FRED ADMINISTRATION GUIDE

Document reference: Fred/Administration Guide/3.1
Version: 3.1
Document revision: 0.9
Release date: 13/03/2015

COPYRIGHT

TABLE OF CONTENTS

1. INTRODUCTION

The Alfred Desktop application has a client part and a backend part. The backend part is a module for Alfresco, that enables the client to talk to the Alfresco repository. This document describes how Alfred Desktop can be configured at the server side for different use-cases.

If you are looking for documentation on how to install the Alfred Desktop backend, please consult the Alfred Desktop Backend Installation Guide

2. PREREQUISITES

A functional Alfresco installation with:

1. The Alfred Desktop .amp install
2. A minimal server configuration config.js

You should be able to connect with the client to the Alfresco repository at this point.

Please consult the Alfred Desktop Backend Module Installation Guide to setup the prerequisites.

3. PRODUCT BRANDING PREREQUISITES

In case of an upgrade to a version with updated product branding, the following steps are required to clean the Windows caches:

- Icon cache
    - Disk Cleanup (*thumbnails*)
- Thumbnail cache
    - Delete all thumbnailcache_*.db in %LocalAppData%\Microsoft\Windows\Explorer
- MUI cache
    - Delete all entries containing "Fred" from
        - HKEY_CLASSES_ROOT\Local Settings\Software\Microsoft\Windows\Shell\MuiCache
        - HKEY_CURRENT_USER\Software\Classes\Local Settings\Software\Microsoft\Windows\Shell\MuiCache
        - HKEY_CLASSES_ROOT\Local Settings\MuiCache (*possible subtree of keys*)
        - HKEY_CURRENT_USER\Software\Classes\Local Settings\MuiCache (*possible subtree of keys*)

Upon reboot, the cache will be populated with the new product branding. This might include new logos, new product name, etc.

4. BASIC CONFIGURATION

4.1. OVERVIEW

Alfred Desktop uses a JSON-style server side configuration file, *config.js*.

This config file should be created in *$TOMCAT/shared/classes/alfresco/extension/templates/webscripts/eu/xenit/config/* and named *config.js*

4.1.1. KEYWORDS

Below you can find a table with a full list of supported configuration-keywords:

Parameter Name	Remarks	Type	Description
namespace	required	*string* or *string[]*	All types/properties in this namespace/model are considered business-related-properties and can be read and edited by default. You can fine tune the metadata-panel & search-options using the forms-configurations. See Chapter 4 on "Metadata Configuration"
rootdoctype	required	*string*	This is the root of your document type hierarchy. Every document created/uploaded through Alfred Desktop will have a document of this type by default. The user can specify a more specialized type.
rootnoderef	required	*string*	Indicates the client where to find the Company Home folder. This can be used to limit certain user groups into a substructure of your Alfresco Repository. Example: the members of the Alfresco group "sales" should only see the /Company Home/Sales folder. (This is an experimental feature)
template-metadata-mode	optional	*string*	In Alfred Desktop you can create new documents from a template from Alfresco. This property has two possible values: • "MetadataFirst" [default & recommended] – when a user chooses a template, Alfred Desktop will ask for the document name (and metadata) first, before actually creating the document. • "CopyTemplateFirst" – when a user chooses a template, Alfred Desktop will create the document first and ask the user for the name and metadata afterwards. This is useful if you have rules set up in Alfresco that automatically insert metadata based on folder-context.

Parameter Name	Remarks	Type	Description
externalAlfrescoUrl	optional	*string*	Alfred Desktop has the option to copy document & folder links into the clipboard to share this link with others. If your Alfresco is not reachable over the internet at the same URL you are using in your intranet, you can use this property to set a different base url. Example: you connect with Alfred Desktop to your Alfresco server in your company network using the url http://intra.alfresco.mycompany.com/alfresco but if you want to send a link to a document over email, you should use the publically accessible http://alfresco.mycompany.com/alfresco
useNtlmLinks	optional	*string*	Supported values: “False” (default), “True” Alfred Desktop has the option to copy documents & folder links into the clipboard to share this link with others. If you enable this, Alfred Desktop will use the /wcservice variant of the link, which has support for SSO (NTLM, Kerberos)
useShareLinks	deprecated	*string*	Is Obsolete since Alfred Desktop 3.5, the value of this will have no more effect.
extended	optional	*object*	Object to hold extended-configuration-properties. This is the appropriate place to add custom configuration properties to support customizations. See topics on: • Incomplete Metadata Policies • Report a problem • Digital Signature • Take offline • Take Offline
minReqClientVersion	optional	*string*	The version number of the minimum allowed Alfred Desktop client. Default value is "2.0.0" for Alfred Desktop backend 3.x.
maxReqClientVersion	optional	*string*	The version number of the maximum allowed Alfred Desktop client. (ex. "3.2" all clients with versions in the 3.2 range or below will be supported).
documentmodel	deprecated	*string* or *string[]*	The fully qualified name (qname) of your business model. This property is deprecated, but can be present to obtain backwards compatibility with older Alfred Desktop clients (v1.6-2.1).

4.1.2. GROUP CONFIGURATION

Note: This feature is deprecated since Fred 3.3 and will be removed in one of the next releases.

Additionally, this configuration can be mapped to Alfresco-groups. If you have a Department-A group and a Department-B group, they can have a different configuration applied.

When a user connects through Alfred Desktop, it looks up all the groups a user is a member of. The first group-configuration it can find in config.js is loaded for this user. If no configuration for any of his groups is found, it falls back to the configuration named “default”.

4.2. EXAMPLE CONFIGURATIONS

4.2.1. GENERIC MINIMAL CONFIGURATION

This is a minimal config.js configuration – from the Backend Installation Guide. It uses the standard Alfresco content-model. All new documents will have the type cm:content.

This is the very minimum configuration. It doesn’t use custom metadata/dictionary-models.

```
var configByGroup = {
    "default": {
        "namespace" : "",
        "rootdoctype" : "{http://www.alfresco.org/model/content/1.0}content",
        "rootnoderef" : companyhome.nodeRef.toString()
    },
};
```

4.2.2. DEFAULT CUSTOM CONFIGURATION

If you want to use Alfresco/Alfred Desktop to its full potential, you want to use a custom dictionary model.

An example dictionary model is provided in Annex A in the Backend Installation Guide. The example configuration is specific for this model. Please adapt this configuration to your own dictionary model(s).

```
var configByGroup = {
    "default": {
        "namespace"          : ["http://www.xenit.eu/fred/example/model/0.1"],
        "rootdoctype"        : "{http://www.xenit.eu/fred/example/model/0.1}document",
        "rootnoderef"        : companyhome.nodeRef.toString()
    }
};
```

Using this configuration, Alfred Desktop will use the dictionary model in Annex A as the business-model because the configuration defines namespaces as a list with one element:

http://www.xenit.eu/fred/example/model/0.1

This implicates that Alfred Desktop will ask the user to provide metadata properties like http://www.xenit.eu/fred/example/model/0.1}customer_name where appropriate.

Setting the rootdoctype to {http://www.xenit.eu/fred/example/model/0.1}document indicates that all document types an end-user should be able to create/search, inherit from this type.

4.2.3. GROUP CONFIGURATIONS

Note: This feature is deprecated since Fred 3.3 and will be removed in one of the next releases.

If, for example, a specific group like the (fictive) sales department uses Alfresco through Alfred Desktop, they shouldn't be bothered with Invoice-type documents.

They work – meaning: create documents – with the following types:

1. Sales (fred:sales in the example dictionary model)
   1.1. RFI (fred:rfi inherits from fred:sales)
   1.2. RFQ (fred:rfq inherits from fred:sales)
   1.3. Proposal (fred:proposal inherits from fred:sales)
   1.4. Contract (fred:contract inherits from fred:sales)

If this group creates a new document, they would like to choose between these document types. The Invoice type is just clutter.

This can be accomplished in the Alfred Desktop configuration by setting rootdoctype to a different document type in a group-specific configuration:

```
var configByGroup = {
    "Sales": {
        "namespace"          : ["http://www.xenit.eu/fred/example/model/0.1"],
        "rootdoctype"        : "{http://www.xenit.eu/fred/example/model/0.1}sales",
        "rootnoderef"        : companyhome.nodeRef.toString()
    },

    "default": {
        "namespace"          : ["http://www.xenit.eu/fred/example/model/0.1"],
        "rootdoctype"        : "{http://www.xenit.eu/fred/example/model/0.1}document",
        "rootnoderef"        : companyhome.nodeRef.toString()
    }};
```

5. METADATA CONFIGURATION

5.1. INTRODUCTION

Alfred Desktop presents “forms” with data & metadata in views and panels. Alfred Desktop has extensive configuration possibilities for displaying and working with the metadata in your Alfresco system.

The style of configuration is heavily inspired on the Share Forms configuration in Alfresco.
Alfred Desktop comes with a default configuration set for metadata and works straight out of the box, but you might want to configure how your custom dictionary model is displayed and used.

5.2. WHERE ARE FORMS USED ?

Forms are used in the following components of Alfred Desktop:

5.2.1. VIEW & EDIT METADATA

The metadata panel is a dynamic panel that evaluates the selected document or folder, looks at the document type, available aspects and properties and generates a customized form.
This panel supports three different modes: (1) view mode – the default; (2) edit mode – to edit metadata; and (3) create mode – when creating new documents from a document template.

FIG. 1. METADATA PANEL IN EDIT-MODE

5.2.2. SEARCH PROPERTIES

The list of properties in the search-dropdown (both in the quick-search as in the second tab) can be configured using a forms-configuration, which simply lists all the properties you would like to have available in the dropdown.

FIG. 2. SEARCH PROPERTIES

5.2.3. COLUMNS IN THE FOLDER VIEW

In the folder view, the user can select properties to display or hide in a column in the view. This menu can be fully configured, including support for submenus.

FIG. 3. COLUMNS IN FOLDERVIEW

5.2.4. WORKFLOWS & TASKS

Alfred Desktop has support for Alfresco jBPM & Activiti Workflows’. Default configurations are bundled with Alfred Desktop for the standard workflows in Alfresco. You can extend the default configurations with customizations for your custom workflows.

FIG. 4. WORKFLOW TASK/FIGCAPTION

5.3. TERMINOLOGY

The forms can be configured in specific JSON files. See paragraph 4.4. An empty form configuration template looks as following.

```
[
  {
    "Id": "",
    "Evaluator": "",
    "Forms": [
      {
        "FieldVisibility": [
          {
            "Id": "",
            "Mode": ["Create","View","Edit"],
            "Show": true|false,
            "Force" true|false:
          }
        ],
        "sets":
        [
          {
            "Id": "",
            "Appearance" : "",
            "Label" : "",
            "Label-id" : "",
            "Template" : "",
            "Fields":
            [
              {
                "Id": "",
                "Template" : "",
                "Parameters" : {
                    "param" : "value"
                },
              }
            ],
            subsets: []
          }
        ]
      }
    ]
  }
]
```

In these JSON files, the following terms can be found:

5.3.1. FORM - PROPERTIES

The main properties are specific for the form. They contain the id and the Evaluator.

