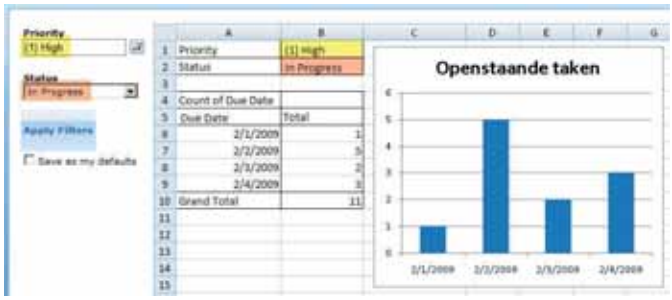
SharePoint filter webparts

DE KRACHT ERVAN AAN DE HAND VAN EEN AANTAL VOORBEELDEN

Ton Stegeman

Microsoft Office SharePoint Server 2007 biedt in de enterprise versie een set van negen zogenaamde filter webparts. Zoals de naam doet vermoeden kunnen deze webparts gebruikt worden om SharePoint content te filteren. De naam “filter webpart” is eigenlijk misleidend. Dit type webpart is veel krachtiger dan de naam doet vermoeden.

Bij filteren zijn altijd twee webparts betrokken. De filter-provider genereert op zijn eigen specifieke manier een filterwaarde en geeft deze via een *webpart connectie* door aan de filterconsumer. Deze consumer verwerkt het binnenkomende filter en voert het filteren uit. Figuur 1 toont een voorbeeld van een filter webpart dat een SharePoint lijst met taken filtert.



FIGUUR 1. FILTER- PROVIDER EN CONSUMER IN ACTIE

Waarom filter webparts?

De filter webparts in SharePoint zijn eigenlijk *filter-providers*. Er zijn 2 typen providers. De eerste groep vraagt de gebruiker een item te selecteren uit verschillende bronnen.

- Choice Filter – Configureer handmatig een aantal opties op het webpart en laat de gebruiker kiezen.
- Text Filter – De gebruiker voert een tekst in.
- Date Filter – De gebruiker kiest een datum.
- SharePoint List Filter – Uit de vooraf geselecteerde lijst kan de gebruiker een item kiezen.
- SQL Server 2005 Analysis Services Filter – Maakt een verbinding met een SSAS cube en laat de gebruiker een item uit de ingestelde dimensie kiezen.
- Business Data Catalog Filter – Laat de gebruiker een selectie maken uit backend systemen die gekoppeld zijn via de BDC.

De tweede categorie filter webparts zijn de zogenaamde *context filters*. Deze webparts bepalen hun filterwaarde op basis van de context waarin ze gebruikt worden en geven deze door aan de verbonden consumers. Context filters zijn:

- Current User Filter – Lees een profiel eigenschap van de ingelogde gebruiker.
- Page Field Filter – Gebruik een van de metadata velden van de huidige pagina als filterwaarde.
- Query String Filter – Lees een filterwaarde uit de query string.

Het woord ‘filter’ in de naam doet deze set van webparts eigenlijk te kort. Ze kunnen namelijk in meer scenario’s dan het filteren van content gebruikt worden. Dit blijkt uit onderstaande lijst van filter-consumers die standaard in MOSS aanwezig zijn:

- List View webpart – het standaard webpart dat SharePoint items op een pagina toont.
- Excel Web Access – gebruik de filterwaarde als workbook url, of gebruik de waarde als named item in een Excel sheet.
- Data Form webpart – gebruikt de binnenkomende waarde om items te filteren, of gebruik deze als parameter voor de datasource (bijvoorbeeld een webservice).
- RSS webpart – stuur de url van een RSS feed, of een parameter voor een RSS feed door via een filter-provider.
- Image webpart – de url van het plaatje wordt bepaald door de filter-provider.
- Filter webpart – de meeste filter webparts accepteren een filterwaarde en gebruiken die als default waarde.

De filter webparts blijken een krachtig hulpmiddel om andere webparts in SharePoint te sturen. Filteren van content is daarbij slechts één van de mogelijkheden. Denk bijvoorbeeld aan een afdelingsportal waar je kunt kiezen voor welke afdeling je de content wilt zien. Vervolgens zie je relevante documenten, taken, nieuws enzovoort. In Excel Services wordt de laatste financiële rapportage getoond. Het Data Form webpart haalt via een webservice relevante data uit CRM. De aansturing van al deze webparts gebeurt vanuit één filter-provider. Met behulp van het Current User Filter wordt de afdeling al default op de juiste afdeling gezet, afhankelijk van de ingelogde gebruiker.

