

# DEWD V1.2 DOCUMENTATION

## About

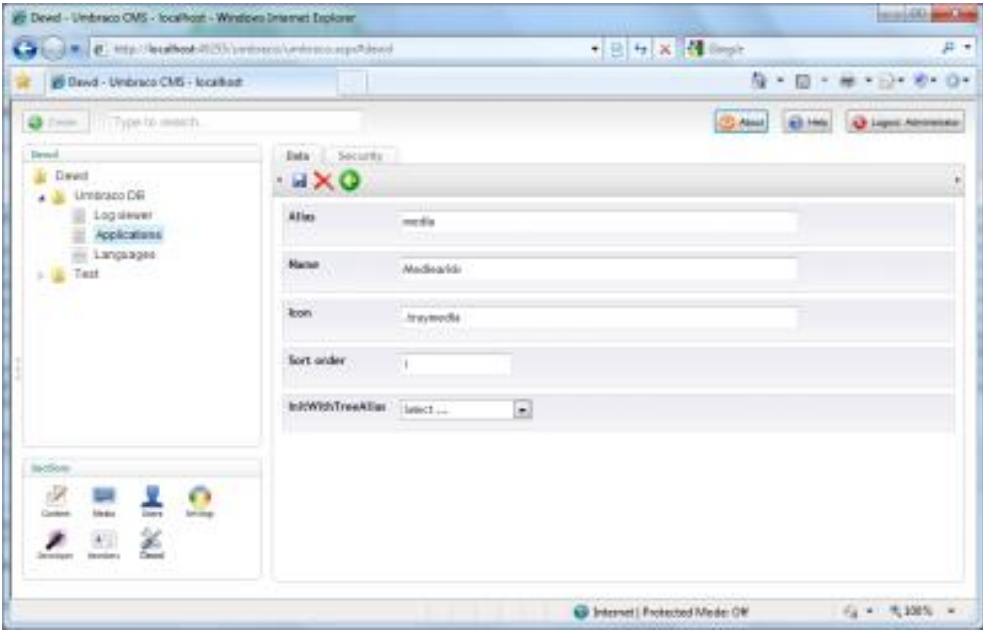
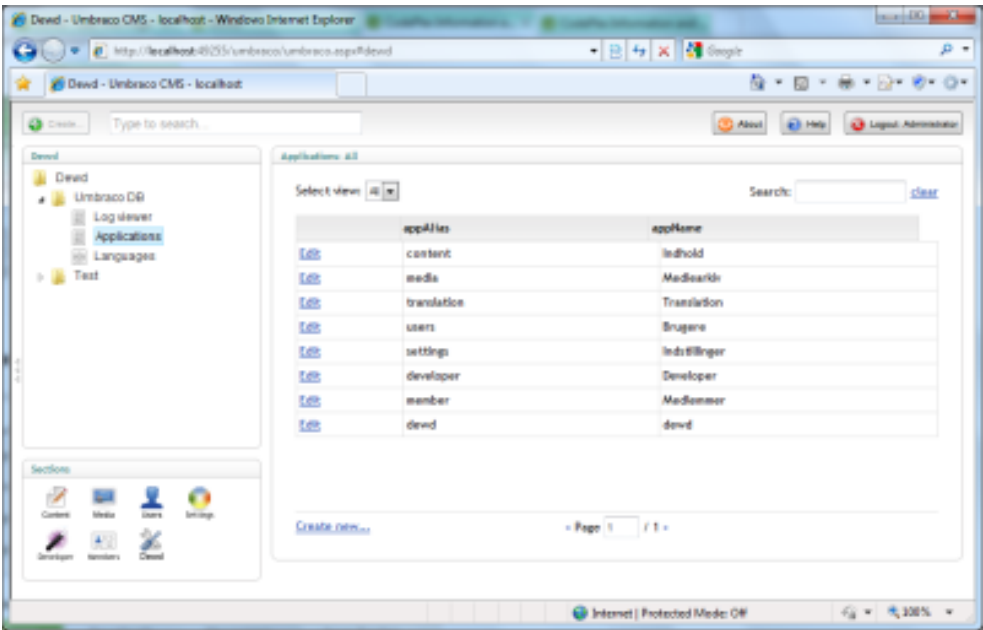
DEWD (pronunciation: dood, [du:d]) is an Umbraco 4.x extension used to edit sequential data such as rows in a SQL server database table. It's meant to allow developers to quickly set up user friendly table editing in Umbraco (eg. for editors) without having to write custom code.