- Id: the id of the form, example activiti$activitiAdhoc for the adhoc workflow.
- Evaluator: the evaluator string that is used. Example string-compare

5.3.2. FORM > FIELD SET

Field sets groups fields together under a section (with an optional template). Each set has a mandatory property Id, a list of Fields and a few additional optional properties.

- Id - [mandatory] *string*: the id of the set.
- Fields - *Field*-array: the fields that are included in the set.
- Label - *string*: the label that the set will get. This is displayed in Alfred Desktop.
- Label-id - *string*: the id of the label, to support localisation. *Currently not yet in use.*
- Template - *string*: the template the set needs to use. This can be used to alter how a field-set is displayed. Valid values are provided below.
- Parameters - *object*: this is a key-value object (*string* to *string*) with parameters that are passed to the Field Set template. Supported parameters depend on the template-type and are documented below.

The Template property supports the following values:
The Template property supports the following values:

- gui.controlfactory.fieldset.centered - the default fieldset in the metadata panel
- gui.controlfactory.fieldset.group - draws a border to 'group' fields together.
  - Parameter collapsible: true|false
  - Parameter start-collapsed: true|false
  - Parameter centered: true|false
- gui.controlfactory.fieldset.1-column
- gui.controlfactory.fieldset.2-column
- gui.controlfactory.fieldset.3-column

The centered-fieldset is the default. It right-aligns the field-labels and left-aligns the field-values around the center. The group-fieldset draws a border around the Fieldset, with an optional title provided by the Label attribute. The column-variants put fields in 1 or more columns. These are mainly used in the workflow-panels.

5.3.3. FORM > FIELD SET > FIELD

A *Field* represents a single property/field/entity/menuitem.

- Id - [mandatory] *string*: The id of the field. Example: cm:title.
- Mode[1]: There are tree modes in Alfred Desktop: View, Edit and Create. This specifies in which modes the field needs to be visible. Example Mode: ["View", "Edit"] will only be visible when the form is viewed, and edited. But not when it is used for creation. When a field is not present in the fieldvisibility array, but it is in the sets. All modes are used by default.
- Template - *string*: the template for this field. This can be used to alter the default control for a field. If not provided, a template will be automatically assigned, matching the data-type of the property. Usually omitted.
- Parameters - *object*: this is a key-value object (*string* to *string*) with parameters that are passed to the Field template. Supported parameters depend on the (specified or auto-assigned) template.
- Show[1,2] - *boolean*: when a field needs to be visible this can be omitted. When a field needs to be hidden, set to false. Usually omitted.
- Force[1,2] - *boolean*: default value is False. When set to True, the field will be always visible. Usually omitted.

The Template property supports the following values:

- gui.controlfactory.property.textbox - general purpose textbox control
- gui.controlfactory.property.label - read only control
- gui.controlfactory.property.textarea - multi-line textbox control
- gui.controlfactory.property.listconstraint - dropdown - only applicable for properties that have a ListConstraint
- gui.controlfactory.property.datetimepicker - date picker control - only applicable for data-type d:date or d:datetime
- gui.controlfactory.property.categorypicker - category picker control - only applicable for data-type d:category
- gui.controlfactory.property.checkbox - checkbox control
- gui.controlfactory.property.currency - display number as currency - only applicable for data-type d:int and d:long
- gui.controlfactory.property.size - displays file size - only applicable for data-type d:content
- gui.controlfactory.property.mimetype - displays file mimetype - only applicable for data-type d:content
- gui.controlfactory.property.authority - people picker control - only applicable for data-type d:text
  - Parameter key-type: name|reference
  - Parameter authority: user|group|both

The textbox template will give a textbox as inputfield. The listconstraint template will display a combo box as input field. The typeahead template will give a text box with type ahead function enabled. The textarea template will give a large text area.

(1): Properties were moved from FieldVisibility to Field. FieldVisibility is deprecated, but still supported for backwards compatability.
(2): In the future this will be moved to the Parameters object.

5.3.4. FORM > FIELDVISIBILITY

Note: FieldVisibility is deprecated. Functionality and properties moved to the respective fields, but this keyword is still supported for backwards compatability. If both are present in the configuration, the properties defined on the Field directly take presedence.

In the FieldVisibility array is defined which fields should be visible in which state or whether they should be visible or hidden.

- Id: The id of the field. Example: cm:title.
- Mode: There are tree modes in Alfred Desktop, View, Edit and Create. This specifies in which modes the field needs to be visible. Example Mode: ["View", "Edit"] will only be visible when the form is viewed, and edited. But not when it is used for creation. When a field is not present in the fieldvisibility array, but it is in the sets. All modes are used by default.
- Show: when a field needs to be visible this can be omitted. When a field needs to be hidden, set to false.
- Force: default value is false. When set to true, the field will be mandatory.

5.4. CONFIGURING FORMS

5.4.1. CONFIGURATION FILES

In Alfred Desktop the following form configurations are available by default.

Filename	Scope
metadata-forms-config.json	View/Edit Metadata
columns-forms-config.json	Columns in the detail pane
working-copy-columns-forms-config	Columns in the detail pane
column-sets-forms-config.json	Columns in the detail pane
search-columns-default-forms-config.json	Search / Quick Search
workflow-activiti-forms-config.json	Workflow
workflow-jbpm-forms-config.json	Workflow
workflow-task-forms-config.json	Workflow
facet-forms-config.json	Faceted search

These files are written to work with a default alfresco installation. To create your own configuration file, you can choose any file name as long as it ends with forms-config.json. If you want to override a forms configuration of the default file, you have to use the same id in the main properties (see 4.3.1). After you have created your custom configuration file, you have to insert it in the Company Home/Data Dictionary/Fred/Forms folder. Alfred Desktop will automatically download this file on startup and override al the default configuration that are specified in your file.

5.4.2. EXAMPLES

The syntax of the form configuration will be explained by two examples.

FORMS CONFIGURATION OF A TYPE

Let’s create a form for a custom type xenit:mytype. We create a file named xenitCustomType-forms-config.json. Note that any file name will do, as long as it ends with forms-config.json and if it is placed in the Company Home/Data Dictionary/Fred/Forms folder in your alfresco.

```
{
"Id": "xenit:mytype",
"Evaluator": "node-type",
    "Forms": [
        {
            "FieldVisibility": [
                {
                    "Id": "cm:name"
                },
                {
                    "Id": "cm:categories",
                    "Mode": ["View", "Edit"]
                }
            ],
            "sets": [
            {
                    "Id" : "title",
                     "Fields" : [
                     {
                            "Id" : "cm:name"
                     }]
                },
                {
                    "Id" : "auditable"
                },
                {
                    "Id" : "auto"
                }
            ]
        }
    ]
}
```

The id of the forms configuration is the same as the name of the type.
The evaluator attribute tells Alfred Desktop under which condition the configuration must be used. The following values are supported:

- string-compare: the condition is matched when the id attribute equals the given one.
- node-type: the condition is matched when the type of the selected node equals the id attribute.
- aspect: the condition is met when the selected node has an aspect specified using the id attribute.

In the field visibility we define which fields need to be displayed if the evaluator condition is met. In our example the field cm:name will be displayed in all three modes (View, Edit and Create) and the field cm:categories will only be displayed in View and Edit mode.

In the Sets attribute it can be seen that the cm:name field will be displayed in the title set. Next to the title set there are two empty sets visible. The most important one is the auto set. Fields that are not explicitly assigned to a set will all be displayed in the auto set. The auditable set that is visible and doesn't contain any fields is there to insure the order in which the fields are displayed. If there is another type that does have fields in the auditable set then those fields will be placed between the cm:name field and the cm:categories field.

By Type vs by Aspect

A good practice is to configure fields using an aspect evaluator. (when applicable) This allows you to set default configuration for aspect properties while still enabling specific configuration for some document types that require a different rendering of a property.

For instance, specify a configuration for cm:titled, but override this for cm:folder to hide the field.

```
{
    "Id": "cm:titled",
    "Evaluator": "aspect",
    "Forms": [
    {
        "FieldVisibility": [
        {
            "Id": "cm:title",
            "Mode": ["View","Edit"]
        }]
    }]
}, {
    "Id": "cm:folder",
    "Evaluator": "node-type",
    "Forms": [
    {
        "FieldVisibility": [
        {
            "Id": "cm:title",
            "Mode": ["View"]
        }]
    }]
}
```

Configuration of the aspect-picker

By default, the aspect picker (edit mode) will display all aspects encountered so far. Meaning not all aspects existing in the repository will appear and maybe some aspects are included that are not meant to be user selectable.

You can specify a static list of aspects to that extend on a node-type or aspect basis, just like you configure the visibility of node properties. Let's limit the aspect picker to cm:titled for all nodes with aspect cm:versionable.

```
{
    "Id": "cm:versionable",
    "Evaluator": "aspect",
    "Forms": [
        {
            "FieldVisibility": [
                {
                    "Id": "cm:title"
                }
            ],
            "sets": [
            {
                    "Id" : "aspect-picker-set",
                    "Fields" : [
                    {
                        "Id" : "aspect-picker",
                        "Mode": ["Edit", "Create"],
                        "Force": true,
                        "Parameters" : {
                            "aspect.1" : "cm:titled"
                        }
                    }]
                }]
        }]
}
```

Configuration of the Controls

Label colors

If you want certain controls to be more striking you could change the label colors or add a fieldset with a legend.