Onmisbare bouwstenen dus voor een goede inrichting van een MOSS-omgeving. Helaas zijn deze filter webparts alleen beschikbaar na aanschaf van de enterprise licentie. Het onderliggende mechanisme is echter onderdeel van– WSS. Dit betekent dat eigen filter-providers en consumers geschreven kunnen worden. Ze kunnen bovendien samenwerken met de standaardset, waarmee de kracht van het filter framework verder vergroot wordt.
Nu gaan we een start maken met het bouwen van een eigen filter pro-

vider. Daarna breiden we die uit, zodat het ook een filter consumer wordt. Deze webparts worden zo opgezet dat ze zowel in MOSS als in WSS bruikbaar zijn. Tot slot laten we zien dat het bouwen van een filter provider voor alleen MOSS eenvoudiger is. Door gebruik te maken van een aantal base classes is veel werk al gedaan.

De filter-provider

De 'ListItemFilter' provider leest items uit een SharePoint lijst en toont deze in een dropdown. In dit artikel vind je slechts de relevante stukken code. De volledige versie (met meer functionaliteit) is te vinden op CodePlex (http://www.codeplex.com/eoffice). Voeg na het aanmaken van een nieuwe class library in Visual Studio een nieuwe class toe. De definitie van de class ziet eruit als in Codevoorbeeld 1.

```
public class ListItemFilter : WebPart, IWebEditable
{
   ListControl _listControl = null;

   [Browsable(false),
   Personalizable(PersonalizationScope.Shared)]
   public string ListUrl { get; set; }
   public ListItemFilter()
   {
this.ChromeType = System.Web.UI.WebControls.WebParts.
PartChromeType.None;
   }
   protected override void CreateChildControls()
   {
      base.CreateChildControls();
      _listControl = new DropDownList();
      _listControl.AutoPostBack = true;
      Controls.Add(_listControl);
   }
   protected override void OnPreRender(EventArgs e)
   {
      base.OnPreRender(e);
      // Populate the drop down
   }

   EditorPartCollection IWebEditable.CreateEditorParts()

   {
```

CODEVOORBEELD 1: DEFINITIE VAN DE PROVIDER CLASS

Gebruik het ASP.Net webpart (System.Web.UI.WebControls. WebParts namespace) als basis. De eerste stap is het toevoegen van een *editorpart*, waarmee de gebruiker het webpart kan configureren. Dit begint met het implementeren van de interface *IWebEditable*. Aan het webpart is een property toegevoegd waarin de url van de geselecteerde lijst wordt opgeslagen. Het WebBrowsable attribuut is toegevoegd om te voorkomen dat het ASP.NET framework de gebruiker ook vraagt deze eigenschap in te vullen.

Voeg aan het project een nieuwe class toe die afstamt van EditorPart. In de constructor van dit editorpart wordt het ID uniek gemaakt met het ID van het webpart. ASP.NET raakt nu niet in de war bij het opslaan van de webpart properties als hetzelfde webpart vaker op dezelfde pagina gebruikt wordt. Zie Codevoorbeeld 2.

```
class ListSelectEditorPart : EditorPart
{
   private TextBox _listUrl;

   public ListSelectEditorPart(string webPartID)
   {
      this.ID = "ListSelectEditorPart" + webPartID;
      this.Title = "List Item Filter Options";
   }

   protected override void CreateChildControls()
   {
      base.CreateChildControls();

      _listUrl = new TextBox();
      Controls.Add(_listUrl);
   }

   public override void SyncChanges()
   {
      EnsureChildControls();
      ListItemFilter webPart = WebPartToEdit as ListItemFilter;
      if (webPart != null)
      {
         _listUrl.Text = webPart.ListUrl;
      }
   }

   public override bool ApplyChanges()
   {
      EnsureChildControls();
      ListItemFilter webPart = WebPartToEdit as ListItemFilter;
      if (webPart != null)
      {
         webPart.ListUrl = _listUrl.Text;
      }
      return true;
   }

}
```

CODEVOORBEELD 2: HET EDITORPART.

De belangrijkste methodes:

- CreateChildControls - voeg de benodigde controls toe aan de Controls collectie.
- SynChanges – Zet de webpart properties op de controls.
- ApplyChanges – sla de geselecteerde waarden op in de webpart properties.