### **DEWD features:**

- ▣ User friendly data editing (mimics Umbracos content editor)
- ▣ Multiple views (grids) of the same table
- ▣ Paging and search
- ▣ Uses build-in umbraco editor controls for editing content
- ▣ Input validation
- ▣ Open architecture allowing you to plug in your own stuff where needed
- ▣ MIT License

Also check out the [FAQ](#).

# Screenshots



# Installation

DEWD is distributed as an Umbraco package and therefore installation is pretty straightforward.

## Installations steps using downloaded package:

- Go to <http://dewd.codeplex.com/releases> and grab the latest package
- Login to Umbraco and go to the Developer-section
- Open Packages -> Install local package
- ❖ If you've already installed DEWD uninstall the existing one under "Installed packages" before continuing (remember to backup your Eksponent.Dewd.config file!)
- Click in "I understand the security risks associated with installing a local package"
- Find the package and hit "Load Package" to begin installation
- Click in "Accept license" and hit the button "Install package"
- Wait a few seconds and you should see a screen saying "Package is installed"
- Grant yourself permissions to use DEWD**
- ❖ Go to Users section
- ❖ Find your user eg. Administrator in Users > Users > Administrator
- ❖ Make sure that "Dewd" is checked in the Sections-field and press Save
- Hit refresh in the browser to reload Umbraco and you should see the DEWD-icon under Sections (bottom left corner)

If the DEWD-icon doesn't show up in the bottom left corner, please check the Umbraco log for what might have gone wrong (umbracoLog table).

# Configuration

DEWD is (currently) configured using an XML file, which is located under **/config/Eksponent.Dewd.config**.

Support for intellisense (using XSD) has been introduced in v1.2. A XSD file is automatically added to the /config folder, which will enable intellisense in the configuration file, if it has the global namespace

"http://eksponent.com/dewd/1.2". Like so:

```
<dewd xmlns="http://eksponent.com/dewd/1.2">
  ...
</dewd>
```

The configuration file contains the following main elements:

- <container>**: folder which may contain other containers or repositories (shown in tree)
- <repository>**: represents a data set with one or more views and an editor (shown in tree)\*
- <view>**: encapsulates functionality for retrieving and displaying data in a list\*
- <editor>**: encapsulates functionality for editing a single data record\*
- <field>**: encapsulates functionality for editing single column/field in a data record\*

\* = The "type" attribute may be used to override the default implementation for these elements, eg. <view type="MyCustomViewType,MyAssembly" />

## Example: Simple log viewer

```
<dewd>
  <repository name="Simple log viewer">
    <primaryKey name="id" />
  </repository>
</dewd>
```

```

    <view name="Latest log entries">
      <sql><![CDATA[SELECT id, logHeader AS Header, logComment AS Comment ORDER BY id
DESC]]></sql>
    </view>
  </repository>
</dewd>

```

#### Example: Direct editing of umbraco language table

```

<dewd>
  <repository name="Languages" icon="mediaFile.gif">
    <primaryKey name="id" />
    <view name="All">
      <sql>
        <![CDATA[SELECT id, languageISOCode, languageCultureName FROM umbracoLanguage]]>
      </sql>
    </view>
    <editor tableName="umbracoLanguage">
      <field title="ISO code" sourceField="languageISOCode" umbracoDataType="Textstring" />
      <field title="Culture name" sourceField="languageCultureName"
umbracoDataType="Textstring" />
    </editor>
  </repository>
</dewd>

```

All elements are described in detail below.

