacts in a second view. Both accounts and contacts have the same address fields, so you can use UI Filters for countries, states and the like. As long as you use the same parameter names in the queries associated to the account and contact view, these views will be filtered based on the same UI Filter control.

## Default values in UI Filters

If you have set a parameter used by a UI Filter with the setParameter method of the SwFilteredView object, then this value will serve as the default for the UI Filter control. However, when directly accessing a Filtered View through a URL rather than using the SwFilteredView object model, you must specify the default parameters in the query string instead. One example when you have to do this is the sitemap. Please read the next chapter for more details.

## Using Filtered Views in the CRM Sitemap

With the introduction of UI Filters it now perfectly makes sense to include a Filtered View in the sitemap. For instance, it is often a requirement to filter data on regions. You may want to show accounts in the United States, Europe or particular countries. Or you want to show open orders from such regions. So far you had to create separate views for each region, which can be quite frustrating and very time consuming. You can use SQL reports to solve this issue, but not everyone likes it and it doesn't feel like CRM.

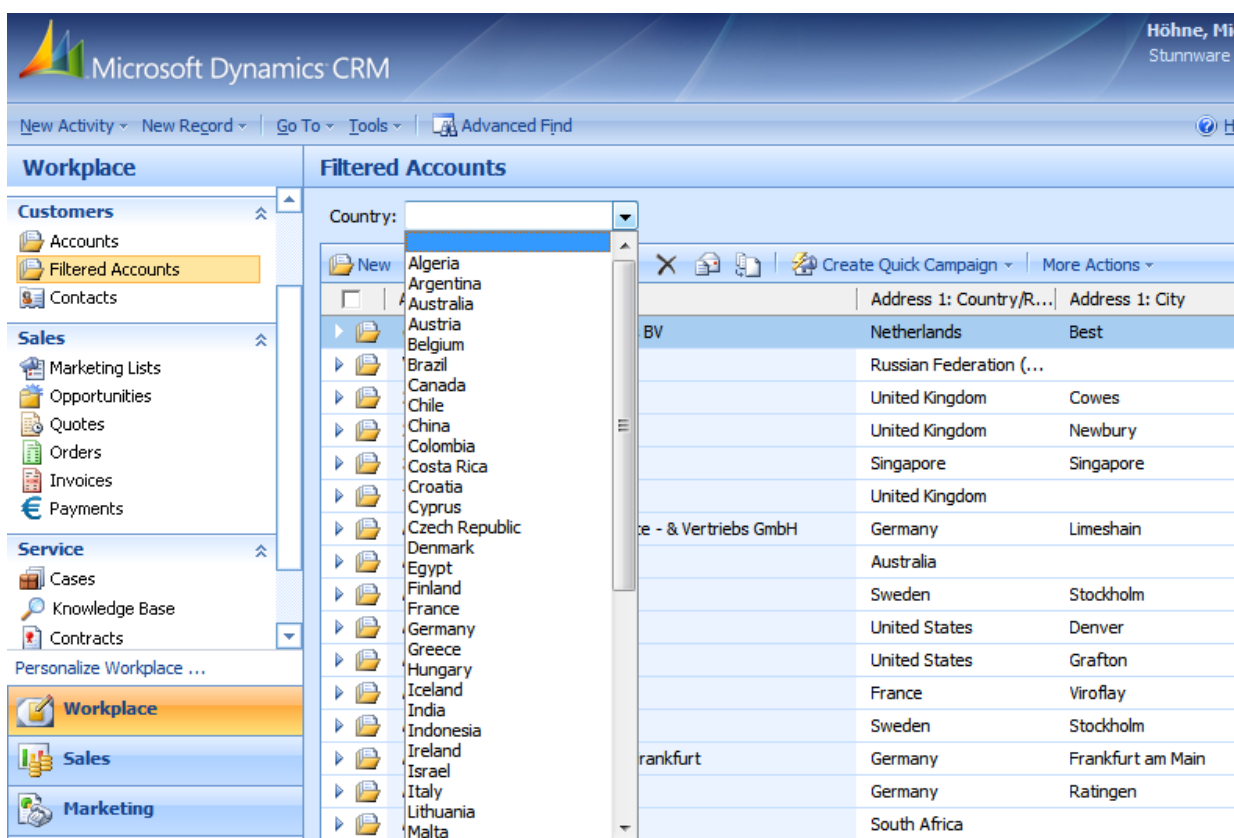
Using a filtered view you can setup a solution very easily. You design it the same way as any other filtered view, but instead of hosting it on a form, you add it to the sitemap:

```
<SiteMap>
  <Area Id="Workplace" ResourceId="Area_Workplace" ShowGroups="true"
Icon="/_imgs/workplace_24x24.gif" DescriptionResourceId="Workplace_Description">
  <Group Id="Customers" ResourceId="Group_Customers"
DescriptionResourceId="Customers_Description">
  <SubArea Id="nav_accts" Entity="account"
DescriptionResourceId="Account_SubArea_Description"/>
  <SubArea Id="nav_FilteredAccounts" Icon="/_imgs/ico_16_1.gif"
AvailableOffline="true" Url="/isv/stunnware.com/FilteredViews/viewFrame.aspx?viewName=Active
Accounts">
    <Titles>
      <Title LCID="1031" Title="Filtered Accounts"/>
      <Title LCID="1033" Title="Filtered Accounts"/>
    </Titles>
  </SubArea>
  <SubArea Id="nav_conts" Entity="contact"
DescriptionResourceId="Contact_SubArea_Description"/>
</Group>
```

