Content Conversion Specialists

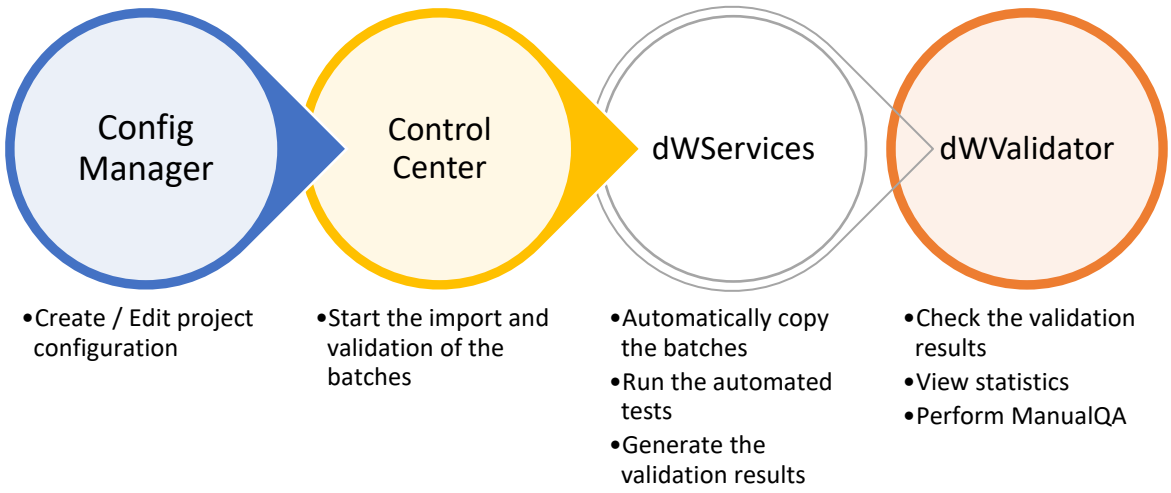
Configuration Manager

Contents

Last updated: 06/29/2021

Get started

Basic workflow



The standard workflow for the validation process consists in 4 stages:

1. Create / Edit the project configuration that will be used.

With Configuration Manager you can create and edit project configurations. Use this application to group files on specific categories (docblocks), create automatic and manual tests and define the manual QA selection rules. *For more details, see Configuration Manager Help*.

2. Start the import and validation of the batches.

In Control Center you simply add the path of the batch, the destination path and project configuration that will be used and trigger the validation process. *For more details, see Control Center Help*.

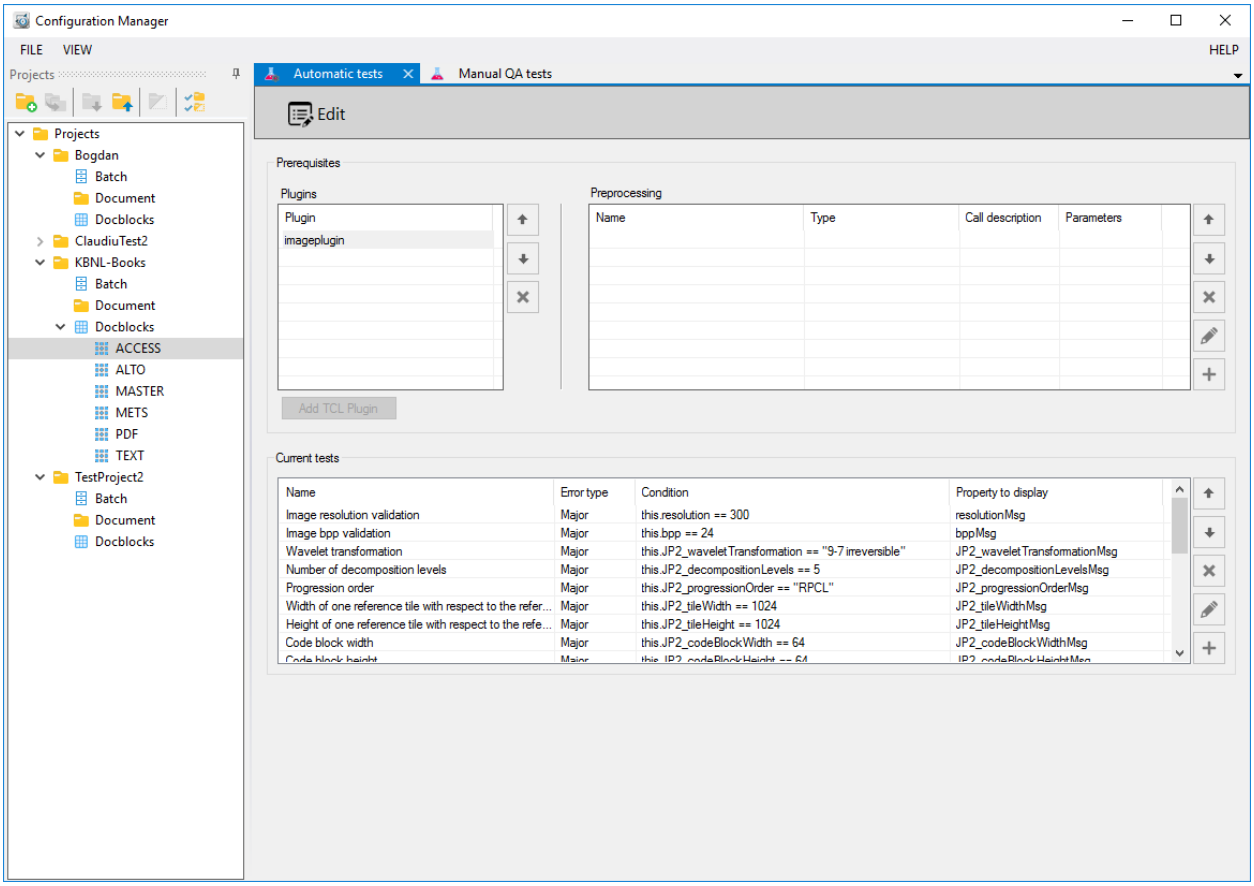
3. Check the progress of the automatic validation.

The dWServices are background services – they perform the tasks automatically, without any user interaction. The services will automatically copy the batch to the destination path and create the validation tasks. These tasks are also automatically processed by the services and the results are added to the Database. *For more details, see dWServices Help*.

4. Check the results and statistics.

Use dWValidator to check the automatic tests results and statistics. There are detailed reports on every level – from batch level to file level – that can be used to check what tests failed and why. The file itself can also be inspected to double check the results. The same application can be used to perform the manual tests, each file type having a set of tests to be performed, defined in project configuration. *For more details, see dWValidator Help*.

Configuration Manager



Configuration Manager was designed to make project configuration as easy as possible.

Because parts of a project configuration are kept in .xml files and other parts are saved in database, it's easy to miss-configure something that will eventually lead to errors. Configuration Manager gathers all the necessary data that you need to configure into one application.

Structure of a project configuration

A project is split into 3 items in Configuration Manager:

- **Batch** – where you can set up Automatic and Manual tests at batch level, configure batch specific file groups (docblocks) and define the Manual QA selection rule
- **Document** – where you can set up Automatic and Manual tests at document level
- **Docblocks** – is the level where you can add / remove document specific file groups (docblocks).

For each docblock added, a separate node will be created inside the “Docblocks” / “Batch” node in tree.
For more details, see Docblocks view Help.

Each docblock will have views where Automatic and Manual tests can be defined at that docblock level.