## <container>

### Description

The container element is used to structure the various repositories into a tree structure which will make sense to the end user. A container is shown as a folder in the tree navigation on the left side of the screen

### Attributes

- **name:** The name of the container shown in the left tree navigation. This attribute is required.
- **drt:** Default repository type (DRT) allows an IRepository implementation to be specified for a subtree of containers and repositories.
- **connection:** Only relevant for TableRepository; allows a custom connection string to be specified for a subtree of containers and repositories.
- **url:** Allows a custom url to be displayed on the right side, when user clicks on the container in the tree.
- **hidden:** Allows a container to be hidden from the tree (but still accessible eg. for a custom module).

## <repository>

### Description

A repository corresponds to a single set of data (eg. a SQL data table), which has one or more views and in most cases also an editor.

### Attributes

- **name:** The name of the repository shown in the left tree navigation. This attribute is required.
- **icon:** Optional file name of icon shown in the left tree navigation.
- **type:** Allows the default implementation to be overridden. The specified type must implement `Ekspontent.Dewd.Repositories.IRepository`.
- **connection:** Only relevant for TableRepository; allows a custom connection string to be specified for a repository.
- **url:** Allows a custom url to be displayed on the right side, when user clicks on the repository in the tree.

- **hidden:** Allows a repository to be hidden from the tree (but still accessible eg. for a custom module).

## <primaryKey> (sub-node to <repository>)

### Description

Specifies the primary key used to uniquely identify a single record in the data set. When a user clicks on a row in the view grid, the primary key of that row is given to the editor which in turn can look up the row in the data set and retrieve its field values.

### Attributes

- **name:** The name of the primary key to be used. Should correspond to the name of the primary key in the view and the primary key of the editor data source. This attribute is required.
- **manual:** If the value of the primary key column is entered manually instead of being autogenerated by the data store, this parameter should be set to: true

## <view> (sub-node to <repository>)

### Description

Defines which data is retrieved from the data store and how the data is shown. When the user clicks on a repository in the left tree navigation, the default view (first view-childnode of <repository>) is shown on the right side. The view can support paging and searching. If an editor is defined as well, the view will show "Edit"-links in the first column of the grid as well as an "Create new..."-button.

### Attributes

**name:** The name of the view which is shown in the "Select view"-dropdown at the top of the view page. This attribute is required.

**type:** Allows the default implementation to be overridden. The specified type must implement `Eksponent.Dewd.Views.IView`.

**controlType:** Allows the default list/grid control implementation to be overridden. The specified type must implement `Eksponent.Dewd.Controls.View.IViewControl`.

## <sql> (sub-node to <view>)

### Relevant to implementations of IView:

TableView

### Description

Contains the SQL to execute against the database to retrieve the data, which is shown on the view page (statement is written inside the sql-node).

## <parameter> (sub-node to <view>)

### Relevant to implementations of IView:

TableView

### Description

May be used to add SQL parameters for the sql-node. Uses the same value-getter logic as defaultValue-node.

```
<view name="Todays entries">
  <sql>SELECT * FROM umbracoLog WHERE CAST(Datestamp AS date)=@today</sql>
  <parameter name="@today" get="today" />
</view>
```

### Attributes

**type:** Allows the default implementation to be overridden. The specified type must implement `Eksponent.Dewd.Fields.ValueGetters.IValueGetter`.

#### Attributes for Standard-type

**get:** Sets a special value; the following attribute values may be used: null, dbnull, today, now, now-utc, guid.

**targetType:** Allows type conversion to take place. Default namespace is System.

### <columns> (sub-node to <view>)

Relevant to implementations of `IViewControl`: **ScrollingGrid**

#### Description

May contain a comma separated list of column names to display in the grid or a list of column-subnodes, which defines how each column should be displayed.

### <column> (sub-node to <columns>)

Relevant to implementations of `IViewControl`: **ScrollingGrid**

#### Description

Allows simple configuration of each column in the grid including formatting, header.