The highlighted part of the sitemap is a custom subarea directly accessing a filtered view. All you need to know is the correct URL, which is `/isv/stunnware.com/FilteredViews/viewFrame.aspx?viewName=<name of Filtered View>`. The name is what you have specified in the Filtered View record:

|                  |                 |        |      |                   |
|------------------|-----------------|--------|------|-------------------|
| General          | Query           | Layout | Code | View OnLoad Event |
| <b>General</b>   |                 |        |      |                   |
| Name *           | Active Accounts |        |      |                   |
| Entity *         | account         |        |      |                   |
| Sort Attribute * | name            |        |      |                   |

That 's it. You can configure all other values in the sitemap to your needs. Here's the result of an account list using a UI Filter to select accounts in a specific country:



The screenshot shows the Microsoft Dynamics CRM interface. The left sidebar contains navigation options for Customers, Sales, and Service. The main area displays a 'Filtered Accounts' view with a picklist for 'Country' set to 'Germany'. Below the picklist is a table of account records.

|                       | Address 1: Country/R...  | Address 1: City   |
|-----------------------|--------------------------|-------------------|
| BV                    | Netherlands              | Best              |
|                       | Russian Federation (...) |                   |
|                       | United Kingdom           | Cowes             |
|                       | United Kingdom           | Newbury           |
|                       | Singapore                | Singapore         |
|                       | United Kingdom           |                   |
| ze - & Vertriebs GmbH | Germany                  | Limeshain         |
|                       | Australia                |                   |
|                       | Sweden                   | Stockholm         |
|                       | United States            | Denver            |
|                       | United States            | Grafton           |
|                       | France                   | Viroflay          |
|                       | Sweden                   | Stockholm         |
| rankfurt              | Germany                  | Frankfurt am Main |
|                       | Germany                  | Ratingen          |
|                       | South Africa             |                   |