The interface

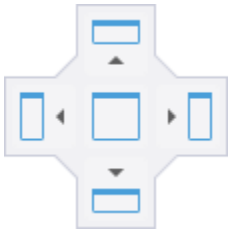
Window styles

In Configuration Manager you can customize the position, size and behavior of windows, to create a layout that best suits your needs. The interface has two basic window types, *tool window* (Projects tree, Help window) and *document window* – or view (Automatic tests window, Manual QA tests window, Docblocks window).

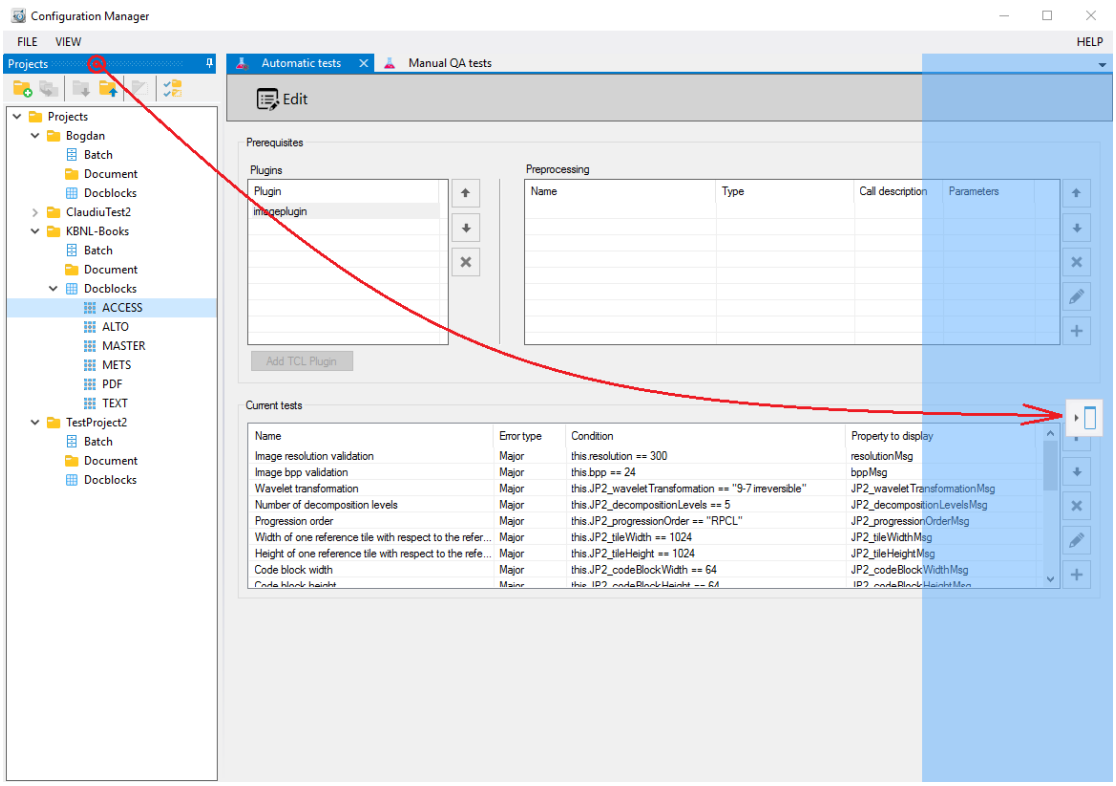
Tool windows can be resized and dragged by their title bar, but their docking positions are more restrictive and they cannot be left floating. Document windows can be dragged by their tab and docked anywhere, they even can be left floating.

Docking

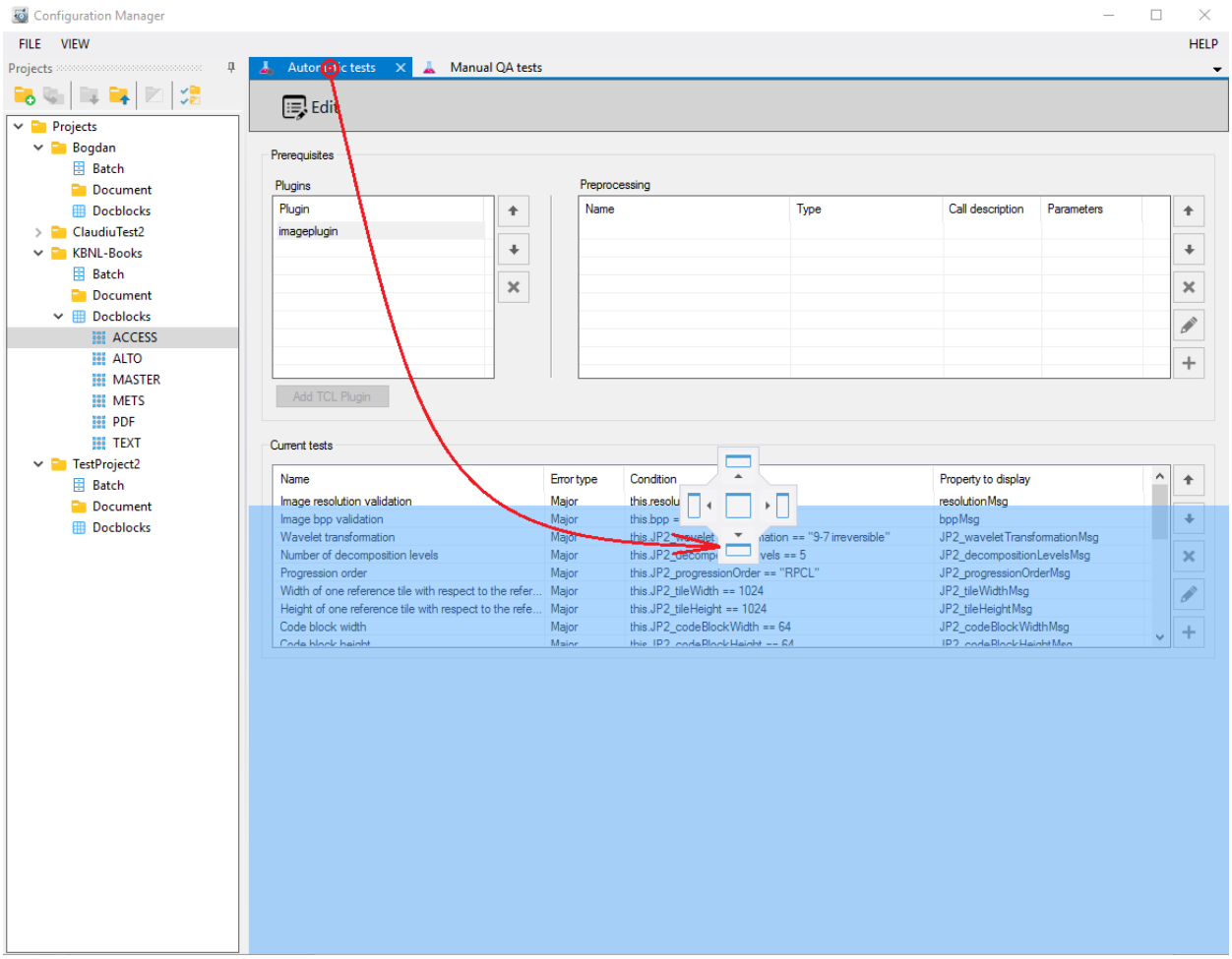
While dragging a window, check for a dock indicator where it can be dropped and hover that indicator. A blue highlighted area will appear, showing you where the window will be docked if you release the mouse button now.