#### Attributes

**field:** A single field name or a comma separated list of field names to display. If multiple fields are specified format-attribute should be used to format the output.

**format:** A `String.Format` to apply to the data value of each field. Optional.

**title:** Header/title of the column. Optional.

**width:** Width of the column. Optional.

**nowrap:** Whether to disable text wrapping for the cells. Optional.

### <snippet> (sub-node to <view>)

Relevant to implementations of `IViewControl`:

Snippet

#### Description

Experimental control type which allows the developer to include an ASP.NET control declaration, which is then used as grid.

#### Example

```
<view name="Errors" controlType="Snippet">
  <sql>
    <![CDATA[
      SELECT umbracoLog.id, logHeader AS Header, LEFT(logComment,100) AS Comment, userName
      AS [User] FROM umbracoLog
      LEFT JOIN umbracoUser ON umbracoUser.id=umbracoLog.userId
    ]]>
  </sql>
  <snippet>
    <![CDATA[
      <dewd:ScrollingGrid runat="server" AutoGenerateColumns="false">
        <Columns>
          <asp:BoundField HeaderText="ID" DataField="id" />
          <asp:BoundField HeaderText="Category" DataField="Header" />
        </Columns>
      </dewd:ScrollingGrid>
    ]]>
  </snippet>
</view>
```

```

        <asp:BoundField HeaderText="Text" DataField="Comment" />
    </Columns>
</dewd:ScrollingGrid>
]]>
</snippet>
</view>

```

## <editor> (optional sub-node to <repository>)

### Description

Defines how a record in the data set should be edited eg. which fields should be shown to the editor.

### Attributes

**tableName:** The name of the database table which the record to edit is located in. **Relevant implementations:** TableEditor.

**type:** Allows the default implementation to be overridden. The specified type must implement EkspONENT.Dewd.Editors.IEditor. Very relevant if you want to have special save/delete events for your editor.

## <field> (optional sub-node to <editor>)

### Description

Defines how a single column/property in the data source should be handled by the editor eg. which umbraco editor control should be used to edit the content.

### Examples

```

<editor tableName="DewdTypeTable">
  <field title="Title" sourceField="title" umbracoDataType="Textstring" />
  <field title="Numeric" type="NumericField" sourceField="testNumeric"
umbracoDataType="Textstring" format="{0:0.00}" />
  <field title="Date" sourceField="testDate" umbracoDataType="Date Picker" />
  <field title="Internal guid" sourceField="testGuid"><nullable /><defaultValue get="guid"
/></field>
</editor>

```

### Attributes

**title:** Title which is displayed to the left of the editor control.

**sourceField:** Column/property/field name in the data set to get and set value to.

**umbracoDataType:** Umbraco content editor control (input field) displayed. Can be either named data type (eg. Textstring) or specified via ID. Optional attribute.

**tabTitle:** Name of the tab to place the field under.

**type:** Allows the default implementation to be overridden. The specified type must implement EkspONENT.Dewd.Fields.IField.

**format:** Allows a String.Format to be applied to the value before being sent to the editor control. Useful for number formatting etc., so on a floating point column you could use format="{0:0.00}" to limit value displayed to two decimals.

**culture:** Allows a custom culture to be used when formatting the value using the string format specified in the format-attribute.

**defaultValue:** Allows a default value to be specified for new rows.

**readonly:** Tells the editor to ignore the field, when updating the data source (eg. for calculated or auto-generated columns).

## <nullable> (optional sub-node to <field>)

**Relevant to implementations:**

TableView

**Description**

Specifies that the field should save a DbNull value to the database field, when a certain value is entered in the field.

**Attributes**

**whenValueEquals:** Value that triggers the DbNull value. An empty string is the default trigger.

**<listControlDataSource> (optional sub-node to <field>)****Relevant to umbracoDataType(s):**

Dewd Dropdown, Dewd DuoSelect

**Description**

Allows list items on certain editor controls (see "Relevant to umbracoDataType") to be generated dynamically. This is highly relevant for fields having relations to other database tables or entities.