To change the label color of a control add the parameter "label-color" to a field control. The following code snippet also adds a fieldset with legend "Info" and wraps around the cm:name and type property.

```
[{
    "Id" : "default-node",
    "Evaluator" : "string-compare",
    "Forms" : [
    {
        "sets" : [
        {
            "Id" : "",
            "Template" : "gui.controlfactory.fieldset.group",
            "Parameters" :
            {
                   "centered" : "true",
                   "collapsible" : "false",
                   "start-collapsed" : "false"
            },
            "Label" : "Info",
            "Fields" :
            [
                {
                    "Id" : "cm:name",
                    "Control" : {
                        "Parameters" : {
                            "label-color" : "DarkOrange"
                        }
                    }
                },
                {
                    "Id" : "type",
                    "Control" : {
                        "Parameters" : {
                            "label-color" : "DarkOrange"
                        }
                    }
                }
            ]
        }]
    }]
}]
```

Date Picker

Default when a date property is empty it will stay empty, although this can be configured.
When a date property should always be the date of today, we can add the parameter "override-default-value" and set the value to "today".
Static dates are also supported (format: DD/MM/YYYY).

```
[{
    "Id" : "default-node",
    "Evaluator" : "string-compare",
    "Forms" : [
    {
        "sets" : [
        {
            "Id" : "example-set",
            "Fields" : [
            {
                "Id": "fred:exampleDate",
                "Parameters" :
                {
                     "override-default-value" : "today"
                }
            }]
        }
    }]
}]
```

People Picker

Default the people picker will search for users but this can be configured. It is configurable by adding the parameter "authority" to the "gui.controlfactory.property.authority" template.
The value of this parameter can be set to "user", "group" or "both" to search for users, groups or both respectively.

```
[{
    "Id" : "people-picker-example",
    "Evaluator" : "aspect",
    "Forms" : [
    {
        "sets" : [
        {
            "Id" : "example-set",
            "Fields" : [
            {
                "Id": "fred:examplePeople",
                "Template" : "gui.controlfactory.property.authority",
                "Parameters" :
                {
                    "authority" : "group"
                }
            }]
        }]
    }]
}]
```

Size & mimetype

Show content properties like size and mimetype in the metadata-panel.

```
[{
    "Id" : "default-node",
    "Evaluator" : "string-compare",
    "Forms" : [
    {
        "sets" : [
        {
            "Id" : "example-content-props-set",
            "Fields" : [
            {
                {
                    "Id": "sizeProperty",
                    "label": "Size Prop",
                    "Template" : "gui.controlfactory.property.size",
                    "Parameters": {
                        "type": "property",
                        "qname": "cm:content"
                    }
                },
                {
                    "Id": "mimeTypeProperty",
                    "label": "Mimetype Prop",
                    "Template" : "gui.controlfactory.property.mimetype",
                    "Parameters": {
                        "type": "property",
                        "qname": "cm:content"
                    }
                }
            }]
        }]
    }]
}]
```

Configuration of the document-type picker

It is possible to specify which document types should be visible in the type picker.

```
{
       "Id": "type",
       "Control": {
            "Parameters" : {
                 "doctype.1" : "cm:content",
                 "doctype.2" : "fred:document",
                 "doctype.2.1": "fred:sales",
                 "doctype.2.1.1": "fred:rfi",
                 "doctype.2.1.2": "fred:rfp",
                 "doctype.2.2": "fred:product",
                 "doctype.2.3": "fred:marketing",
                 "doctype.2.3.1": "fred:events",
                 "doctype.3" : "cm:folder"
            }
      }
}
```

5.4.3. FORM CONFIGURATION OF A WORKFLOW

The following example will illustrate how to override a default forms configuration in Alfred Desktop. In this example we will override the activiti$activitiAdhoc workflow type. To start we will create a file named customWorkflow-forms-config.json.

For this example we only want to display the title, the description, due date, start date and the priority. The forms configuration will look as follows:

```
[{
    "Id" : "activiti$activitiAdhoc",
    "Evaluator" : "string-compare",
    "Forms" : [{
        "FieldVisibility" :
        [{
            "Id" : "title",
            "Mode" : ["View", "Edit"]
        }, {
            "Id" : "bpm:workflowDescription",
            "Mode" : ["Create"]
        }, {
            "Id" : "bpm:workflowStartDate",
            "Mode" : ["View", "Edit"]
        }
    ],
    "sets" : [{
        "Id" : "",
        "Appearance" : "title",
        "Label" : "Info",
        "Label-id" : "workflow.set.general",
        "Fields" : [{
            "Id" : "title"
        },{
            "Id" : "bpm:workflowDescription",
            "Control" : {
                "Template" : "gui.controlfactory.property.textarea"
            }
        }
    ]
    }, {
        "Id" : "info",
        "Appearance" : "",
        "Template" : "gui.controlfactory.fieldset.3-column",
        "Fields" : [{
            "Id" : "bpm:workflowDueDate"
        }, {
            "Id" : "bpm:workflowStartDate"
        }, {
            "Id" : "bpm:workflowPriority"
        }]
    }]
}]
```

To override the default Alfred Desktop forms configuration of the adhoc workflow, we use the same id in the general id attribute field. For an explanation of the evaluator attribute, see 4.4.2.1.

In the FieldVisibility array we only specify three of the five fields. Alfred Desktop will automatically display fields occurring only in the set array in all modes. In our example this is the case for bpm:workflowDueDate and bpm:workflowPriority. These fields will be displayed in View, Edit and Create mode.

If for some reason a field is never allowed to be shown, add "Show" : false with the field id. This will prevent the field from being shown. If a field is mandatory, add "Force" : true with the field id.

In the Sets array, we have specified two sets. This is done to create some special layout. In the first set, the title and workflow description are displayed below each other. The second set containing the remaining three fields. These fields will be put in a three-column table specified by the template attribute.

Appearance: “title” will give the label of the set the makeup style of a title. It is also possible to leave this empty.

The template that is defined with field bpm:workflowDescription in the general set means that the input field is a text area.

5.4.4. FACETED SEARCH CONFIGURATION

The following example will show how to override the default facet configuration in Alfred Desktop. Overriding the default facet configuration will allow the administrator to specify which facets need to be returned by the solr indexing system.

In the following example we will create a config that will return facets for a fictional property projectleader. The .__.u postfix means that the property is untokenized. This means that the value of the property is considered as a whole and is not split up in smaller tokens. For more information about tokenized and untokenized, see the Alfresco documentation.

```
[{
"Id" : "search",
    "Evaluator" : "string-compare",
    "Forms" : [{
    "sets" : [{
        "Id" : "default-facets",
            "Fields" : [{
                "Id" : "title"
                },{
                    "Id" : "@{http://www.alfresco.org/model/content/1.0}creator.__.u",
                },{
                    "Id" : "@{http://www.alfresco.org/model/content/1.0}modifier.__.u",
                },{
                    "Id" : "@{http://www.alfresco.org/model/content/1.0}taggable",
                },{
                    "Id" : "@{http://www.alfresco.org/model/content/1.0}categories",
                },{
                    "Id" :
"@{http://www.fred.org/model/content/2.3}projectleader.__.u",
                }
            ]
    }]
}]
```

5.4.5. FORMS COLUMN CONFIGURATION

Users can select which columns they want to view in Alfred Desktop's folder view.

Alfred Desktop comes with 2 sets of configuration that define how users can select different columns:

- column lists allow the user to add or remove a single column
- column sets allow the user to use a (forms-config) predefined set of columns or his own saved set of columns

FIG. 5. FORMS COLUMN CONFIGURATION

Column List Configuration

The syntax to define the list of columns a user can select:

```
{
    "Id": "default-column-list",
    "Evaluator": "string-compare",
    "Forms": [
        {
            "sets": [
                {
                    "Id": "System",
                    "Label": "System Properties",
                    "Appearance": "section",
                    "Fields": [
                        {
                            "Id": "Format"
                        },
                        {
                            "Id": "Location"
                        },
                        {
                            "Id": "Info"
                        }
                    ],
                    "subsets": [
                        {
                            "Id": "WorkingCopy",
                            "Appearance": "submenu",
                            "Fields": [
                                {
                                    "Id": "cm:workingCopyOwner"
                                }]
                        }]
                }]
        }]
}
```

Note that *Id* can either by "default-column-list" of "column-list".

The former will replace the whole list of built-in column-properties under "System Properties".
The latter will prepend the forms-config-defined list to the "System Properties"

Using the *Appearance* option, a subsection can be rendered as either a section or a submenu.

Column Set Configuration

It can be usefull to group properties together in such a way that the user can instantly select the group or set of properties.

FIG. 6. FORMS COLUMN SET CONFIGURATION

This can be defined via the column set configuration:

```
{
    "Id": "default-column-sets",
    "Evaluator": "string-compare",
    "Forms": [
        {
            "sets": [
                {
                    "Id": "AuditSet",
                    "Label": "Who / When",
                    "Fields": [
                            {
                                "Id": "cm:author"
                            },
                            {
                                "Id": "cm:modified"
                            },
                    ]
                },
                {
                    "Id": "ByApsectsSets",
                    "Label": "By Aspects",
                    "subsets": [
                        {
                            "Id": "AuditableAspect",
                            "Label": "Auditable",
                            "Fields": [
                                {
                                    "Id": "cm:creator"
                                },

                                {
                                    "Id": "cm:created"
                                },
                                {
                                    "Id": "cm:modifier"
                                },
                                {
                                    "Id": "cm:modified"
                                },
                            ]
                        }]
                }]
        }]
}
```

In this configuration, individual fields represent the columns in a set (and are not rendered in the menu). The sections are the column-set menu items.

6. CREATING DOCUMENTS FROM TEMPLATES

To create documents from document templates in the Alfresco repository, you have to add the templates to the folder *'Node Templates'* in *Data Dictionary*.

FIG. 7. DOCUMENT TEMPLATES IN DATA DICTIONARY

All the documents that are inserted in this folder will be shown under "New Document" in the context menu.

FIG. 8. CREATING A NEW DOCUMENT FROM A TEMPLATE

Within the "Node Templates" folder you can also create subfolders and put your document templates in it to group them together.

You can configure which templates will be available for which user or group by setting the reading rights on the templates in the Document Template folder. So you can attribute a template to the finance department.

7. LOADING RUNTIME CLIENT CONFIG FROM ALFRESCO

Alfred Desktop uses the Spring.NET framework for dependency injection.

Starting with Fred 3.1, it's possible to reconfigure parts of Alfred Desktop with a Spring.NET configuration loaded from the repository.

This can be used to customize, for instance, the context menu to your users' satisfaction. The configuration is server side and can be Alfred Desktop version dependable. It will be applied to every user connecting to that repository.

The configuration file must be placed in /Company Home/Data Dictionary/Fred/Config/[Fred version]/*-custom-context.xml.

7.1. VERSION DEPENDANT CONFIGURATION

Different configurations can be used for different Alfred Desktop/Fred versions. The configuration of the most specified matching version will be used.

Example:
When 3 folders, "3", "3.1" and "3.1.127" exist in /Company Home/Data Dictionary/Fred/Config/
and connect with Fred 3.1.127. Then the configuration file inside folder "3.1.127" will be used. If a different Fred 3.1 is connecting then the "3.1" configuration will be used. The configuration file inside folder "3" will be used if the Alfred Desktop/Fred version is in the version 3 range but differs from 3.1.

Note: Leading zero's have to be ignored when creating version folders.