Example of how to dock Projects tree on the right side:



Example of how to split the view to see both document windows:

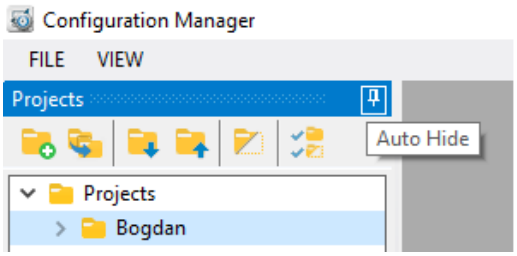


To move a dockable window without snapping it into place, hold the **Ctrl** key while you drag the window. This will prevent the window to dock to a position, leaving it floating (if the window allows it).

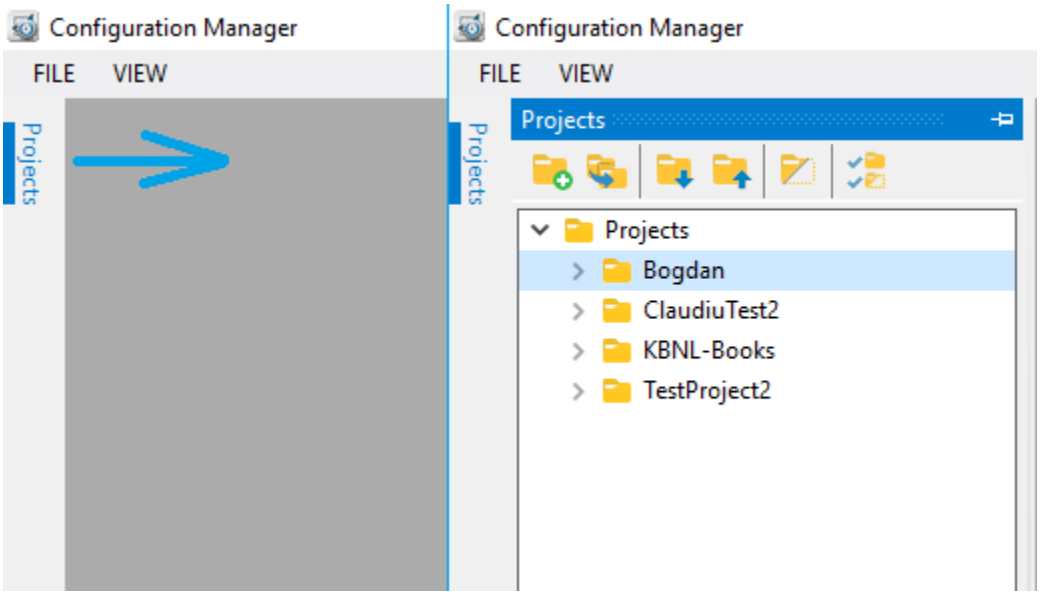
Closing and auto-hiding

Any window from Configuration Manager can be closed by clicking the X on the upper right corner of the title bar (or tab), except the Projects tree – this window doesn't have a close button.

Tool windows also have an "auto-hide" feature, which will make a window slide out of the way when you use a different window. To hide a window, press the "pin" button from the title bar:



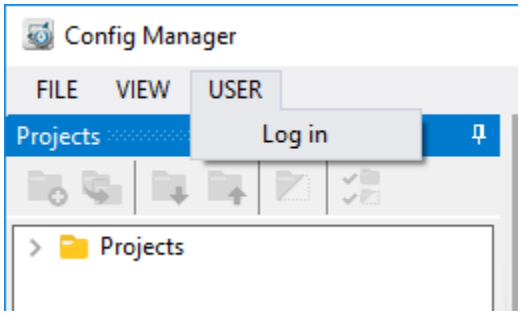
When a tool window is auto-hidden, its name appears on a “hide bar”, a tab on the edge of the application. To use that window again, point to the tab so that the window slides back into view.



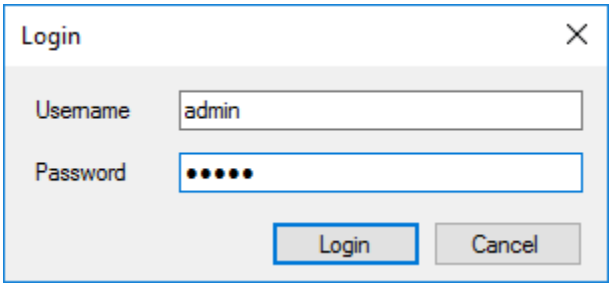
When the mouse cursor leaves the window area, it will automatically hide again. To make the window active again (stop it from auto-hide), press the “pin” button.

Users

Configuration Manager requires credentials in order for changes to be made to projects. The necessary right for this is "**User can change settings in project configuration**". If you have this kind of user, you can log in using **Log in** option from **USER** menu:

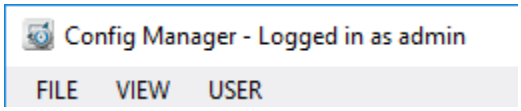


You will be prompted with a dialog to enter your credentials:



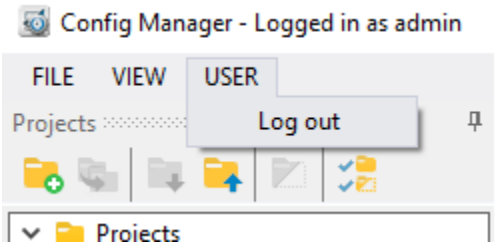
Error messages will appear if the user name you're trying to log in with doesn't exist or the typed password is not correct.

Once you're successfully logged in, your username will appear in the application title bar.

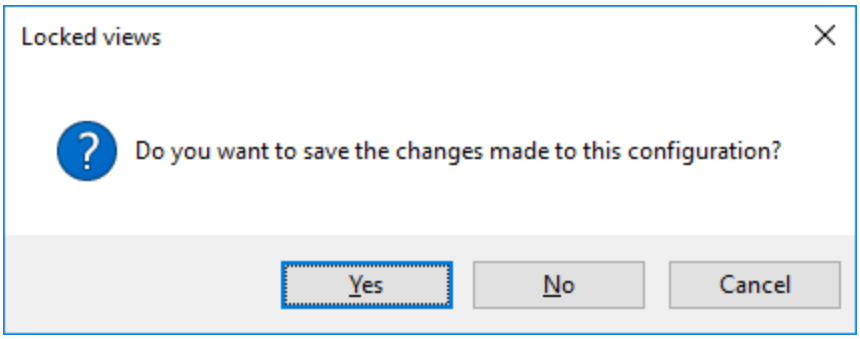


If the Projects tree buttons and Editing buttons are still disabled, it means that your user doesn't have the necessary right.

You will always have the option to log out in **USER** menu:



If you're editing a project configuration and you choose to log out, you will be asked if you want to save the changes made.



Selecting **Yes** will save the changes and log you out of the application, selecting **No** will discard all the changes and log you out of the application.

Editing toolbar

When selecting a node from the tree, all views (document windows) that are opened for that node are displayed as read-only, to prevent users making accidental changes. An editing toolbar will be displayed at the top of each window.

In order for any change to be made, the view has to be "locked for editing", using **Edit** button.

> ***Edit** button can be disabled for two reasons: either you are not logged in, or your user account doesn't have the necessary right to modify configuration. For more details, see* Users Help.

Once the view is locked, two more buttons will appear (**Submit** and **Cancel**), and also an icon indicating the locked state.