In sterk afgeslankte vorm ziet het editor part er nu uit als in Codevoorbeeld 2.

Het toevoegen van het editorpart is overigens niet specifiek voor een filter webpart. Deze methode is toepasbaar op alle webparts. De volgende stap is om van het webpart een filter-provider te maken. Implementeer hiervoor de interface *ITransformableFilterValues* uit de Microsoft.SharePoint.WebPartPages namespace. Deze interface schrijft een aantal properties voor. Hiermee wordt bijvoorbeeld bepaald of de provider meerdere waarden ondersteunt. Deze eigenschappen bepalen met welke filter-consumers de provider een connectie ondersteunt. Verder schrijft de interface een naam voor de parameter en een functie voor die de filterwaarde(n) uit ons webpart leest. Deze zijn weergegeven in Codevoorbeeld 3.

```
public bool AllowMultipleValues
{
   get { return true; }
}

public string ParameterName
{
   get { return "FilterValue"; }
}

public System.Collections.ObjectModel.ReadOnlyCollection<string>
ParameterValues
{
   get
   {
      List<string> values = new List<string>();
      if (_listControl != null)
      {
         foreach (ListItem item in _listControl.Items)
         {
```

```
            if (item.Selected)
                values.Add(item.Value);
         }
      }
      System.Collections.ObjectModel.ReadOnlyCollection<string>
result =
         new System.Collections.ObjectModel.ReadOnlyCollection<
string>(values);
      return result;
   }
}
```

CODEVOORBEELD 3: IMPLEMENTEER ITRANSFORMABLEFILTERVALUES

Connecties tussen provider en consumer worden gemaakt als standaard ASP.NET webpart connecties. Het enige dat SharePoint daaraan toevoegt is de interface. De provider moet dus een *connection point* definiëren voor de interface ITransformableFilterValues. Voeg, zoals in Codevoorbeeld 4, een methode toe aan het webpart en decoreer deze met het *ConnectionProvider* attribuut.

```
[ConnectionProvider(
"filter value",
"UniqueIDForFilterValueConnection",
AllowsMultipleConnections = true)]
public Microsoft.SharePoint.WebPartPages.ITransformableFilter
Values SetFilterConnection()
{
   return this;
}
```

CODEVOORBEELD 4: VOEG EEN CONNECTIONPOINT TOE AAN HET WEBPART

De tekst 'filter value' is de tekst die de gebruiker ziet in het context-menu van het webpart bij het verbinden van de webparts. De setting AllowMultipleConnections bepaalt dat de provider met meerdere consumers verbonden kan worden. Het webpart is nu een werkende filter-provider. Het kan worden gebruikt om lijsten te filteren of een Excel Web Access rapport te voorzien van data. Tevens kan het een url doorsturen naar het Image Web Part.

De filter-consumer

Het filter-provider webpart zoals hierboven beschreven, moet ook kunnen werken als filter-consumer. Hiermee kan de gebruiker een master-detail filter maken. Stel dat een document library wordt uitgebreid met een Categorie en een Sub-categorie veld. De eerste filter-provider op de pagina toont de beschikbare categoriën. Het webpart wordt verbonden met een tweede ListItemFilter. Dit tweede filter toont de bijbehorende Sub-categorie waarden en geeft de geselecteerde waarde(n) vervolgens door aan de verbonden consumer(s). Tevens kan het *Current User Filter* gebruikt worden om de Categorie uit de profile database te lezen en daarmee de lijst sub-categoriën te filteren. Codevoorbeeld 5 toont hoe het webpart een filter-consumer wordt.

```
[ConnectionConsumer(
   "filter",
   "UniqueIDForListItemFilterConsumer",
   AllowsMultipleConnections = false)]
public void SetFilter(Microsoft.SharePoint.WebPartPages.
IFilterValues filterValues)
{
   _filterValues = filterValues;

   SPList list = GetList(ListUrl);
   if (list !=null && filterValues != null)
   {
      EnsureChildControls();
      List<Microsoft.SharePoint.WebPartPages.ConsumerParameter>
      parameters =
            new List<Microsoft.SharePoint.WebPartPages.
            ConsumerParameter>();
      foreach (SPField field in list.Fields)
      {
         if (!field.Hidden)
         {
            parameters.Add(new Microsoft.SharePoint.
WebPartPages.ConsumerParameter(
                  field.Title,

Microsoft.SharePoint.WebPartPages.ConsumerParameter
Capabilities.SupportsEmptyValue |

Microsoft.SharePoint.WebPartPages.ConsumerParameter
Capabilities.SupportsSingleValue |

Microsoft.SharePoint.WebPartPages.ConsumerParameter
Capabilities.SupportsMultipleValues |

Microsoft.SharePoint.WebPartPages.ConsumerParameter
Capabilities.SupportsAllValue));
         }
      }
      filterValues.SetConsumerParameters
(new ReadOnlyCollection<Microsoft.SharePoint.WebPartPages.
ConsumerParameter>(parameters));
      if (string.IsNullOrEmpty(filterValues.ParameterName) ||
         filterValues.ParameterValues == null ||
         filterValues.ParameterValues.Count == 0)
      {
         FilterandPopulateDropdown();
      }
   }
}
```