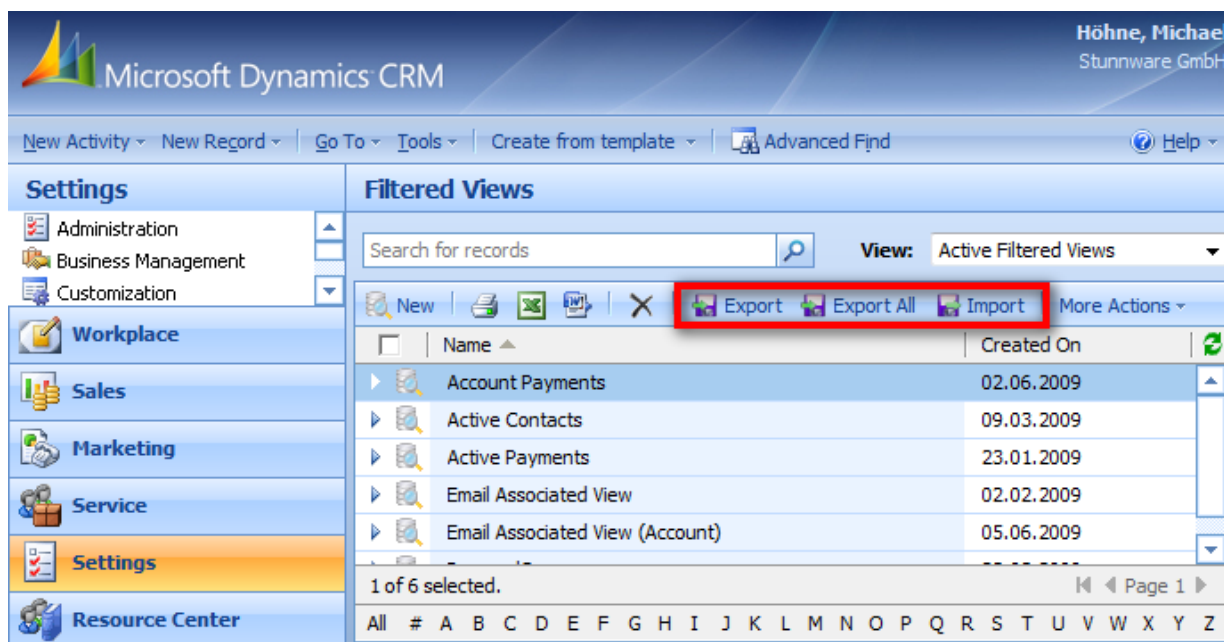
### Specifying default values in the sitemap

To specify default values in the sitemap, you must add the parameters to the Url. For instance, the sitemap example on the last page used the Url [/isv/stunnware.com/FilteredViews/viewFrame.aspx?viewName=Active Accounts](#). Say that you have defined a UI Filter for a picklist field as shown in the above picture. Lets further assume that this parameter is named "country" in the underlying Fetch XML query. To set the parameter to "Germany" initially, you would add the parameter name and value to the Url:

[/isv/stunnware.com/FilteredViews/viewFrame.aspx?viewName=Active Accounts&country=Germany](#). The UI Filter picks up this value and initializes itself accordingly.

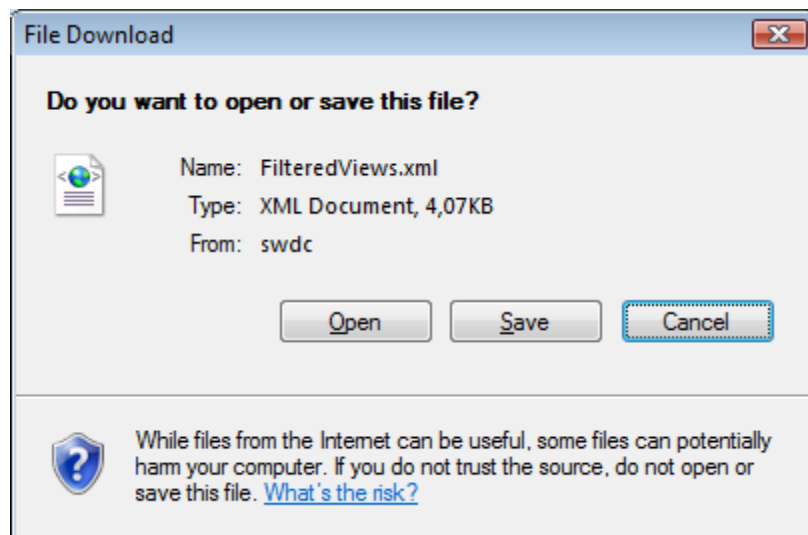
## Deploying Filtered Views

The Filtered Views add-on has an easy-to-use implementation to copy Filtered Lookup view definitions between organizations. It's called export and import.