The only exception in buttons displayed on the editing toolbar is on "Batch configuration" view for batch level – here an extra button will be displayed: **ManualQA selection**. *For more details, see ManualQA selection Help.*

Edit button

- Enables the controls of the view so changes can be made; on batch level, it also enables **ManualQA selection** button
- Displays and enables **Submit** and **Cancel** buttons
- Displays the "Lock" icon to highlight the state of the document window
- Prevents user to change the node in tree or close the application

Submit button:

- Saves all the changes made in the view
- Updates Database and / or xml configuration files with the changes made
- Sets the view in read-only mode again
- Hides itself, the **Cancel** button and the "Lock" icon

Cancel button:

- Discards any changed made since **Edit** button was pressed
- Sets the view in read-only mode again
- Hides itself, **Submit** button and the "Lock" icon

Handling projects

Projects tree

It contains a tree-like structure with the projects found both in Database and in configuration folder.

Projects tree has only two docking areas: left side and right side and cannot be closed. It can be, however, hidden, by using the “pin” button.

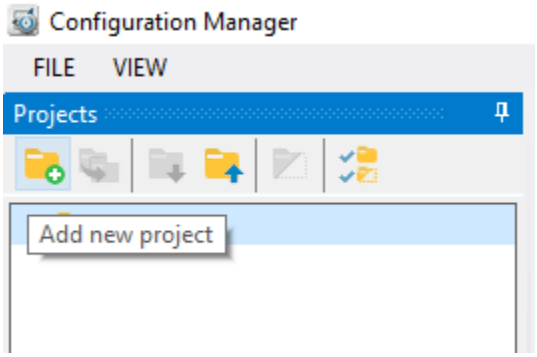
On top of Project tree window there are six buttons:

- **Add new project** - used to create new project configurations. *For more details, see Create a new project configuration Help.*
- **Copy project** - used to create a new project configuration based on an existing one. *For more details, see Copy a project configuration Help.*
- **Export project** - used to create a backup of the entire project in order to restore it on another environment. *For more details, see Export a project configuration Help.*
- **Import project** - used to add a project from a backup file. *For more details, see Import a project configuration Help.*
- **Hide project** - used to hide the selected project. *For more details, see Hide a project configuration Help.*
- **Show all** – used to display all / only active projects. *For more details, see Hide a project configuration Help.*

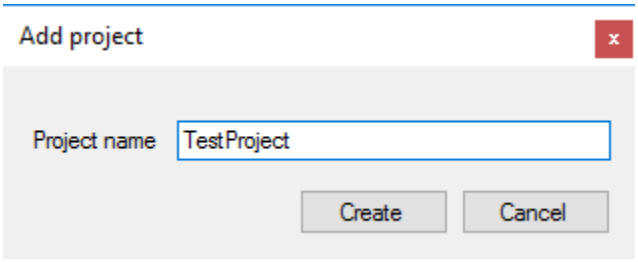
Tree buttons can be disabled for two reasons: either you are not logged in, or your user account doesn’t have the necessary right to modify configuration. For more details, see Users Help.

Create a new project configuration

To create a new project configuration, use **Add new project** button, located at the top of the Projects tree.



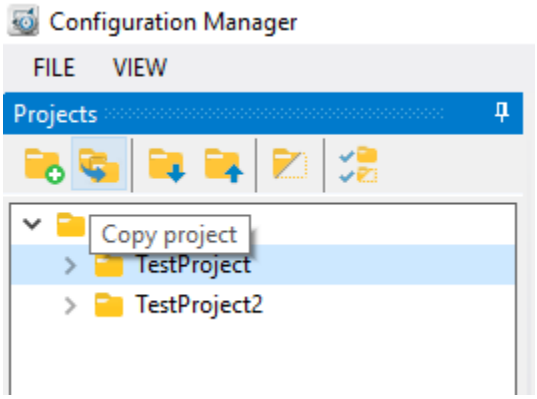
You will be prompted with a dialog where you need to add the project name:



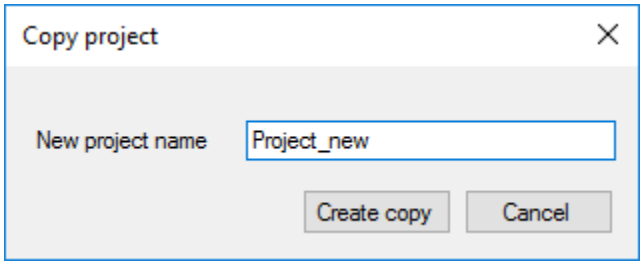
Press **Create** button and you are done. Your new project configuration is now available for you to edit.

Copy a project configuration

You can create a new project configuration and use all the settings of an existing project by selecting a project from Projects tree and clicking **Copy project** button.



You will be prompted with a dialog where you need to add the project name:



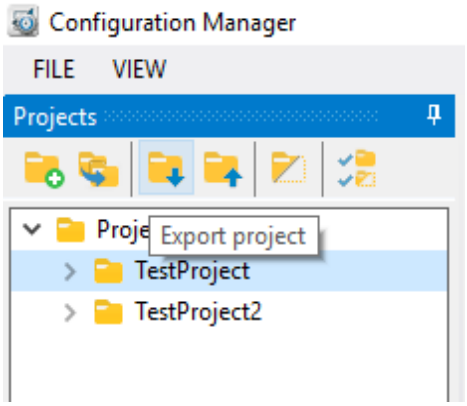
Press **Create copy** button to generate a new project configuration that will have all the docblocks, automatic and manual tests, exploration and manual QA selection rules as the selected project.

*If you receive a warning message saying that the project already exists but you cannot see the project, you can click **Show all** to display also the hidden projects. For more details, see* Hide a project configuration Help.

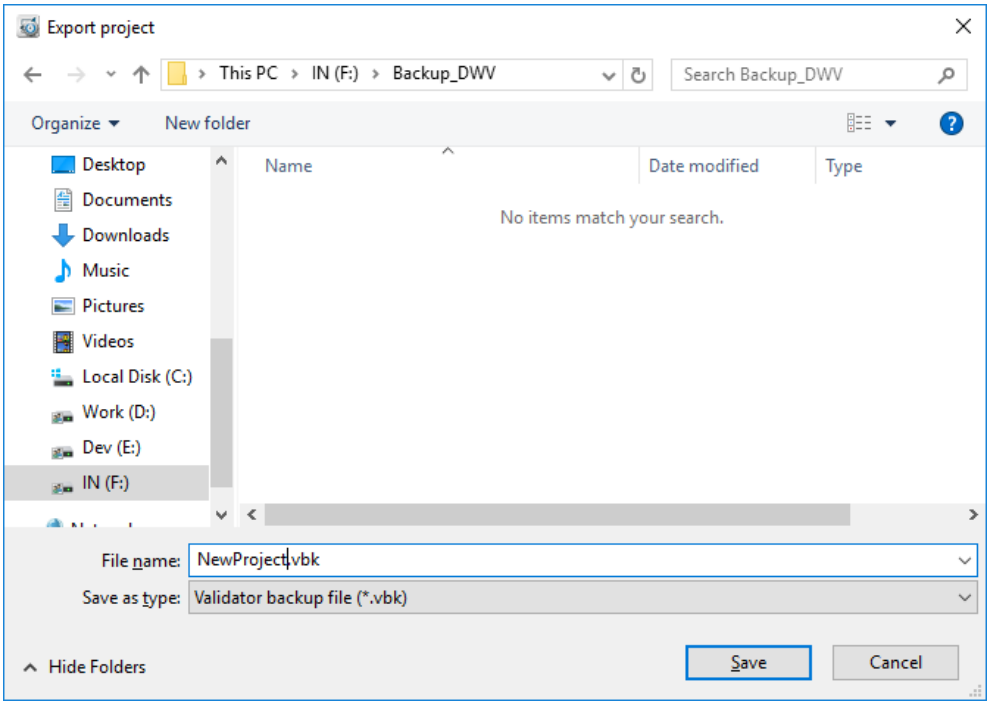
Export a project configuration

If you want to move a project configuration from an environment to another, you can create a backup file of the project, that you can later import it to a new environment (*for more details, see Import a project configuration Help*).