FIG. 9. DATA DICTIONARY CONFIGURATION FOLDER

7.2. CONTEXTMENU

There are multiple uses of context menus in Alfred Desktop. All these menus are defined in spring configuration, but not all menus can be customized on the server side spring configuration. For instance, the menubar at the top of the screen is already visible before connecting to a repository. Customizing this menu in the server-side spring configuration won't have any effect. Never the less, the right click context menu is fully customizable.

In Fred 3.1 it is necessary to fully override the view.mediator.[TYPE] object id, this is not advised to do so in any other version of Alfred Desktop/Fred. Therefore it is recommended that a separate 3.1 spring configuration folder is created for these kinds of customizations. In future versions of Alfred Desktop it will be possible to append commands to the already existing context menu, or remove them by extending the command builder objects. This will be further clarified in later version documentations.

```
<objects xmlns="http://www.springframework.net">
<object id="view.mediator.folder"
```

```
type="Xenit.Fred.Session.Controller.Browser.ItemMediators.FolderItemMediator"
            parent="view.mediator.abstract-reponode"
            singleton="false">
        <property name="IconProvider" ref="view.item-icon-providers.folder" />
        <property name="Commands">
            <list element-type="Xenit.Fred.View.CommandBuilder.ICommandBuilder,
Xenit.Fred.View" merge="true">
                <ref object="command.builder.open-folder" />
                <ref object="command.builder.refresh" />
                <ref object="command.builder.copy" />
                <ref object="command.builder.cut" />
                <ref object="command.builder.new-folder" />
                <ref object="command.builder.new-document" />
                <ref object="command.builder.paste" />
                <ref object="command.builder.paste-internal" />
                <ref object="command.builder.show-in-browser" />
                <ref object="command.builder.zoom-view" />
                <ref object="command.builder.edit-metadata" />
                <ref object="command.builder.delete" />
                <ref object="command.builder.rename-reponode" />
                <ref object="command.builder.favorites-add" />
                <ref object="command.builder.acl" />
            </list>
        </property>
    </object>
</objects>
```

These changes made to the commands list will apply on the context menu of folders.

7.2.1. DEFINING COMMANDS

Commands are defined with command builders.

Reference list of command builders:

- command.builder.about
- command.builder.abstract
- command.builder.abstract-controller-sr
- command.builder.acl
- command.builder.assignee-remove
- command.builder.bookmark-connect-offline
- command.builder.bookmark-delete
- command.builder.bookmark-edit
- command.builder.bookmark-open
- command.builder.bookmark-open-wo-persitentcache
- command.builder.browser

- command.builder.cache-dictionary-memory-clear
- command.builder.cache-dictionary-mongo-clear
- command.builder.cache-node-memory-clear
- command.builder.cache-node-mongo-clear
- command.builder.cache.content-clear
- command.builder.cancel-checkout
- command.builder.checkin
- command.builder.checkout
- command.builder.clipboard-spy
- command.builder.clipboard.print-storage
- command.builder.clipboard.print-storage-lindex-0
- command.builder.clipboard.print-storage-lindex-minus-1
- command.builder.close-session
- command.builder.columns.forms-abstract
- command.builder.commit-offline
- command.builder.commitstatus
- command.builder.compare-document
- command.builder.convert-to-pdf
- command.builder.copy
- command.builder.copy-fred-link
- command.builder.copy-link
- command.builder.copy-property-displayvalue
- command.builder.copy-property-qname
- command.builder.copy-property-stringvalue
- command.builder.copy-property-title
- command.builder.copy-ref
- command.builder.cut
- command.builder.daeja-view
- command.builder.delete
- command.builder.delete-column-set
- command.builder.developer.disable-incomplete-tracker
- command.builder.developer.mimetypes
- command.builder.developer.preview-mapping
- command.builder.developer.reload-forms-config
- command.builder.developer.repo-info
- command.builder.developer.run-incomplete-tracker
- command.builder.developer.ticket-client-delete
- command.builder.developer.ticket-copy
- command.builder.developer.ticket-server-delete
- command.builder.developer.ticket-validate
- command.builder.digitalsignature
- command.builder.digitalsignature-external
- command.builder.discard-offline-sync
- command.builder.disconnect
- command.builder.doctemplates
- command.builder.document-template
- command.builder.edit-metadata

- command.builder.editonline
- command.builder.enable-versioning
- command.builder.export-zip
- command.builder.export.exportmetadata
- command.builder.favorites-add
- command.builder.favorites-remove
- command.builder.favorites-rename
- command.builder.group-by-menu
- command.builder.help
- command.builder.impersonated-permission-abstract
- command.builder.impersonated-permission-create
- command.builder.impersonated-permission-delete
- command.builder.impersonated-permission-read
- command.builder.impersonated-permission-write
- command.builder.impersonating-browser
- command.builder.istorage-explorer
- command.builder.launch-explorer
- command.builder.load-column-set
- command.builder.mail-as-attachment
- command.builder.mail-as-link
- command.builder.maillogfiles
- command.builder.missingmetadata
- command.builder.new-document
- command.builder.new-folder
- command.builder.node-drop-handler
- command.builder.node-types
- command.builder.open-appdata-fred-location
- command.builder.open-document
- command.builder.open-folder
- command.builder.open-peer-assocs
- command.builder.open-settings
- command.builder.paste
- command.builder.paste-internal
- command.builder.paste-internal-link
- command.builder.permissions-remove
- command.builder.permissions.add
- command.builder.permissions.change-role
- command.builder.permissions.toggle-inheritance
- command.builder.preview
- command.builder.print.printdocument
- command.builder.properties
- command.builder.quit
- command.builder.refresh
- command.builder.refresh-document
- command.builder.refresh-offline
- command.builder.remove-aspect
- command.builder.rename-reponode

- command.builder.rename-reponode-execute
- command.builder.repository-browser
- command.builder.restore-deleted
- command.builder.revert-version
- command.builder.save-column-set
- command.builder.savedsearches-add
- command.builder.savedsearches-remove
- command.builder.savedsearches-rename
- command.builder.settype
- command.builder.show-columns-menu
- command.builder.show-forms-column-sets-menu
- command.builder.show-forms-columns-menu
- command.builder.show-in-browser
- command.builder.show-link-target
- command.builder.show-node
- command.builder.show-node-childs
- command.builder.start-workflow
- command.builder.sync-offline
- command.builder.tab-hosted-controller
- command.builder.tab-hosted-controller-by-id
- command.builder.toggle-group-column
- command.builder.toggle-show-column
- command.builder.toggle-sync
- command.builder.trash
- command.builder.version-history
- command.builder.view-metadata
- command.builder.webbrowser-abstract
- command.builder.workflow-overview
- command.builder.workflow-started
- command.builder.workflow.instance.copy-fred-link
- command.builder.workflow.show-node
- command.builder.workflow.task.copy-fred-link
- command.builder.workflow.task.show-in-browser
- command.builder.zoom-view

Note: adding command builders which are not relevant for the node in question won't be visible in the context menu, this is explained in title 6.2.3.1 Evaluators.

7.2.2. ITEM MEDIATORS

Reference list of configurable item mediator types:

- view.mediator.abstract
- view.mediator.abstract-reponode
- view.mediator.acl
- view.mediator.aspect
- view.mediator.assignee-group
- view.mediator.assignee-person

- view.mediator.authority-add
- view.mediator.bookmark
- view.mediator.bookmark-add
- view.mediator.bookmark-edit
- view.mediator.category
- view.mediator.classification
- view.mediator.doctemplate
- view.mediator.doctype
- view.mediator.document
- view.mediator.documenutcommitstatus
- view.mediator.dummy
- view.mediator.exception
- view.mediator.extended-assoc
- view.mediator.extended-node-reference
- view.mediator.favorites-folder
- view.mediator.favorites-item
- view.mediator.favorites-tree-item
- view.mediator.filtered-view-folder
- view.mediator.filtered-view-item
- view.mediator.filtered-view-tree-item
- view.mediator.folder
- view.mediator.group
- view.mediator.group-picker
- view.mediator.info
- view.mediator.link
- view.mediator.loading
- view.mediator.manageAspect
- view.mediator.missing.document
- view.mediator.namevalue
- view.mediator.newfolder
- view.mediator.offline-folder
- view.mediator.offline-folder-container
- view.mediator.package-item
- view.mediator.package-item-folder
- view.mediator.peer-assocs-folder
- view.mediator.person
- view.mediator.person-picker
- view.mediator.property
- view.mediator.property-value
- view.mediator.propertywrapper
- view.mediator.recyclebin-document
- view.mediator.recyclebin-folder
- view.mediator.savedsearches-folder
- view.mediator.savedsearches-item
- view.mediator.search.document
- view.mediator.search.folder
- view.mediator.searchfacet

- view.mediator.searchfacetvalue
- view.mediator.searchfacetvaluemimetype
- view.mediator.searchnodetypemediator
- view.mediator.tree.acl
- view.mediator.tree.category
- view.mediator.tree.category-as-folder
- view.mediator.tree.category-container
- view.mediator.tree.category-picker
- view.mediator.tree.folder-browser
- view.mediator.trial
- view.mediator.unknown-node
- view.mediator.versionhistory
- view.mediator.virtual-folder
- view.mediator.workflow-instance
- view.mediator.workflow-task
- view.mediator.working-copy
- view.mediator.workingcopies-folder
- view.mediator.workingcopies-item
- view.mediator.List-contraint-property-value

“Search document” and “search folder” are separated from the generic “document” mediator and the “folder” mediator. The reason for this is that the context menu of the search results differ from the context menu in the contents view workspace. This implementation might change in future versions of Alfred Desktop.

7.2.3. CREATING CUSTOM COMMANDS

Custom commands can be used to extend the context menu with customized functionalities.

The following properties can be set for commands:

- Title : display text in context menu (mandatory)
- Name : the name of the command
- Group : the context menu is divided in categorized groups. (default: "main")

These customized functionalities are for example: opening a web browser tab.

When a command inherits from the webbrowser-abstract, the web browser tab will be opened when this command is used. The command can be further configured to specify the URL for the web browser tab. The property External can be set to true to open the URL in the default web browser.

```
<object id="command.builder.admin"
        parent="command.builder.webbrowser-abstract">
        <property name="Title" value="server details" />
        <property name="Name" value="server details" />
        <property name="Group" value="admin" />
        <property name="UrlPattern" value="{url:schema}://{url:host}:
{url:port}/{url:context}
              /service/api/server?a=false&amp;alf_ticket={auth:ticket}" />
        <property name="External" value="false" />
    </object>
```

7.2.4. EVALUATORS

Evaluators can be used to specify whether the command is visible or disabled for the node in question.

Evaluators are configured as list-properties with name "Executable" or "Relevant" in the command-builder. When an executable evaluator returns false, then the item will be grayed out in the context menu. When a relevant evaluator returns false, then the item won't be visible in the context menu.