### Exporting Data

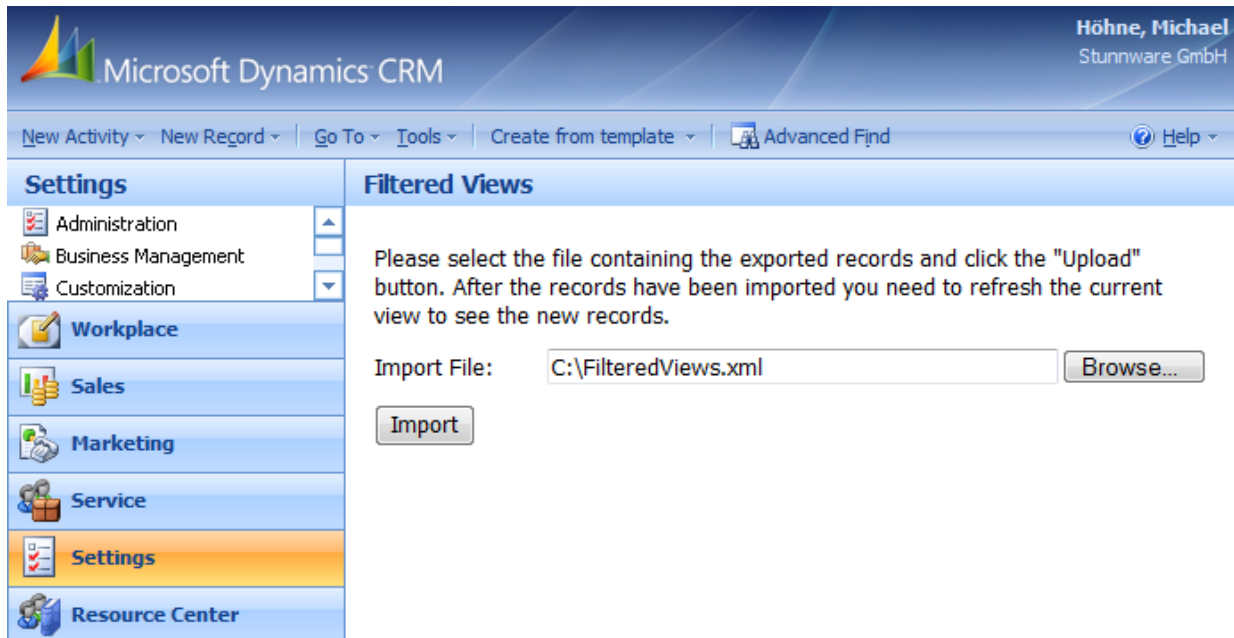
To export one or more Filtered View definitions, you simply select them in the Filtered Views view and click the "Export" button. The selected views are processed at the server and an export file is downloaded to your client:



Save the file to disk and you are ready to import it into another organization. To export all views, choose "Export All" instead. The export file also contains view translations, UI filters and their translations.

## Importing Data

To import the previously exported views into an organization, you again switch to the Filtered Views view and click the "Import button":



Select the file you exported before and then click the Import button to process the file. The import is based on the unique identifiers of the Filtered View records and will overwrite existing records. Be aware that when deleting Filtered View records and immediately trying to re-import them, the import process may fail. This is due to the fact that CRM doesn't delete records instantly. Instead it sets an internal deletion state code flag preventing these items from being shown in the CRM client. Until the CRM Deletion Service finally picks up these items they are still in the database though and therefore an import of a record with the same id fails.

Make sure that the source and target systems use the same Filtered View customizations, otherwise the import may break.