The default implementation (TableListControlDataSource) allows a sql-subnode to retrieve rows from a SQL statement and display these as list items. Example:

```
<listControlDataSource>
  <sql><![CDATA[SELECT id, userName FROM [umbracoUser] ORDER BY id]]></sql>
</listControlDataSource>
```

**Attributes**

**type:** Allows the default implementation to be overridden. The specified type must implement Eksponent.Dewd.Controls.IListControlDataSource.

**includeEmpty:** When set to **true** this automatically inserts an empty item (with the text: Select ...) at the beginning of the list items.

**emptyValue:** If "includeEmpty" is set to true, this specifies the value that value that should be returned when the first (Select...) list item is selected by the user.

**<select> (optional sub-node to <field>)****Description**

Element allowing configuration of the **DuoSelect** editor control (data type).

**Attributes**

**sortable:** When set to true, allows the user to sort the select values inside the DuoSelect editor control.

**<linkingField> (optional sub-node to <field>)****Description**

Element allowing configuration the field type **TableLinkingField**, which is used to support simple many-to-many relationships between tables.

**Example:**

```
<field type="TableLinkingField" title="Categories" umbracoDataType="Dewd DuoSelect">
  <select sortable="false" />
  <listControlDataSource>
    <sql><![CDATA[SELECT ID, CategoryName FROM Categories]]></sql>
  </listControlDataSource>
  <linkingField linkingTable="ProductCategoryLinking" localKey="ProductID"
foreignKey="CategoryID" />
```

```
</field>
```

#### Attributes

**linkingTable:** Specifies the name of the linking table which contains two columns of keys.

**localKey:** Name of the key/column in the linking table which corresponds to the ID (primary key) of the row being edited.

**foreignKey:** Name of the key/column in the linking table which corresponds to the values returned from the editor control. In the example above this would be ID in the Categories table.

### <caption> (optional sub-node to <field>)

#### Description

If specified the value will be displayed as text under the field name.

### <validation> (optional sub-node to <field>)

#### Description

Specifies what kind of validation should be applied to the field. A field may contain multiple validation sub-nodes which are evaluated in the order they appear in the configuration file.

The default implementation (EkspONENT.Dewd.Fields.Validators.Required) simply checks whether the field contains a value (string has length greater than zero).

#### Attributes

**type:** Allows the default implementation to be overridden. The specified type must implement EkspONENT.Dewd.Fields.Validators.IValidator.

**errorMessage:** Text which is displayed in the validation summary if validation fails, eg. "Error in field {0}" (first string argument is replaced by the field name, all IValidator implementations should support this).

#### Attributes for Required-type

**required:** Specifies whether a field value is required (should always be set to true)

**trim:** Specifies if the value is left/right-trimmed for whitespace

#### Attributes for RegEx-type

**pattern:** Specifies the RegEx pattern to match against. If the specified pattern does not match the value, the validation fails.

### <defaultValue> (optional sub-node to <field>)

#### Description

Allows a default value to be specified for the field. Optionally the field can revert to the default value eg. if a certain value is specified by the user. The Standard implementation allows a value to be specified as text in the defaultElement or in the value-subelement. Please note that for simple default values you can choose to use the defaultValue attribute on the field node instead.

#### Example:

```
<field title="A number" sourceField="myNumber" umbracoDataType="Textstring" defaultValue="0" />
<field title="Sort order" sourceField="sortOrder" umbracoDataType="Textstring">
  <defaultValue>0</defaultValue>
</field>
<field title="Date" sourceField="createDate" umbracoDataType="Date Picker">
  <defaultValue get="now" />
</field>
```

## Attributes

**type:** Allows the default implementation to be overridden. The specified type must implement Eksponent.Dewd.Fields.ValueGetters.IValueGetter.