CODEVOORBEELD 5: MAAK HET WEBPART EEN FILTER-CONSUMER

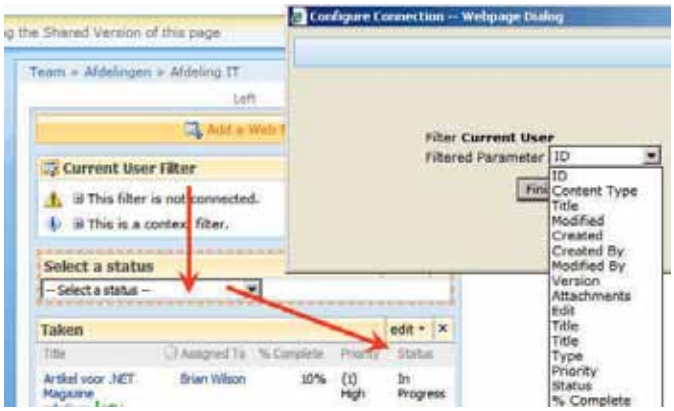
Voeg een methode toe met een parameter van het type *IFilterValues*. Met behulp van het *ConnectionConsumer* attribuut wordt het connection-point toegevoegd aan de class en wordt het webpart een consumer. Als filter-consumer accepteert het webpart slechts één connectie. Zodra de connectie gelegd wordt, wordt aan de gebruiker gevraagd naar welk veld in de lijst de connectie gelegd moet worden. Zie Figuur 2. Het is de taak van de methode SetFilter om een ReadOnlyCollection van *ConsumerParameter* objecten op te bouwen.

De gebruiker kan tijdens het configureren van de connectie nu kiezen uit alle zichtbare velden van de geselecteerde lijst. Aan deze parameters worden één of meerdere capabilities meegegeven. Deze bepalen samen met de properties van de provider de beschikbaarheid van de parameter. De tweede taak van de SetFilter methode is het afhandelen van het binnenkomende filter in de methode FilterandPopulateDropdown. Het webpart construeert een CAML query en voert deze uit. Zie de code op CodePlex voor de volledige implementatie.

Default waarden

Het ListItemFilter kan nu, naast het versturen van filterwaarden ook filterwaarden ontvangen en de keuzelijst daarmee filteren. Om het webpart ook filterwaarden naar andere filter webparts te laten versturen als default waarde, is de interface *IDefaultFilterValue* in Codevoorbeeld 6 geïmplementeerd. Naast het implementeren van de interface wordt er wederom een nieuw ConnectionPoint aan het webpart toegevoegd.

```
#region IDefaultFilterValue Members
public string DefaultValue
{
   get
   {
      if (_listControl != null)
```



FIGUUR 2: INSTELLEN VAN DE CONNECTIE

```
        {
            foreach (ListItem item in _listControl.Items)
            {
                if (item.Selected)
                {
                    return item.Value;
                }
            }
        }
        return null;
    }
}
#endregion

[ConnectionProvider(
"default value",
"UniqueIDForDefaultValueConnection",
AllowsMultipleConnections = true)]
public Microsoft.SharePoint.WebPartPages.IDefaultFilterValue
SetDefaultValueConnection()
{
    return this;
}
```

CODEVOORBEELD 6: VERSTUREN VAN DEFAULT WAARDEN NAAR ANDERE FILTER WEBPARTS

Het laatste ConnectionPoint voor het ListItemFilter is de consumer voor de default values. Hierdoor kan de default waarde van het filter gezet worden met een van de andere filter webparts.

```
[ConnectionConsumer(
    "default value",
    "UniqueIDForListItemFilterDefaultValue",
    AllowsMultipleConnections = false)]
public void SetDefaultFilterValueProvider
(Microsoft.SharePoint.WebPartPages.IDefaultFilterValue provider)
{
    _defaultValueProvider = provider;
}
```

CODEVOORBEELD 7: LISTITEMFILTER WEBPART ACCEPTEERT BINNENKOMENDE DEFAULT WAARDEN

De afhandeling van deze default waarde wordt in de OnPreRender methode gedaan.