Select the desired project from Projects tree and click **Export project** button.



This will open a dialog where you can choose where to save the file and how to name it:



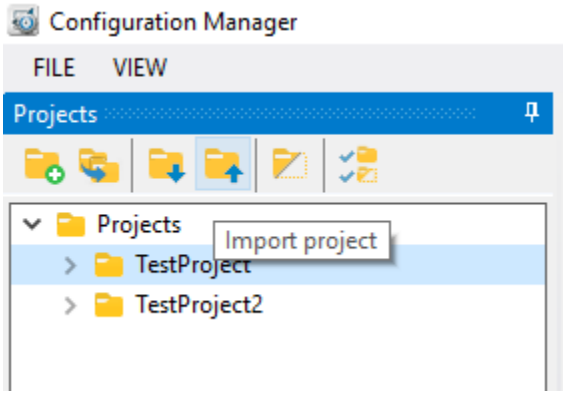
The name you set for the file is not relevant for the application, because the real name of the project is saved inside the backup file. When importing the project, it will have the saved name.

dWValidator uses **.vbk** extension for the backup files, so make sure that you set the correct extension in the file name. Press **Save** to create the backup file.

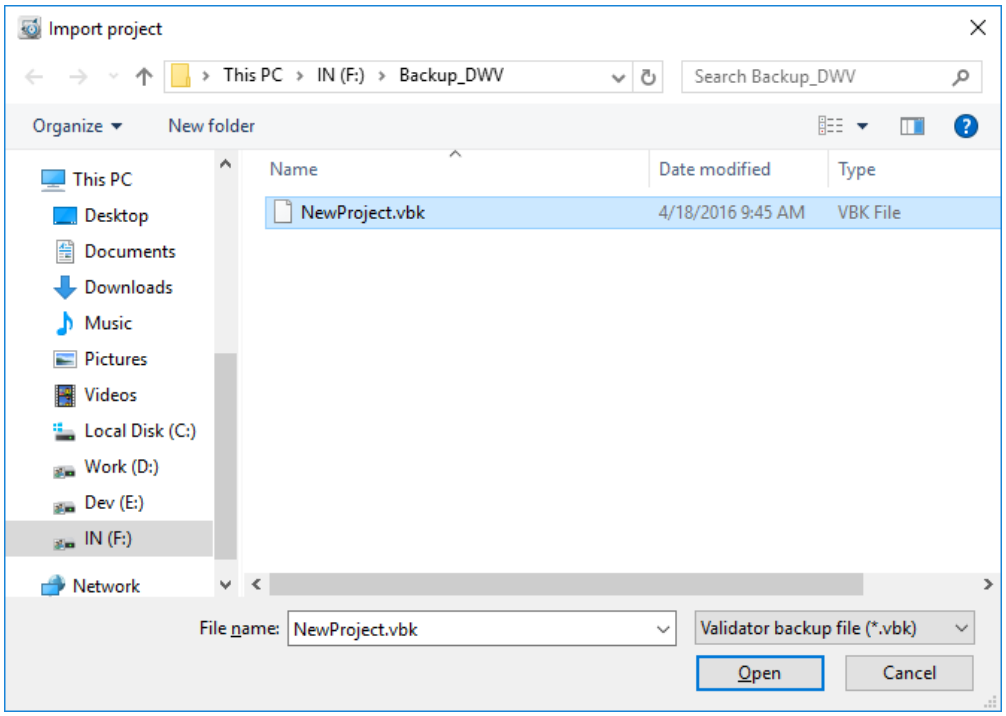
Import a project configuration

If you want to move a project configuration from an environment to another, you can create a backup file of the project (*for more details, see Export a project configuration Help*), that you can later import it to a new environment.

If you already have a backup file, click on **Import project** button from Projects tree:



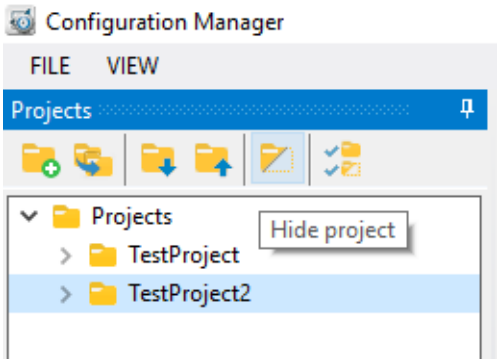
This will open a dialog that you can use to browse for your backup file:



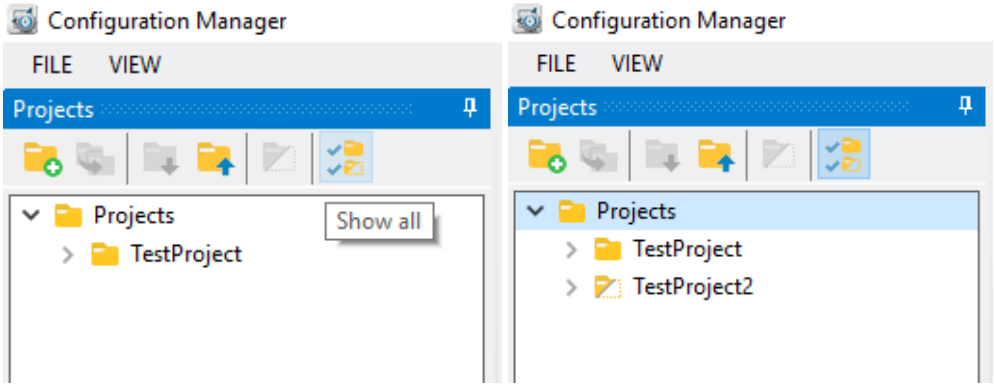
Make sure that the file you select has **.vbk** extension, then press **Open** button to restore the project configuration. Your restored project will appear in Projects tree.

Hide a project configuration

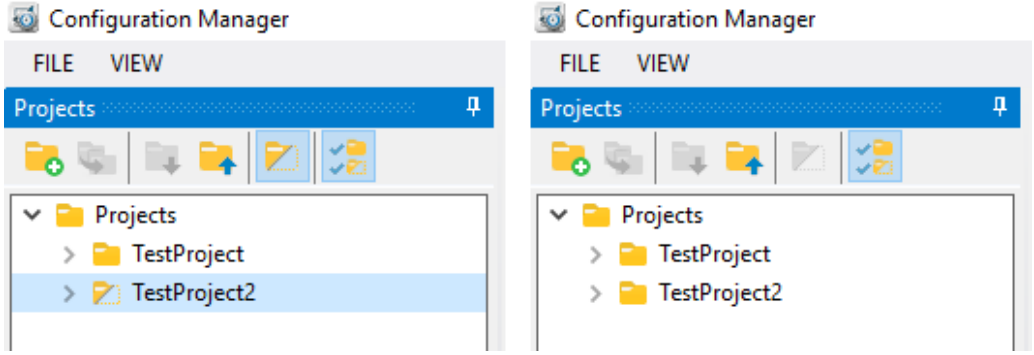
You can hide a project configuration that you're not using at the moment so it no longer appears in tree and will also be hidden in dWValidator. To hide a project configuration, select from Projects tree the project folder and click **Hide project** button.