Note: The evaluators defined in the spring configuration are applied after the built-in evaluators on command itself.

Example usage: the property 'Executable' defined on a command-builder will grey out the command in the context-menu, when (one of) the evaluators returns false. In this example, the context-menu-item will be greyed out of the node is-not or does-not-inherit-from the Alfresco type *cm:content*.

```
<property name="Executable">
   <list element-type="ICommandEvaluator">
      <object parent="command.eval.node-type-required">
         <property name="Type" value="cm:content"/>
    </object>
   </list>
</property>
```

Summary of evaluator types:

- command.eval.can-create-children
- command.eval.can-write
- command.eval.config
- command.eval.content-required
- command.eval.dev-only
- command.eval.disabled
- command.eval.min-repo-version
- command.eval.node-is-working-copy
- command.eval.node-type-required
- command.eval.node-writeable
- command.eval.not
- command.eval.not-available-offline
- command.eval.or

7.2.5. DOCUMENT TEMPLATE PICKER

When you create a document from a template by default you see a list of documents in the context menu.

FIG. 10. DEFAULT DOCUMENT TEMPLATE PICKER

This presentation is handy when you don't have a lot of possible document templates. When you do have a lot of document template you might prefer a template picker with a search box.

FIG. 11. DOCUMENT TEMPLATE PICKER WITH SEARCH BOX

If you want to use the document template picker with a search box you will need a custom spring configuration.
The following piece of code will enable the document template picker with earch box.

```
<objects xmlns="http://www.springframework.net">

    <object id="command.builder.new-document"
          parent="command.builder.abstract"

type="Xenit.Fred.Session.Controller.DocumentTemplates.Command.NewDocumentCommandBuilder
, Xenit.Fred.Session.Controller">
      <property name="Title" value="New Document" />

      <property name="AppMessageBus" ref="service.app-message-bus" />
      <property name="PermissionService" ref="repository.service.permissions" />
      <property name="TemplateService" ref="repository.service.template" />
      <property name="NodeService" ref="repository.service.node" />
      <property name="DictionaryService" ref="repository.service.dictionary" />
      <property name="IconService" ref="view.item-icon-service" />
      <property name="RecordsManagementService" ref="repository.service.records-
management" />

      <property name="TemplateCommandBuilder" ref="command.builder.document-template"
/>

      <property name="TemplatesPickerControllerBuilder">
            <object type="Xenit.Fred.View.ControlMediator.Builder.ControllerBuilder,
Xenit.Fred.View">
                <property name="Id" value="control.document-template.filter-picker"/>
            </object>
      </property>

    </object>

</objects>
```

It is also possible to further configure the document template picker to classify the templates in categories.

FIG. 12. DOCUMENT TEMPLATE PICKER WITH TEMPLATES CLASSIFIED IN CATEGORIES

The following configuration enables a document template picker with templates classified in categories.

```
<objects xmlns="http://www.springframework.net">

    <object id="command.builder.new-document"
          parent="command.builder.abstract"

type="Xenit.Fred.Session.Controller.DocumentTemplates.Command.NewDocumentCommandBuilder
, Xenit.Fred.Session.Controller">
      <property name="Title" value="New Document" />

      <property name="AppMessageBus" ref="service.app-message-bus" />
      <property name="PermissionService" ref="repository.service.permissions" />
      <property name="TemplateService" ref="repository.service.template" />
      <property name="NodeService" ref="repository.service.node" />
      <property name="DictionaryService" ref="repository.service.dictionary" />
      <property name="IconService" ref="view.item-icon-service" />
      <property name="RecordsManagementService" ref="repository.service.records-
management" />

      <property name="TemplateCommandBuilder" ref="command.builder.document-template"
/>

      <property name="TemplatesPickerControllerBuilder">
            <object type="Xenit.Fred.View.ControlMediator.Builder.ControllerBuilder,
Xenit.Fred.View">
                <property name="Id" value="control.document-template.filter-picker"/>
            </object>
      </property>

    </object>

    <object id="control.document-template.filter-picker"
        parent="control.mediator.filter-picker"
        singleton="false">

      <property name="PickerController" ref="control.mediator.categorytemplates" />
    </object>

</objects>
```

7.2.6. CUSTOM DIALOG CONTROL

A feature introduced in Alfred Desktop 3.6 is a custom dialog control. It allows users to fill in a form in a dialog and send the inputs of that form to webscript in Alfresco.
The dialog can be opened by choosing the command from the context menu, if it is configured correctly.

FIG. 13. EXAMPLE DIALOG CONTROL WITH 2 INPUT FIELDS

Forms Configuration

A forms configuration is required to design the form you want to see in the dialog.

Example Forms Configuration:

```
[
    {
        "Id" : "example:sample-dialog",
        "Evaluator" : "string-compare",
        "Forms" : [
            {
                "sets" : [
                    {
                        "Id" : "content",
                        "Fields": [
                            {
                                "Id": "input1",
                                "Label": "Input Text Field 1",
                                "Parameters" : {
                                    "template" : "gui.controlfactory.property.textbox"
                                }
                            },
                            {
                                "Id": "input2",
                                "Label": "Input Text Field 2",
                                "Parameters" : {
                                    "template" : "gui.controlfactory.property.textbox"
                                }
                            }
                        ],
                        "Parameters" : {
                            "endPoint" :
"/s/automation/swm/{space}/{store}/{guid}/createFolderStructure",
                            "requestMethod" : "POST",
                            "invalidate" : "assocs"
                        }
                    }
                ]
            }
        ]
    }
]
```

For each field:
You have to set the "template" in the "Parameters". As of Alfred Desktop 3.6 only text fields are supported so the template "gui.controlfactory.property.textbox" is the only valid one.
In the future other templates (for example date picker fields) will also be supported.

For earch set:
In "Parameters" you have to define the endpoint to which you would like to send your inputs. You do this by setting the "endPoint".
The inputs are sent as a json array in the same sequence as they are defined in the forms configuration. This is important to notice because you will have take this into account in the webscript that is being called. For the example forms-configuration this would mean that the json array being sent to the webscript contains 2 entries, the first being input1 and the second being input2.
The value of "requestMethod" can be set to "GET" or "POST". This signifies the http request method you want to use.
"invalidate" is used to set the cache invalidation. The value of "invalidate" can be set to "none", "node", "assocs" or "both". "none" is the default value and will not do anything. "node" is used to invalidate the node metadata in the Alfred Desktop cache. This is to make sure the metadata of that particular node are up-to-date. "assocs" is used to invalidate the child associations of the node in the Alfred Desktop cache. This is to make sure that the child associations of the particular node are up-to-date. "both" is a combination of "node" and "assocs".

The file in which you made this configuration can have whichever name you want as long as it ends with "forms-config.json".

Spring Configuration

In order to see the dialog you have to configure a command that opens it up. The configuration happens by overwriting the spring configuration
of Alfred Desktop.

Example spring configuration:

```
<objects xmlns="http://www.springframework.net">
    <object id="command.builder.open-sample-dialog-box"
        parent="command.builder.open-dialog-box">
        <property name="Title" value="This a sample dialog box" />
        <property name="ID" value="example:sample-dialog" />
    </object>
</objects>
```

Here the "ID" value has to be set to the id you set in the forms configuration (see the id in the previous example).

7.3. DEFAULT TABS

Beside the browse and search tab, it is possible to configure other default tabs. These defaults tabs will open automatically when connection to the repository is made.
For example, the workflow overview tab can be added which will from then on open automatically and will no longer be closable.

Example default tabs

```
<object id="bootstrap.controllers"
        type="Xenit.Fred.Session.Bootstrap.BootstrapControllers, Xenit.Fred.Session"
        singleton="true">
    <property name="ControllerHost" expression="@(application-
controller).PrimaryWindowController.MainController.TabController" />
    <property name="ControllerList">
        <list element-type="IController">
            <ref object="controller.session.browser.main" />
            <ref object="controller.session.search.main" />
      <ref object="controller.workflow-overview" />
        </list>
    </property>
</object>
```

7.4. WORKSPACES

7.4.1. DEFAULT COLUMNS

Each workspace has its own set of default columns. These sets of default columns can be configured for each separate workspace.

At the moment it is necessary to override the existing defined workspace to change the default columns property. This is subject to change in one of the next versions of Alfred Desktop, so be advised to only apply this configuration in a “3.1” version specified configuration folder.

The following examples also show how to define your own columns with properties from a custom document model and use them as a default column.

Example default columns in contents view workspace

```
<object id="workspace.browser.contents.default"

type="Xenit.Fred.Session.Controller.Browser.Workspace.FolderContentsListViewWorkspace"
        singleton="false">
```

```
    <property name="NodeService" ref="repository.service.node" />
    <property name="DictionaryService" ref="repository.service.dictionary" />
    <property name="ConfigService" ref="repository.service.config" />
    <property name="NodeCacheService" ref="repository.service.nodecache" />
    <property name="PathProvider" ref="path-provider.reponode-or-default" />
    <property name="RefreshCommandBuilder" ref="command.builder.refresh" />
    <property name="ColumnService" ref="view.service.column-state" />

    <property name="FormsConfigService"     ref="repository.service.formconfig" />
    <property name="TranslationService"     ref="service.translation" />
    <property name="ControlFactory"         ref="gui.controlfactory" />
    <property name="ViewArtifactFactory"    ref="view.service.item-factory" />

    <property name="ColumnService" ref="view.service.column-state" />
    <property name="ShowColumnsMenuBuilder" ref="command.builder.show-forms-columns-
menu" />
    <property name="ShowColumnSetsMenuBuilder" ref="command.builder.show-forms-column-
sets-menu" />

    <property name="DefaultColumns">
        <list element-type="Xenit.Fred.View.ColumnBuilder.IColumnBuilder,
Xenit.Fred.View">
            <ref object="view.column.node.property.name" />
            <ref object="view.column.node.type" />
        </list>
    </property>
</object>

<object id="view.column.node.property.custom-property"
        parent="view.column.node.property"
        singleton="true">
      <constructor-arg name="title" value="Custom property" />
      <constructor-arg name="propertyQName" value="cm:exampleProperty" /> <!--custom
property-->
      <property name="Renderer" ref="view.renderer.node.property" />
</object>
```