Context filters

De context filters zijn een bepaald type filter-providers. Ze lezen een waarde uit de context waarin het webpart wordt gebruikt en geven deze door aan de verbonden consumers. In onderstaand voorbeeld wordt het WebContextFilter webpart gemaakt. Het WebContextFilter leest de ingestelde eigenschap van de site (SPWeb) waarop het webpart is gebruikt. De webpart is bijvoorbeeld toepasbaar in de site definition van een project site. Het webpart leest uit de eigenschappen van de site om welk project het gaat en stuurt hiermee de project rapportage in Excel Services. De volledige code van het webpart is te vinden op CodePlex.
De eenvoudigste manier om een context filter te bouwen is het webpart af te laten stammen van *ContextFilterWebPart* uit de Microsoft.SharePoint.Portal.WebControls namespace. Deze class komt uit de microsoft.sharepoint.portal assembly, die specifiek is voor MOSS. In voorgaande voorbeelden werd een webpart gebouwd dat ook in WSS te gebruiken is. Door ContextFilterWebPart te gebruiken, is het WebContextFilter alleen beschikbaar in MOSS.

```
public class WebPropertyFilter : ContextFilterWebPart,
IWebEditable
{
    private string _currentSiteProperty;

    [Browsable(false),
    Personalizable(PersonalizationScope.Shared)]
    public string CurrentSiteProperty
    {
        get { return _currentSiteProperty; }
        set { _currentSiteProperty = value; }
    }

    protected override string DesignTimeHtml
    {
    get
    {
        string result = string.Empty;
        if (string.IsNullOrEmpty(CurrentSiteProperty))
        {

result = "Please select one of the properties in the web part
toolpane.";
            return result;
        }
        else
        {

result = string.Format(" This webpart sends the value of '{0}'
of the current site to connected webparts.", CurrentSiteProperty);
            return result;
        }
    }

    protected override string[] GetRawFilterValues()
    {

        List<string> values = new List<string>();
        if (!string.IsNullOrEmpty(CurrentSiteProperty))
        {
            string val = GetValueForProperty
(CurrentSiteProperty, SPContext.Current.Web);
            if (!string.IsNullOrEmpty(val))
                values.Add(val);
        }
        if (values.Count > 0)
        {
            return values.ToArray();
        }
        return null;
    }

    public override bool FilterConfigured
    {
        get
        {
            return !string.IsNullOrEmpty(CurrentSiteProperty);
        }
    }

}
```

CODEVOORBEELD 8: EEN CONTEXT FILTER WEBPART, WEBCONTEXTFILTER

De webpart property CurrentSiteProperty bevat de geconfigureerde eigenschap van de site. In het WebContextFilter worden

drie methodes van het ContextFilterWebPart overschreven (zie Codevoorbeeld 8):

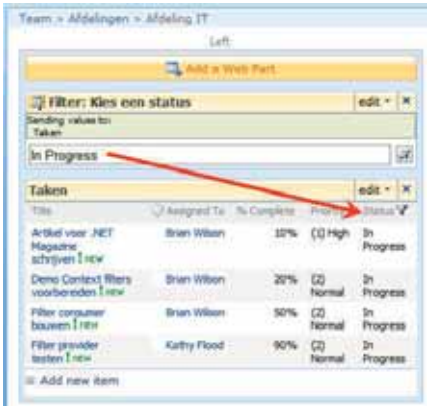
- DesignTimeHtml – Dit is de Html die de gebruiker ziet als de pagina in edit mode weergegeven wordt. Context filters zijn webparts zonder interface; deze tekst helpt de gebruiker de configuratie te interpreteren.
- GetRawFilterValues() – Deze functie geeft een string array terug met de geselecteerde waarde(n). De functie GetValueForProperty leest de waarde van de huidige site.
- FilterConfigured – Door deze property te overschrijven weet het webpart wanneer de configuratie compleet is. Vanaf dat moment zullen waarden worden doorgestuurd naar de verbonden consumers.