This will mark the project as "hidden" and the project will no longer show up in Projects tree. You also have the option to view all projects, even the hidden ones, by clicking **Show all** button:

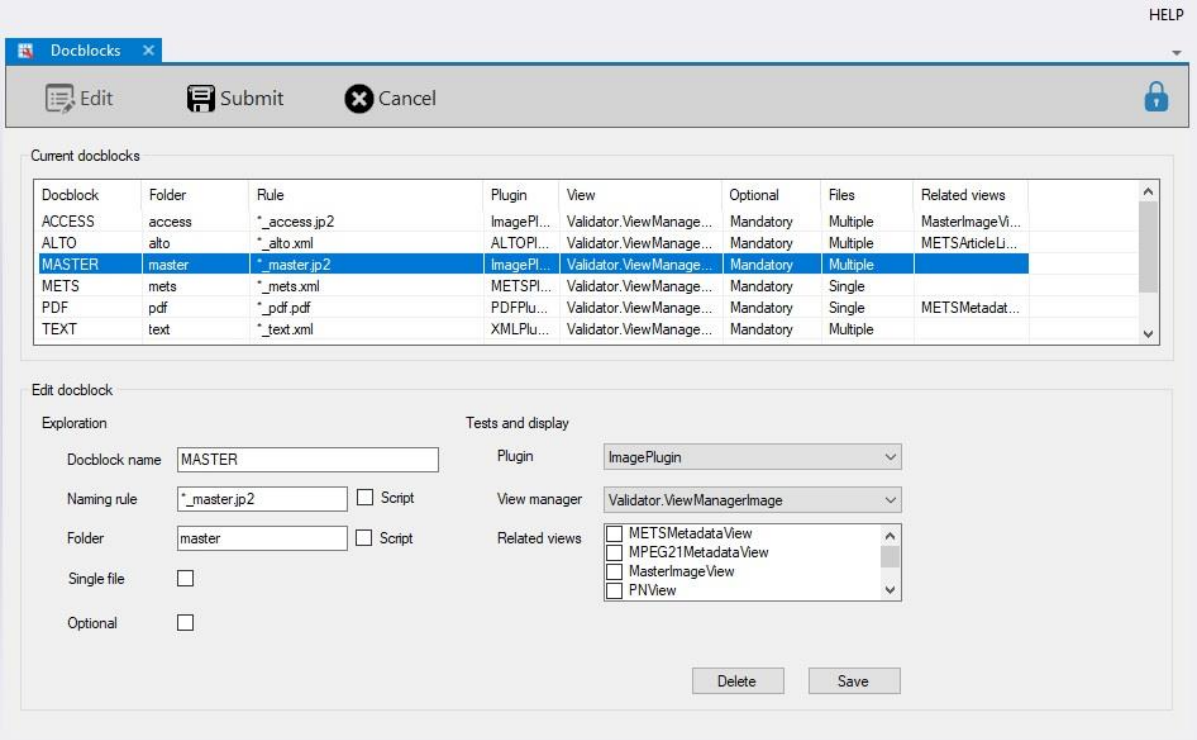


This action will display the hidden projects in tree – can be identified by their icon: . Any batch imported with this configuration will still be hidden in dWValidator. You can make a project active again, by selecting it from Projects tree (when **Show all** is activated), then pressing again **Hide project** button. Un-hiding a project will also make available any batch that was imported with that configuration in dWValidator.



Docblocks view

A “docblock” is a section of a document that contains all the data of a certain type, having file naming similar with the description mask. The docblock with its properties will be the same for all documents of that project. For example, a JP2FILE docblock will represent all the .jp2 file of a document.



The Docblocks view contains:

- Editing toolbar – *for more details, see Editing toolbar Help*.
- A list with the docblocks defined for the project
- Controls used to edit the docblocks.

Add / edit docblocks

To add a new docblock to a project configuration, you need to lock the view for editing, by pressing **Edit** button. In the list, select the first empty row; if the list is empty, select the first row.

In the controls under the list, add or edit the properties of the docblock:

- **Docblock name** – The name of the docblock as it will appear in tree. We suggest to use a descriptive name, that best characterize the file type it represents. For example, use TIFFILE for the docblock for .tiff files of the document.
- **Naming rule** – represents the naming characteristic, something that all the files of this type have in common. *For more details, see Define the exploration rules Help.*
- **Folder** – specify the name of the folder inside the document folder, that contains the files of this docblock. If the files are directly in the document folder, leave this field empty. If "Script" check

is marked, the content of Folder/NamingRule field is the name of a TCL procedure, that returns the relative path, starting from document folder or the naming rule. This is very useful on more complex structures when a document is splitted into multiple parts from different root locations, for example.

- **Single file –** use this check to specify if a document from the batch will have only one file of that type. If left unchecked, it means that multiple files of that type are found in a document.
- **Optional** – this check will mark the files of the docblock as optional for the exploration. *For more details, see Define the exploration rules Help.*
- **Plugin** – select the plugin that will be used to perform the automatic tests. *For more details, see Plugins Help*.
- **View manager** – based on the plugin selected, the View manager dropdown will be prefilled. Select the option that best defines the file type of the docblock. *For more details, see Select the View Manager Help*.
- **Related views** – the list will be filled with the configured views, from which you can select one or more entries. If the list is empty, you have no configured related views. *For more details, see Define Related Views Help*.

When all the properties are filled, use **Save** button from **Edit docblock** group to add the docblock in list.

If you want to edit an existing docblock, select it from the list and make the desired changes. Once finished, press **Save** button from **Edit docblock** group to update the docblock details.

To delete a docblock, select it from the list and use **Delete** button from **Edit docblock** group.

To submit the changes made in "Docblocks" view, click **Submit** button from Editing toolbar.

Define the exploration rules

The exploration rules represent the common structure of all documents of the project. These rules are used to gather and set up the data that will be validated. The exploration rules are similar to a pattern that will be applied on all documents of each batch. The documents that don't fit this pattern will not be included in the validation process.

For example, setting up a docblock like this:

Docblock	Folder	Rule	Plugin	View	Optional	Files	Related views
ACCESSFILE	ACCESS	?????.jp2	ImagePlugin	Validator.ViewManagerImage	Mandatory	Multiple	CustomMasterRelated

means that:

- **Folder**: each document has a subfolder called "ACCESS"
- **Rule**: each ACCESS file from this folder has the named structured like: ?????.jp2 (example: 00001.jp2). Any file that doesn't have this name structure will not be included for validation
- **Plugin**: the plugin used to perform the automatic tests will be ImagePlugin (*for more details, see ImagePlugin Help*). There will be a possibility to add also TCLPlugin for this docblock when defining the automatic tests.

- **View**: the images will be displayed in dWValidator using the ViewManagerImage (*for more details, see ViewManagerImage Help*).
- **Optional**:
  - If the value for this field is set to "Mandatory", all documents must have the subfolder "ACCESS" that contains files with the defined naming rule. Any document that doesn't fulfill these rules will not be included for validator.
  - If the value is set to "Optional", some documents can have the subfolder "ACCESS" with the files, other documents might not have it, but they all will be taken into account for validation.
- **Files**:
  - If the value is set to "Multiple", all documents have more than one ACCESS file
  - If the value is set to "Single", all documents have only one ACCESS file
- **Related views**: beside the ACCESS images, the configured related view will also be displayed (in this case, the master file). *See Define related views Help for more information.*

In the naming rule:

- **? –** represents any character from the file name.
- ***** – represents a set of characters from the file name.