Example default columns in workflow overview workspace

My tasks

```
<object id="controller.workflow-mytasks"
        parent="controller.workflow.instance.list-abstract"
        type="Xenit.Fred.Session.Workflow.Controller.MyTasksController,
Xenit.Fred.Session.Workflow"
        singleton="false">

    <property name="ControlFactory" ref="gui.controlfactory" />
    <property name="PathProvider" ref="path-provider.workflow" />
    <property name="DictionaryService" ref="repository.service.dictionary" />

    <property name="DefaultColumns">
        <list element-type="IColumnBuilder">
            <object parent="view.column.base">
                <property name="Id" value="Task" />
                <property name="Title" value="My Tasks" />
                <property name="Renderer" ref="view.renderer.workflow-task.title" />
            </object>
            <ref object="view.column.workflow.task.description" />
            <ref object="view.column.workflow.task.duedate" />

            <ref object="view.column.workflow.task.initiator" />
            <ref object="view.column.workflow.task.created" />
            <ref object="view.column.workflow.task.priority" />
            <ref object="view.column.workflow.task.example" />

        </list>
    </property>
</object>

<object id="view.column.workflow.task.example"
        parent="view.column.base">
    <property name="Id" value="Test" />
    <property name="Title" value="Test" />
    <property name="Renderer" ref="view.renderer.workflow-task.property" />
    <property name="Parameters">
        <dictionary key-type="string" value-type="string">
            <entry key="qname" value="cm:exampleProperty" />
        </dictionary>
    </property>
</object>
```

Workflows I've started

```
<object id="controller.workflow.instance.list-abstract"
        parent="controller.listview"
        abstract="true">
    <property name="WorkflowService" ref="repository.service.workflow" />
    <property name="ControlFactory" ref="gui.controlfactory" />
    <property name="MediatorMapping">
        <name-values merge="false">
            <add key="LoadingItemMediator" value="view.mediator.loading" />
            <add key="WorkflowInstanceMediator" value="view.mediator.workflow-instance"
/>
        </name-values>
    </property>
    <property name="MappingDecorators">
        <name-values merge="false">
            <add key="LoadingItemMediator" value="view.decorator.loading" />
        </name-values>
    </property>

    <property name="DefaultColumns">
        <list element-type="IColumnBuilder">
            <ref object="view.column.workflow.instance.title" />
            <ref object="view.column.workflow.instance.message" />
            <ref object="view.column.workflow.instance.started" />
            <ref object="view.column.workflow.instance.test" />
        </list>
    </property>
</object>

<object id="view.column.workflow.instance.test"
        parent="view.column.base">
    <property name="Id" value="Test" />
    <property name="Title" value="Test" />
    <property name="Renderer" ref="view.renderer.workflow-instance.property" />
    <property name="Parameters">
        <dictionary key-type="string" value-type="string">
            <entry key="qname" value="cm:exampleProperty" />
        </dictionary>
    </property>
</object>
```

7.5. SEARCH-BASED VIRTUAL FOLDERS/VIEWS

You can add custom sets of *virtual folders* to the tree view. These *virtual folders* are predefined searches where the results will be shown as if it is a folder. They are useful when a specific search is done often, by multiple users.

FIG. 14. VIRTUAL FOLDERS EXAMPLE

To add this feature, a *'filtered-view'* object should be defined in the Spring.NET configuration, with parent id "workspace.browser.foldertree.filtered-view". The property "Name" defines the display name in the left panel (Invoicing in the figure above). The "Folder" property defines from where in the repository Alfred Desktop should load the XML files with search-definitions.

Additionally, two workspaces need to be configured: one for the left folder-tree panel and one for the folder-contents panel in the middle.

To add the new workspaces to the workspace providers we need to override both providers (tree and content). As stated before, the xml definitions can change with new releases of Alfred Desktop, so it would be a good practice to store these config changes in a '3.1' configuration folder in the repository.

Spring.NET Config for custom Virtual Folders

```
<!-- Filter Service -->
<object id="service.my-custom-filter" parent="service.filtered-view-base"
singleton="true">
    <property name="Name" value="New filter" />
    <property name="Folder" value="/Company Home/Data Dictionary/Fred/Filtered Views"
/>
</object>

<!-- Folder Tree Workspaces -->
<object id="workspace.my-custom-filter.foldertree"
    parent="workspace.browser.foldertree.filtered-view"
    singleton="false">
        <property name="FilteredViewService" ref="service.my-custom-filter" />
</object>
<!-- Folder Contents Workspaces -->
<object id="workspace.my-custom-filter.contents"
    parent="workspace.browser.contents.filtered-view"
    singleton="false">
        <property name="FilteredViewService" ref="service.my-custom-filter" />
</object>

<!-- Override Workspace Providers -->
<object id="workspace-provider.browser.foldertree"
     parent="workspace.provider.base" singleton="false">
         <property name="Workspaces">
             <list element-type="Xenit.Fred.View.Logic.Workspace.IViewWorkspace,
                Xenit.Fred.View">
                <ref object="workspace.browser.foldertree.browser" />
                <ref object="workspace.browser.foldertree.workingcopies" />
                <ref object="workspace.browser.foldertree.savedsearches" />
                <ref object="workspace.browser.foldertree.favorites" />
                <ref object="workspace.my-custom-filter.foldertree" />
```

```
            </list>
        </property>
</object>
<object id="workspace-provider.browser.contents"
      parent="workspace.provider.base" singleton="false">
         <property name="Workspaces">
             <list element-type="Xenit.Fred.View.Logic.Workspace.IViewWorkspace,
                 Xenit.Fred.View">
                 <ref object="workspace.browser.contents.favorites" />
                 <ref object="workspace.my-custom-filter.contents" />
                 <ref object="workspace.zoom.contents.category" />
                 <ref object="workspace.browser.contents.default" />
                 <ref object="workspace.browser.contents.peerassocs" />
                 <ref object="workspace.browser.contents.workingcopies" />
            </list>
        </property>
    </object>
```

The configuration for these filters is the same as the saved searches. So the easiest way to create a new filter is to do the search in Alfred Desktop, add the columns and save the search with the "include columns" checkbox checked. Then go to "File" -> "Open Settings Location" and open savedsearches.xml. Search for your recently added saved search and copy the "savedsearch" tag to a new XML file, save this new XML file in the repository in the folder you specified in the Sprint.NET configuration file (example: /Company Home/Data Dictionary/Fred/Filtered Views).

The icon attribute in the "savedsearch" tag can be any icon inside the "[Alfred Desktop install directory]/images/icon16"-folder.

Example Virtual Folders

```
<savedsearch type="path" name="Example Filter" icon="searchplain.png">
    <sortoptions>
        <displaycolumns>
            <column id="Name" name="Name" index="0" />
            <column id="Info" name="Info" index="1" />
            <column id="example-prefix:exampleProperty" name="Example property"
index="4">
                <type>property</type>
                <qname>example-prefix:exampleProperty</qname>
                <renderer>view.renderer.node.property</renderer>
                <width>150</width>
            </column>
            <column id="Location" name="Location" index="5" />
        </displaycolumns>
        <columntosort id="Name" name="Name" order="Ascending" />
    </sortoptions>
    <searchnode>
        <compositenode searchoperator="SearchOperatorAnd">
            <searchtermtype qname="{http://www.xenit.eu/model/example-
model/1.0}exampleType" />
            <searchtermaspect qname="{http://www.xenit.eu/model/example-
model/1.0}exampleAspect" />
            <searchtermall value="*" />
        </compositenode>
    </searchnode>
</savedsearch>
```

7.5.1. DATE RANGES

Date ranges for date properties can be configured using a start date and an end date.

Example of date range with start date and end date

```
<savedsearch type="path" name="test" id="1666bd8e-ca19-47f9-a6f2-de714dfbd530"
icon="searchplain.png">
  <sortoptions>
    <displaycolumns>
      <column id="cm:name" name="Name" index="0">
        <type>property</type>
        <qname-enum>PROPERTY_NAME</qname-enum>
        <qname>cm:name</qname>
        <fixed>True</fixed>
        <width>150</width>
        <renderer>view.renderer.node.property.name</renderer>
      </column>
      <column id="Info" name="Info" index="1">
        <width>150</width>
      </column>
      <column id="Location" name="Location" index="2">
        <width>150</width>
      </column>
    </displaycolumns>
    <columntosort id="cm:name" name="Name" order="Ascending" />
  </sortoptions>
  <searchnode>
    <compositenode searchoperator="SearchOperator:AND">
      <searchtermpath name="Company Home"
nodereference="workspace://SpacesStore/e05799e7-601c-4253-915f-16e5c3bc1825" />
      <compositenode searchoperator="SearchOperator:OR">
        <searchtermtype qname="{http://www.alfresco.org/model/content/1.0}folder" />
        <searchtermtype qname="{http://www.alfresco.org/model/content/1.0}content" />
      </compositenode>
      <searchtermdatetime startdate="2017-09-07T00:00:00+02:00" enddate="2017-09-
08T23:59:59+02:00" qname="{http://www.alfresco.org/model/content/1.0}created" />
    </compositenode>
  </searchnode>
</savedsearch>
```

You can also set a date range without a start date. The resulting start date will be today - 30 days.

Example of date range with only an end date

```
<savedsearch type="path" name="test" id="1666bd8e-ca19-47f9-a6f2-de714dfbd530"
icon="searchplain.png">
  <sortoptions>
    <displaycolumns>
      <column id="cm:name" name="Name" index="0">
        <type>property</type>
        <qname-enum>PROPERTY_NAME</qname-enum>
        <qname>cm:name</qname>
        <fixed>True</fixed>
        <width>150</width>
        <renderer>view.renderer.node.property.name</renderer>
      </column>
      <column id="Info" name="Info" index="1">
        <width>150</width>
      </column>
      <column id="Location" name="Location" index="2">
        <width>150</width>
      </column>
    </displaycolumns>
    <columntosort id="cm:name" name="Name" order="Ascending" />
  </sortoptions>
  <searchnode>
    <compositenode searchoperator="SearchOperator:AND">
      <searchtermpath name="Company Home"
nodereference="workspace://SpacesStore/e05799e7-601c-4253-915f-16e5c3bc1825" />
      <compositenode searchoperator="SearchOperator:OR">
        <searchtermtype qname="{http://www.alfresco.org/model/content/1.0}folder" />
        <searchtermtype qname="{http://www.alfresco.org/model/content/1.0}content" />
      </compositenode>
      <searchtermdatetime enddate="2017-09-08T23:59:59+02:00" qname="
{http://www.alfresco.org/model/content/1.0}created" />
    </compositenode>
  </searchnode>
```

```
</savedsearch>
```

It is also possible to omit the end date. The resulting end date will be today + 1 day

Example of date range with only a start date

```
<savedsearch type="path" name="test" id="1666bd8e-ca19-47f9-a6f2-de714dfbd530"
icon="searchplain.png">
  <sortoptions>
    <displaycolumns>
      <column id="cm:name" name="Name" index="0">
        <type>property</type>
        <qname-enum>PROPERTY_NAME</qname-enum>
        <qname>cm:name</qname>
        <fixed>True</fixed>
        <width>150</width>
        <renderer>view.renderer.node.property.name</renderer>
      </column>
      <column id="Info" name="Info" index="1">
        <width>150</width>
      </column>
      <column id="Location" name="Location" index="2">
        <width>150</width>
      </column>
    </displaycolumns>
    <columntosort id="cm:name" name="Name" order="Ascending" />
  </sortoptions>
  <searchnode>
    <compositenode searchoperator="SearchOperator:AND">
      <searchtermpath name="Company Home"
nodereference="workspace://SpacesStore/e05799e7-601c-4253-915f-16e5c3bc1825" />
      <compositenode searchoperator="SearchOperator:OR">
        <searchtermtype qname="{http://www.alfresco.org/model/content/1.0}folder" />
        <searchtermtype qname="{http://www.alfresco.org/model/content/1.0}content" />
      </compositenode>
      <searchtermdatetime startdate="2017-09-07T00:00:00+02:00" qname="
{http://www.alfresco.org/model/content/1.0}created" />
    </compositenode>
  </searchnode>
</savedsearch>
```

7.6. ZOOMED VIEW

You can access the zoomed view by selecting the "zoom" option from the context menu by right clicking on a folder. This view will show that folder as a virtual root.