Gebruik Filter Actions

Wanneer het zeker is dat het filter webpart alleen in MOSS Enterprise gebruikt gaat worden, is het schrijven van een filter dat input van de gebruiker vraagt eenvoudiger. Dit levert hetzelfde resultaat als het ListItemFilter webpart, met minder code. In plaats van zelf de interface te implementeren en de connecion-points te definiëren, is het mogelijk het webpart af te laten stammen van *UserInputFilterWebPart*. Dit heeft als bijkomend voordeel dat het webpart samen werkt met het *Filter Actions* webpart. Figuur 3 toont dit webpart. Als het op de pagina aanwezig is, worden er een aantal scripts op de pagina geregistreerd. De filter webparts versturen niet direct hun waarden. De gebruiker heeft de mogelijkheid meerdere filters in te stellen. Na een klik op de Apply Filter knop worden de resultaten van alle filter webparts in één keer naar de consumers verstuurd.
Als de SharePoint personalisatie opties beschikbaar zijn, heeft de gebruiker tevens de mogelijkheid de gekozen waarden als default te bewaren. In Codevoorbeeld 9 is de code van het Status webpart uit Figuur 3 weergegeven.

```
public class FilterProviderUser : UserInputFilterWebPart
{
   private DropDownList _selectStatus;

   public FilterProviderUser()
   {
      this.ChromeType = System.Web.UI.WebControls.
WebParts.PartChromeType.None;
   }

   protected override void CreateChildControls()
   {
      base.CreateChildControls();
      _selectStatus = new DropDownList();
      _selectStatus.ID = "SelectStatusFilter";

_selectStatus.Items.Add(new ListItem
("-- Select a status --", string.Empty));
      _selectStatus.Items.Add("In Progress");
_selectStatus.Attributes.Add("onChange",
"SetApplyFiltersToActiveIfPresent();");
      this.Controls.Add(_selectStatus);
   }

   protected override bool AutoPostBack
   {
      get
      {
         EnsureChildControls();
         return _selectStatus.AutoPostBack;
      }
      set
      {
         EnsureChildControls();
         _selectStatus.AutoPostBack = value;
      }
   }

   protected override string[] GetUserSelectedValues()
   {
      this.EnsureChildControls();
      if (!string.IsNullOrEmpty(_selectStatus.SelectedValue))
         return new string[] { _selectStatus.SelectedValue };
      return null;
   }

   protected override void OnPreRender(EventArgs e)
   {
      if (string.IsNullOrEmpty(_selectStatus.SelectedValue) &&
         base.ShouldUseDefaultValues)
      {
         string[] values = base.GetEffectiveDefaultValues();
         if ((values != null) && (values.Length > 0))
            _selectStatus.SelectedValue = values[0];
      }
      base.OnPreRender(e);
   }
}
```

CODEVOORBEELD 9: GEBRUIK HET USERINPUTFILTERWEBPART ALS BASIS



FIGUUR 3: SAMENWERKING MET HET FILTER ACTIONS WEBPART

In de CreateChildControls wordt een client side change event op de dropdown list toegevoegd. Deze roept de javascript functie *SetApplyFiltersToActiveIfPresent* aan. Dit script kleurt de Apply Filters button oranje na het selecteren van een item in de dropdownlist en voorkomt dat de waarde direct verstuurd wordt.
De abstracte base class van het webpart schrijft voor dat de AutoPostback property wordt overschreven. Daarnaast zijn twee extra methodes overschreven:
GetUserSelectedValues – Geeft de geselecteerde waarde terug.
OnPreRender – Haalt de default waarde op. Deze kan worden doorgegeven met een ander filter webpart, of de gebruiker kan een default waarde hebben gekozen en opgeslagen.

Conclusie

Na het toelichten van het belang van filter webparts in SharePoint, is in een aantal voorbeelden laten zien hoe eenvoudig het zelf schrijven van deze webparts is. MOSS heeft een aantal basis webparts die je snel op gang helpen, zoals in de laatste voorbeelden is laten zien. Met iets meer werk kan de kracht van filter webparts ook prima in WSS gebruikt worden. Het behandelde ListItemFilter is daarvan een voorbeeld. Het ASP.NET framework biedt webparts en de connecties daartussen. WSS voegt daar de filter interfaces en een aantal consumers aan toe. MOSS biedt daar bovenop een aantal extra consumers en een set providers.

Ton Stegeman, (ton.stegeman@e-office.com) als technisch architect werkzaam bij e-office. Op zijn weblog (www.tonstegeman.com/blog) is veel SharePoint development gerelateerde content te vinden.