Examples of naming rules:

- ????-??-??_??_ALTO.xml (sample file: 1957-05-12_01_ALTO.xml)
- *_PDF.pdf (sample file: 1957-05-12_PDF.pdf; Issue03_PDF.pdf)
- *_????.jp2 (sample file: MasterImage_0001.jp2; 1957-05-12_0025.jp2)

Example of complex folder structure, where folder definition via scripts is needed:

- Considering following folders structure:

```
batch
    MasterFiles
      doc1
        1_00001.tif
        1_00002.tif
      doc2
        2_00001.tif
        2_00002.tif
    AccessFiles
      doc1
        1_00001.jpg
        1_00002.jpg
      doc2
        2_00001.jpg
        2_00002.jpg
```

- Where we have two documents:
  doc1 containing 1_00001.tif, 1_00002.tif, 1_00001.jpg, 1_00002.jpg
  doc2 containing 2_00001.tif, 2_00002.tif, 2_00001.jpg, 2_00002.jpg

- We can define following rules:
  DOCBLOCK MASTER: Folder: no folder; NamingRule: *.tif
  DOCBLOCK ACCESS: Folder: GetAccessFolder; NamingRule: *.jpg; Script flag ON

- Into scripts library we need to add following procedure:

```
proc GetAccessFolder { } {
        global crt_dir
        return "..\\..\\AccessFiles\\[getfilename $crt_dir]"
}
```

crt_dir always contains the path where a document template match is searched at one moment

Select the View manager

A View manager is an internal structure of dWValidator, which tells the application how to display the files of the docblock. Each View manager has different views, defined specifically for a certain file type.

Docblocks need to have a View manager set, in order to correctly display the files of the docblock in dWValidator. Setting the wrong View manager for a docblock will not affect the validation process, however, the files will not be properly displayed in dWValidator and may lead to errors when trying to view them.

The View managers available for docblocks are:

- **ViewManagerImage** – used to display images (.jpeg, .jpg, .bmp, .png, .ico, .tif, .jp2, .cr2, .gif)
- **ViewManagerAlto –** used specifically for ALTO files
- **ViewManagerMets** – used specifically for METS files
- **ViewManagerXML** – used for any .xml file
- **ViewManagerPDF** – used for any .pdf file
- **ViewManagerWeb** – used for .html or .htm files
- **ViewManagerRtf** – used for text documents (.txt, .rtf, .doc)
- **ViewManagerCSVandExcel** – used to display excel sheets and .csv files (.csv, .xslx)
- **ViewManagerEpub** – used to display ePUB files (.epub)
- **ViewManagerGeneric** – used to display any type of file as binary

For more details, see View managers Help.

Define related views

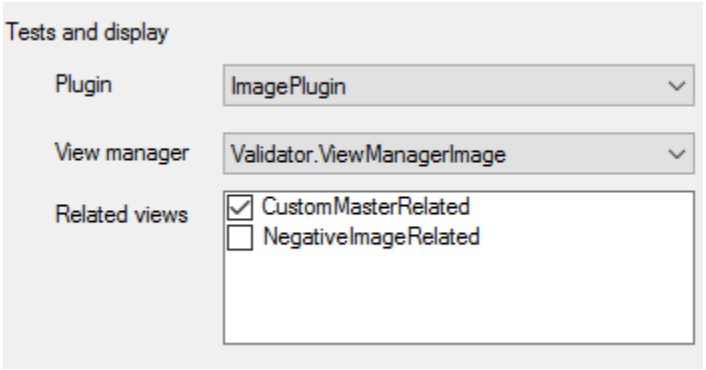
You can configure dWValidator to display one or more files beside the one selected. To do this, you need custom scripting, which is done in *****PROJECT_CFG***\ProjectName\CSScripts\interfaceScript.cs** file (uses C# as language).

There are two types of related views that you can configure: a simple one, which will display the file as is, or a transformed view, where a transformation is applied on the file before it is displayed.

Simple view

```
// example of a simple related class
public class MasterImageView : Scripting.DefaultCCSViewInterface
{
// change the path to the new file that you want to be displayed
	public override string GetFilePath(string _path)
	{
		string fileName = Path.GetFileName(_path);
		string dirName = Path.GetDirectoryName(_path);
		dirName = dirName.Replace("access", "master");
		fileName = fileName.Replace("access", "master");

		return Path.Combine(dirName, fileName);
	}

// set the tab name for the related file
	public override string GetRelatedViewName(int nr)
	{
		return "Master related";
	}

// return one of "Image", "PDF", "XML", "HTML", "Text", "ExcelOrCSV" to know how to display
the file
	public override string GetFileType()
	{
		return "Image";
	}
}
```

First, you need to define a class derived from `Scripting.DefaultCCSViewInterface` – the name of the class will appear in the **Related views** list.



There are three procedures that you need to add to your custom class:

1. `public override string GetFilePath(string _path)`

This procedure needs to return the path of the related file, including the file name.

2. `public override string GetRelatedViewName(int nr)`

This procedure returns the name that will be displayed in dWValidator tab, when showing the file.

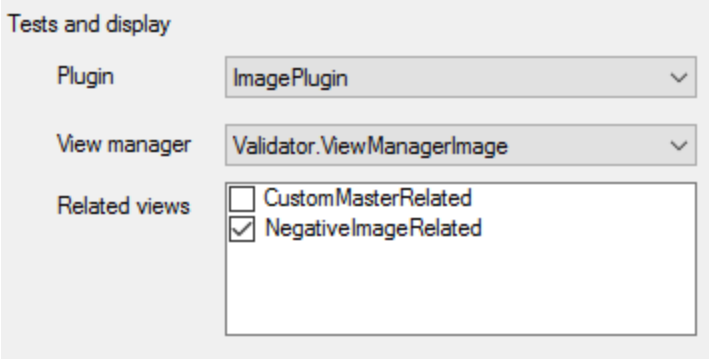
3. `public override string GetFileType()`

This procedure returns the file type. The returned string must be one of the following: Image, PDF, XML, HTML, Text or ExcelOrCSV. Based on this string, a specific View manager will be used to display the file.

Transformed view

```
// example of a related class with a transformed file
public class NegativeImageRelated : Scripting.DefaultCCSViewInterface
{
	// set the tab name for the related file
	public override string GetRelatedViewName(int nr)
	{
		return "Negative Image";
	}

// return one of "Image", "PDF", "XML", "HTML", "Text", "ExcelOrCSV" to know how to display
the file
	public override string GetFileType()
	{
		 return "Image";
	}

// this tells the application that this view should use the data from
GetTransformedData(string _path)
// so it must be implemented, otherwise nothing will be displayed
	public override bool IsTransformedView()
	{
		return true;
	}

// here return a MemoryStream of the displayed data (Bitmap, XML string, HTML string, plan
text, etc) in Unicode encoding
	public override MemoryStream GetTransformedData(string _path)
	{
		Bitmap bmp = new Bitmap(_path);

		for (int x = 0; x < bmp.Width; x++)
		{
			for (int y = 0; y < bmp.Height; y++)
			{
				Color oldClr = bmp.GetPixel(x, y);
				Color newClr = Color.FromArgb(255 - oldClr.R,
					255 - oldClr.G, 255  - oldClr.B);
				bmp.SetPixel(x, y, newClr);
			}
		}

		MemoryStream memoryStream = new MemoryStream();
		bmp.Save(memoryStream, System.Drawing.Imaging.ImageFormat.Jpeg);

		return memoryStream;
	}
}
```

As for the simple view, create a new class derived from `Scripting.DefaultCCSViewInterface` – the name of the class will appear in Related views list.



There are four procedures that you need to add to your custom class:

1. `public override string GetRelatedViewName(int nr)`

This procedure returns the name that will be displayed in dWValidator tab, when showing the file.

2. `public override string GetFileType()`

This procedure returns the file type. The returned string must be one of the following: Image, PDF, XML, HTML, Text or ExcelOrCSV. Based on this string, a specific View manager will be used to display the file.