Within this zoomed view you can browse the folder tree but you can also filter on categories. Only nodes of the selected category will be shown. The categories you can choose from are configurable. By default it uses the standard category tree from Alfresco ("General classifiable").

Example Spring.NET configuration for custom category tree

My custom example document model

```
<?xml version="1.0" encoding="UTF-8"?>
<model name="fredExp:model" xmlns="http://www.alfresco.org/model/dictionary/1.0">

    <description>My custom document model</description>
    <author>author</author>
    <version>0.1</version>

    <imports>
        <import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d" />
        <import uri="http://www.alfresco.org/model/content/1.0" prefix="cm" />
    </imports>

    <namespaces>
        <namespace uri="http://www.xenit.eu/fred/customcategory/model/0.1"
prefix="custcat" />
    </namespaces>

    <aspects>
       <aspect name="custcat:myCustomCategoriesAspect">
           <title>My custom categories aspect</title>
           <parent>cm:classifiable</parent>
           <properties>
               <property name="custcat:myCustomCategoriesProperty">
                   <title>My custom categories property</title>
                   <type>d:category</type>
                   <mandatory>true</mandatory>
                   <multiple>false</multiple>
                   <index enabled="true">
                       <atomic>true</atomic>
                       <stored>true</stored>
                       <tokenised>false</tokenised>
                   </index>
               </property>
           </properties>
       </aspect>
    </aspects>
```

```
</model>
```

Spring.NET configuration

```
     <object id="path-provider.category-repo-node"

             type="Xenit.Fred.Session.ControllerPath.Category.CategoryPathProvider">

         <property name="DictionaryService" ref="repository.service.dictionary" />

         <property name="NodeService" ref="repository.service.node" />

         <property name="ConfigService" ref="repository.service.config" />

         <property name="CategoryService" ref="repository.service.category" />

         <property name="CategoryPropertyQname" value="
{http://my.custom.namespace}myCustomCategoriesProperty" />

     </object>

     <object id="workspace.zoom.tree.zoom"


type="Xenit.Fred.Session.Controller.ZoomBrowser.Workspace.ZoomCategoryTreeWorkspace"

             parent="workspace.session.base"

             singleton="false">

         <property name="CategoryService" ref="repository.service.category" />

         <property name="DictionaryService" ref="repository.service.dictionary" />

         <property name="Classification" value="
{http://my.custom.namespace}myCustomCategoriesAspect" />

     </object>
```

8. METADATA INJECTOR

The metadata injector is capable of writing metadata into documents. This allows templates to be filled in with the metadata.

8.1. DIFFERENCE BETWEEN ALFRED DESKTOP/FRED BACKEND VERSIONS.

Since Fred backend version 3.4, the metadata injection is no longer available in the backend and is shipped as a separated amp.
The configuration file for metadata injection can be found at webapps/alfresco/WEB-INF/classes/alfresco/module/xenit-metadata-injection/module-context.xml.
You should not make any changes to this file, since all changes will be undone when the alfresco.war file get re-extracted (ex. when a new amp is applied).
The configuration file is no longer at webapps/alfresco/WEB-INF/classes/alfresco/ module/xenit-fred/metadata-context.xml, as the metadata-injection is no longer part of the Alfred Desktop/Fred backend amp.

8.2. DOCUMENT TYPES AND INJECTORS

The following document types are supported by the following components

- MS Office (.docx) with *injector.POI*
- MS Office 97-2003 (.doc) with *injector.Office*
- OpenDocumentLibre/OpenOffice (.odt) with *injector.OpenDocument*

These components are by default allowed in the xenit-metadata-injection/module-context.xml,

```
<!-- Content Metadata Injectors -->
<beanid="injector.Office"class="eu.xenit.fred.alfresco.metadata.injector.OfficeMetadata
Injector"
parent="baseMetadataInjector"/>

<beanid="injector.POI"class="eu.xenit.fred.alfresco.metadata.injector.PoiMetadataInject
or"
parent="baseMetadataInjector"/>

<beanid="injector.OpenDocument"
class="eu.xenit.fred.alfresco.metadata.injector.OpenDocumentMetaDataInjector"
parent="baseMetadataInjector"/>
```

8.3. CONFIGURATION

You can enable the metadata injection by ensuring the metadata amp is installed and then setting eu.xenit.alfresco.metadata.injector.enabled=true in the alfresco-global.properties.

Every property that needs to be injected into the document is linked with a *property-provider*. This propertyprovider is responsible to resolve the values that will be injected in the document.

- The default property-provider will return the straight text-presentation of the property value.
- The defaultEmpty property-provider resolves to an empty string, if the metadata property is not present on the node in Alfresco.
  These properties can be configured by overriding the bean "baseMetadataInjector" in a new 'metadata-injector-context.xml' file under the shared/ classpath as shown in the snippet below.

Create a new file named 'metadata-injector-context.xml' in {TOMCAT_HOME}/shared/classes/alfresco/extension/ and use the following example snippet

8.3.1. SNIPPET METADATA-INJECTOR-CONTEXT.XML

```
<!-- Abstract bean definition defining base definition for all metadata extractors -->
<bean id="baseMetadataInjector"
      class="eu.xenit.alfresco.metadata.AbstractMappingMetadataInjector"
      abstract="true"
      init-method="register">
  <property name="registry">
     <ref bean="metadataInjectorRegistry"/>
  </property>
  <property name="dictionaryService">
     <ref bean="dictionaryService"/>
  </property>
  <property name="namespaceService">
    <ref bean="namespaceService"/>
  </property>
  <property name="mappingProperties">
    <map key-type="java.lang.String" value-
type="eu.xenit.fred.alfresco.metadata.injector.PropertyProvider.IPropertyProvider">
      <!-- defaultEmpty will cause the field to be injected as an empty string if the
property does not exist -->
      <entry key="cm:name" value-
ref="eu.xenit.metadata.injector.propertyprovider.defaultEmpty" />
      <!-- default property provider will not inject field if it is not found for a
specific node -->
      <entry key="cm:author" value-
ref="eu.xenit.metadata.injector.propertyprovider.default"/>
      <entry key="cm:creator" value-
ref="eu.xenit.metadata.injector.propertyprovider.default"/>
      <entry key="cm:created" value-
ref="eu.xenit.metadata.injector.propertyprovider.default"/>
      <entry key="cm:modifier" value-
ref="eu.xenit.metadata.injector.propertyprovider.default"/>
      <entry key="cm:modified" value-
ref="eu.xenit.metadata.injector.propertyprovider.default"/>
```

```
      <entry key="cm:title" value-
ref="eu.xenit.metadata.injector.propertyprovider.default"/>
      <entry key="cm:versionLabel" value-
ref="eu.xenit.metadata.injector.propertyprovider.versionLabel"/>
      <entry key="cm:description" value-
ref="eu.xenit.metadata.injector.propertyprovider.description"/>
      <!-- add your custom properties here, use
eu.xenit.metadata.injector.propertyprovider.default provider as the default provider is
recommended -->
    </map>
  </property>
  <!-- extra node properties -->
  <property name="properties">
    <list>
      <ref bean="eu.xenit.metadata.injector.propertyprovider.noderef"/>
      <ref bean="eu.xenit.metadata.injector.propertyprovider.location"/>
    </list>
  </property>
</bean>
```

8.3.2. PROPERTYPROVIDERS

The available PropertyProvider beans are:

- eu.xenit.alfresco.metadata.injector.propertyprovider.default
- eu.xenit.alfresco.metadata.injector.propertyprovider.versionLabel
- eu.xenit.alfresco.metadata.injector.propertyprovider.description
- eu.xenit.alfresco.metadata.injector.propertyprovider.noderef
- eu.xenit.alfresco.metadata.injector.propertyprovider.location
- eu.xenit.alfresco.metadata.injector.propertyprovider.defaultEmpty
- eu.xenit.alfresco.metadata.injector.propertyprovider.categoryname

9. ALFRESCO POLICIES

9.1. EMAIL .MSG RENAMING POLICY

- Default: enabled
- Configuration: alfresco-global.properties -- eu.xenit.policy.emailnaming=false|true

9.1.1. DESCRIPTION

When you upload Outlook emails (.msg-files) to Alfresco, the filename would default to {email-subject}.msg, which can be quite problematic if you want to save multiple emails with the same subject in the same folder in Alfresco.

To resolve that issue, the Alfred Desktop backend installs an Alfresco policy that will extract the sent-date from the Outlook email and renames the email in Alfresco to "{subject} - {cm:sentdate}.msg" when a new email is uploaded.

9.1.2. CONFIGURATION

To disable this policy, you set add eu.xenit.policy.emailnaming=false in the alfresco-global.properties file.

The default value is true.

9.2. METADATA AUTOCOMPLETE

- Default: disabled
- Configuration: alfresco-global.properties (server)
- Requires: Solr

9.2.1. DESCRIPTION

Autocomplete can be enabled by defining the allowed groups in alfresco-global.properties.
When editing an untokenized property, an autocomplete suggestion menu will popup while typing.
The tokenization of the property has to be set to **false** or **both**. In order to do this your
property in the metadata model has to look similar to this:

```
<property name="fred:example_autocomplete_property">
    <title>Autocomplete Property</title>
    <type>d:text</type>
    <index enabled="true">
        <atomic>true</atomic>
        <stored>false</stored>
        <tokenised>both</tokenised>
    </index>
</property>
```

As you can see in the snippet, tokenised is set to **both**.