## Attributes for Standard (default implementation)

**get:** Sets a special value; the following attribute values may be used: null, dbnull, today, now, now-utc, guid.

**targetType:** Allows type conversion to take place.

## <revertTrigger> (optional sub-node to <defaultValue>)

### Description

Allows the field to revert to the default value, if a specific value is specified in the field (eg. in an editor control). One or more revertTrigger nodes may be specified. The default implementation simply checks whether the field value is null or an empty string.

### Example:

```
<field title="Sort order" sourceField="sortOrder" umbracoDataType="Textstring">
  <defaultValue>
    <value>0</value>
    <revertTrigger type="IsNullOrEmpty" /> <!-- equal to <revertTrigger /> -->
    <revertTrigger type="Regex" pattern="^\d+$" negate="true" />
  </defaultValue>
</field>
```

## Attributes

**type:** Allows the default implementation to be overridden. The specified type must implement Eksponent.Dewd.Fields.Criteria.ICriterion.

**negate:** Boolean. Apply a NOT operator to the criterion thus reversing its result.

## Attributes for IsNullOrEmpty

**trim:** Trims the input value before checking whether it's empty.

## Attributes for Regex

**pattern:** Allows a regular expression to determine whether the default value should be used.

## <tweak>

### Description

Special element for tweaking the configuration file as it is loaded. The main purpose of the tweak-element is to include other config-files and get rid of redundancy inside the configuration file. At present the tweak element is able to **clone** or **remove** elements.

### Clone syntax

```
<tweak do="clone" select="../someElementToClone" />
<tweak do="clone" external="/config/dewd/myconfig.config" />
<tweak do="clone" external="/config/dewd/*.config" select="./container[@name='needed
elsewhere']" />
```

### Attributes:

**external:** Path of an external configuration file to be loaded and included. Path may include (end with) wildcard.

**select:** XPath which filters the elements to be cloned. If no external file is loaded, the xpath is run against the local document (using tweak-node as source).

### Remove syntax

```
<tweak do="remove" select="../someElementToRemove" />
```

### Attributes:

**select:** XPath which filters the elements to be removed with tweak-node as source for xpath selection.

**Example: using clone+remove to reuse validation for multiple fields**

```
<editor tableName="umbracoLanguage">
  <!-- in this example validationGroup is just an arbitrary chosen element name to group a
  number of validation-elements. -->
  <validationGroup name="culture">
    <validation required="true" />
    <validation type="Regex" pattern="^[a-zA-Z]{2}\-[a-zA-Z]{2}$" />
  </validationGroup>

  <field title="CultureName" sourceField="languageCultureName" umbracoDataType="Textstring">
    <tweak do="clone" select="../../validationGroup[@name='culture']/*" />
  </field>
  <field title="ISO code" sourceField="languageISOCODE" umbracoDataType="Textstring">
    <tweak do="clone" select="../../validationGroup[@name='culture']/*" />
    <tweak do="remove" select="../validation[@type='Regex']" /> <!-- whups, no Regex for
  this field -->
  </field>
</editor>
```

## Web.config AppSettings

A few settings are available for the appSettings section of web.config, mostly for debugging purposes.

```
<configuration>
  <appSettings>
    <add key="Ekspont.Dewd.Debug" value="True" />
    <add key="Ekspont.Dewd.UseBaseDataReflectHack" value="True" />
    <add key="Ekspont.Dewd.DisableNamespaceRemoval" value="True" />
  </appSettings>
</configuration>
```

**Ekspont.Dewd.Debug:** Will enable debugging. Primary effect is that the parsed configuration file is saved to /data/dewd-debug.config.

**Ekspont.Dewd.UseBaseDataReflectHack:** Will make the ContentControlContainer attempt to override the normal DefaultData implementation with NoSqlDefaultData (using reflection, very ugly!), which disables Umbraco database changes, when getting or setting the Value property on a data type. May help with a failing data type.

**Ekspont.Dewd.DisableNamespaceRemoval:** Dewd will by default try to remove all namespaces from the xml configuration file, this can be disabled using this setting.