9.2.2. SERVER CONFIGURATION

Autocomplete skips the permission checks, suggestions might come from more secure nodes.
Therefore it is important to take access control into consideration.
You will have to add a configuration similar to the following example to alfresco-global.properties on the server.

Example configuration:

- eu.xenit.metadata.autocomplete.authorities=Group-names
    - Example value: GROUP_autocomplete,GROUP_exampleGroup

Tip:
You can create an "autocomplete" group on Alfresco and add that group to the configurated autocomplete authorities.
You can then add users/groups to the autocomplete group via the admin tools of Alfresco Share. The advantage of this
is that you don't need to restart Alfresco because you don't have to edit alfresco-global.properties if you want to
allow autocomplete for certain people.

9.3. INCOMPLETE METADATA POLICY

- Default: disabled
- Configuration: alfresco-global.properties (server) & config.js (for client behavior)
- Requires applying the Incomplete Metadata Tagger amp

9.3.1. DESCRIPTION

By default Alfred Desktop is quite liberal in what a user does with the metadata in Alfresco. Alfred Desktop asks the user to enter/edit the metadata when he uploads a new document, but doesn't really enforce this.

Organizations that would like to enforce a more strict company-policy on metadata can enable the incomplete-metadata-policy.
This policy will apply the *xmd:incompleteaspect* to documents that don't have all mandatory metadata properties set or to a document where the type is one of the *eu.xenit.incomplete.invalidTypes* types. This allows Alfred Desktop/Alfresco to keep track of which user introduced a document into the repository and who is responsible to provide the metadata for a specific document

You can enable this functionality on a folder (-substructure) by applying the (configurable) validation-aspect on said folders. Alfred Desktop comes with a dictionary model that contains the aspect *xmd:validate* to do just that, but you can use any other aspect you'd like to use, including from your own dictionary model.

In practice, this means if a user uploads a document in a folder-structure that has the metadata-validation-marker set, the user is “required” to select a specific node-type and has to provide all mandatory metadata on this document. The document is marked in Alfresco as metadata-incomplete and the user that uploaded the document is the metadata-owner.

Using the property *deepMarkerSearch* you can configure this policy to look only at its direct parent or walk up the folder hierarchy to see if any of the parent folders has the validation-marker set.

9.3.2. INSTALLATION AND SERVER CONFIGURATION

This functionality used to be part of the Alfred Desktop backend .amp up until version 3.4. It is now separated out in a module you have to install next to the Alfred Desktop module.
If your Fred backend version is below 3.5 you will first have to install the Incomplete Metadata Tagger module in order to use this policy.

To configure the incomplete metadata policy add the following lines in the alfresco-global.properties

- eu.xenit.policy.metadata.incomplete.enabled=false|true
- eu.xenit.policy.metadata.incomplete.markerAspects=aspect-qname
    - Example value: xmd:validate
- eu.xenit.policy.metadata.incomplete.excludeAspects=aspect-qname
    - Example value: xmd:ignore
- eu.xenit.policy.metadata.incomplete.invalidTypes=type-qnames
    - Example value: cm:content
    - Example value: cm:content,myCorp:document
- eu.xenit.policy.metadata.incomplete.deepMarkerSearch=true|false
- eu.xenit.policy.metadata.incomplete.propertiesToIgnore=property-qnames

Set the enabled property to true to enable the incomplete metadata policy. In the *markerAspects* you can insert a list of qnames that are marker aspects. Folders that have this marker aspect will be scanned for incomplete metadata. This includes subfolders of the folder that has the marker aspect. To prevent a subfolder to be scanned, you can mark it with an exclude aspect. The *invalidTypes* property is a comma-separated list of node-types that are too abstract and should be flagged as incomplete. To check the entire parent hierarchy for a marker aspect, set the deepMarkerSearch to true. Mandatory properties that need to be ignored can be added to the *propertiesToIngnore* list.

Note: QNames can be short or fully qualified.

9.3.3. CLIENT CONFIGURATION

The Alfred Desktop client behavior can be configured through the extended object in config.js. All these values are optional.
With the default client-configuration, the mandatory-metadata tracking makes the Alfred Desktop client keep track of the documents with missing metadata and are shown in the bottom panel.

Paramter Name	Type	Description

Paramter Name	Type	Description
incomplete.tracker.interval	number (int)	Defines the interval (in minutes) that Alfred Desktop polls Alfresco for metadata updates on incomplete-metadata objects. The default value is 2 (minutes). If the users use Alfred Desktop as the primary interface to Alfresco, you can set a higher value. If the users use other interfaces (Share, WebDAV, ...) together with Alfred Desktop, you might want to keep this value low.
incomplete.ui.enable-missing-tab	boolean	This enables a menu item under Window, labeled "Missing Metadata". This menu item opens a new tab, which lists the documents of the user with missing metadata.

Limiting uploads

The user is still free to ignore this bottom panel. For most deployments, this is sufficient.

However, there is another option available to force the user to provide metadata: there can be limits applied to how many documents with missing-metadata can be in his/her queue, before Fred refuses to upload more documents.

Paramter Name	Type	Description
incomplete.limit.soft	number (int)	Soft limit on the number of objects in the users' incomplete-metadata queue, before Alfred Desktop refuses to upload more content. The default (unset) value is unlimited.
incomplete.limit.hard	number (int)	Hard limit on the number of objects in the users' incomplete-metadata queue, before Alfred Desktop refuses to upload more content. The default (unset) value is unlimited.
incomplete.limit.global	boolean	Supported values: "False" (default), "True" Defines if limits are applied globally or only in folder-contexts that have mandatory-metadata applied.

9.4. CONTENT HASHING POLICY

A new feature introduced in Alfred Desktop 3.6 is exporting and locking a folder and after that importing and releasing that same folder.
When exporting, the folder is exported as a zip. You can extract it and make changes to the files inside that folder. By using a hashing policy, Alfred Desktop can keep track of the changes that were made to each file and detect if files are added or deleted.

The hashing policy will calculate the hash every time a file is uploaded or the content is edited. Then the hash value is added to the property of a hashing aspect.

9.4.1. SERVER CONFIGURATION

The configuration for this policy is located in alfresco-global.properties.
There you add the following line: eu.xenit.hash.supported-algorithm=sha-1

At this point only sha-1 is supported but in the future some other algorithms can be supported as well.

10. WORKING OFFLINE

- Default: enabled
- Configuration: *config.js*

10.1. DESCRIPTION

It is possible for a user to take folders with content offline. This gives the user the possibility to proceed his work without a connection to the repository (ex. while traveling). Taking a folder offline downloads all the content within that folder to the user's local hard drive.

Offline folders are added to the user's Favorites. And an icon indicates which favorite folders are offline available. In the context menu of an offline available folder the "Take Offline" command has a checked checkbox in front of it. If the "Take Offline" command is selected again, the folder will be no longer offline available.

Each time when the user connects to the repository the offline folders will be synchronized with the repository. A user can manually start the synchronization by selecting "Refresh Offline" in the context menu of the offline folder.

When a user has taken folders offline and connects to the relevant bookmark in offline mode. The content within the offline folders will be available for read-only.

10.1.1. CONFIGURATION

This can be configured in the config.js configuration file, under the "extended" object.

Paramter Name	Type	Description
offline.read.enabled	boolean	Supported values: "False", "True" (default)</br />Defines if the user is able to take folders offline.
offline.write.enabled	boolean	Supported values: "False" (default), "True" Defines if the user is able to make changes to the offline documents while in offline mode.

This feature is not yet supported!

11. HIDING FOLDERS

Folders can be hidden by using the same mechanism that Alfresco uses. To hide a folder you have to add the sys:hidden aspect to it.

The sys:hidden aspect cannot be added to a folder from within Alfred Desktop. You have to do this by using the javascript console of Alfresco.
This is a code example to add the sys:hidden aspect (or any other aspect) to a folder:

```
var nodeRef = "workspace://SpacesStore/aad16fd3-74fa-45ea-9863-48b325359ccd"; <-- Change this noderef, this is just an example
var nodeToHide = search.findNode("noderef");
nodeToHide.addAspect("sys:hidden");
nodeToHide.save();
```

12. MISCELLANEOUS

12.1. REPORT A PROBLEM

- Default: disabled
- Configuration: config.js

12.1.1. DESCRIPTION

When a user would encounter a problem with Alfred Desktop, there is always a hassle to get the exact version number and application log files.

This feature enabled a "Report A Problem" menu item under the "Help" menu in Alfred Desktop. It opens up the default mail client, sets the addressee to the pre-configured value (preferably your IT-departments email-address) and attaches the Alfred Desktop log file.

This allows your IT-department to evaluate the reported problem and if necessary, forwards this easily to XeniT, without the need to chase the user for client log files.

12.1.2. CONFIGURATION

This can be configured in the config.js configuration file, under the 'extended' object.

Paramter Name	Type	Description
help.report-problem.to	String	Value should be a (valid) email address, for example: it-support@mycomany.com enables a menu item under the Help-menu, called "Report a problem". This will open the user his default email client, attach the Alfred Desktop log file and uses the configured value in the "To:" field.
help.report-problem.cc	String	Value should be a (valid) email address. This value will be added to the "CC" field if the user uses the "Report a problem"

Users will automatically have the menu option available under Help, the next time they connect to the repository.

13. EXTERNAL ADD-ONS

13.1. DIGITAL SIGNATURE

13.1.1. DESCRIPTION

Digital Signature is an add-on for Alfred Desktop that enables you to digitally sign PDF documents.
This feature has a backend component in the form of a Servlet .war, that runs in Tomcat, just like Alfresco. This does not come bundled with the Alfred Desktop backend, but has to be separately installed – and can be installed on a different Tomcat server.
To enable the digital signature add-on in Alfred Desktop, you have to configure the URL where the Digital Signature application can be found, in the config.js configuration file under the extended-object.

Paramter Name	Type	Description
digitalSignature	String	The URLwhere the digital-signature application can be accessed.

Contact fred@xenit.eu for more information.

13.1.2. CONFIGURATION

In config.js, add “digitalSignature” :“<yoururl>” to the extended property (example below).

path: /opt/alfresco/tomcat/shared/classes/alfresco/extension/templates/webscripts/eu/xenit/config/config.js

```
var configByGroup = {
    "default": {
        "namespace"      : "",
        "rootdoctype"    : "{http://www.alfresco.org/model/content/1.0}content",
        "rootnoderef"    : (this.companyhome != null)?companyhome.nodeRef.toString():"",
        "extended"       : {
    "digitalSignature" : "http://localhost:8080/ops-xenit"
}
    },
};
```

13.2. DAEJA VIEWER

Alfred Desktop has integration support for DaejaViewONE, a webbased viewer/annotation tool.
Contact fred@xenit.eu for more information.