3. `public override bool IsTransformedView()`

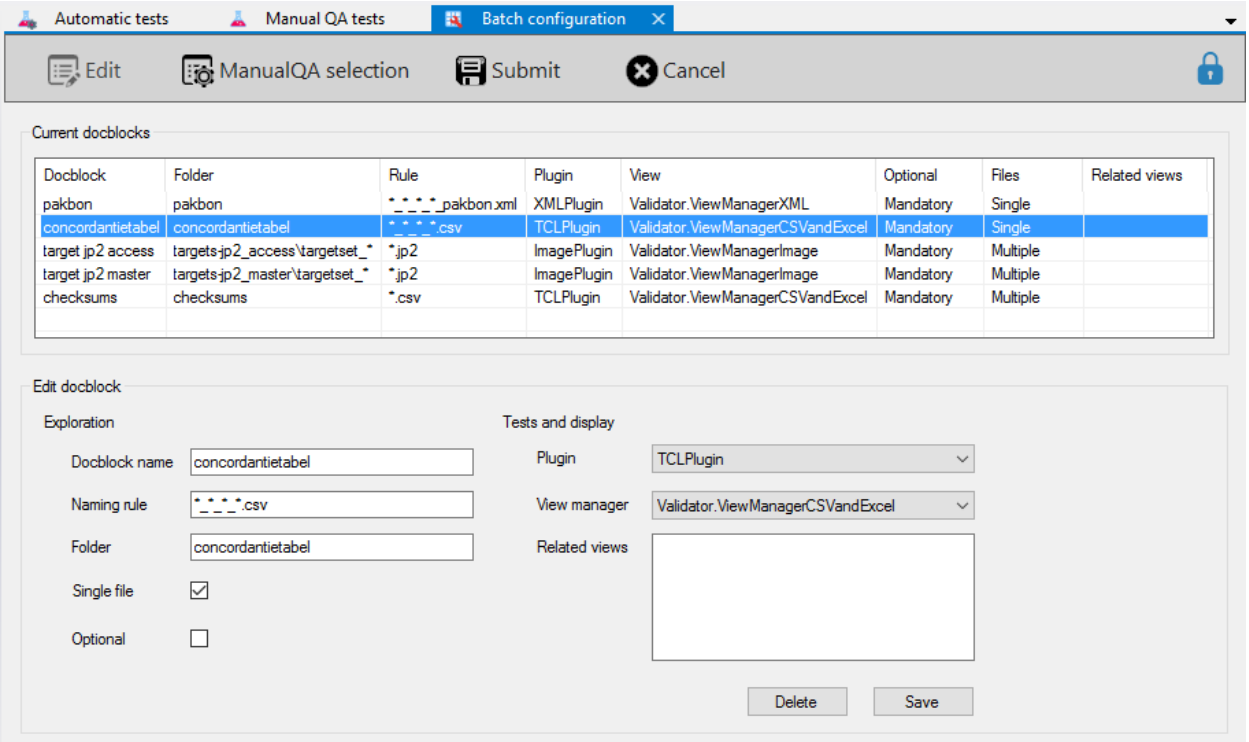
In order to have the transformation, this procedure needs to return true. Otherwise, the program will search for `GetRelatedViewName` procedure, for displaying the original file.

4. `public override MemoryStream GetTransformedData(string _path)`

This procedure returns the transformed file as a memory stream.

Batch configuration

On batch level, a docblock represents a group of similar files that are not included in any document – they are extra files that come with the batch, like checksum files, target files etc. These are not mandatory, so if your batch contains only documents, you don't need to configure anything for "Batch configuration".



The **Batch configuration** view contains:

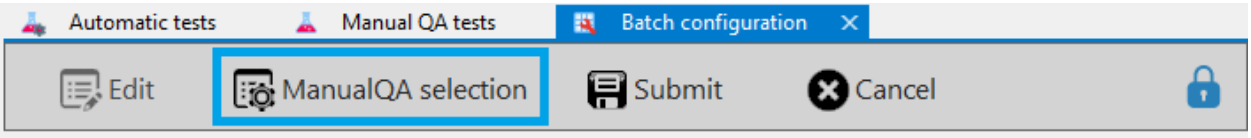
- Editing toolbar – *for more details, see Editing toolbar Help*.
- A list with the docblocks defined for the batch
- Controls used to edit the docblocks, same as for regular docblocks (f*or more details, see Add / edit docblocks Help, Define the exploration rules Help and Select the View manager Help*).

In dWValidator, all the docblocks defined on batch level will be grouped into a "virtual" document, that will be displayed first in the tree.

Define manual QA selection

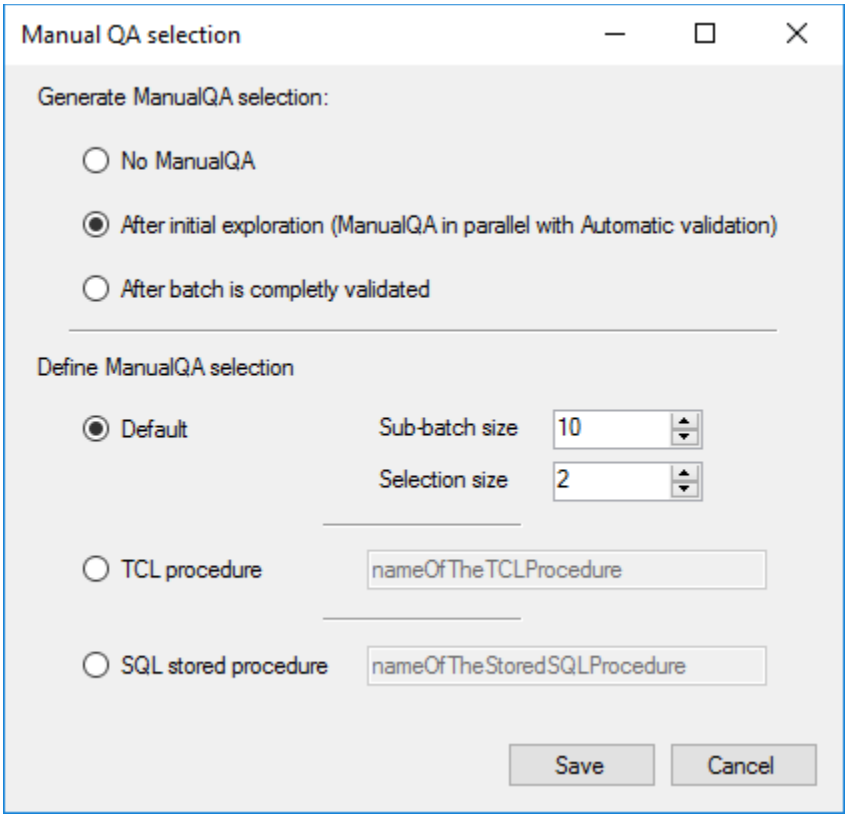
The Manual QA selection is a configuration used for marking the elements that will be part of a set of documents checked manually by the operators.

The Editing toolbar for “Batch configuration” view contains the button for defining manual QA selection.



The button is enabled once you click **Edit** button on Editing toolbar.

ManualQA selection button will open a configuration dialog:



In the top part of the dialog you can define when the ManualQA selection will be generated.

- **No ManualQA** – will not generate a ManualQA selection. This means that no batches for this project can be manually tested
- **After initial exploration** – manual QA set will be generated immediately after folders structure is explored and documents collection is created and updated into database. This is used when the selection rule is independent on automatic validation results. Manual QA can be performed in parallel with automatic tests.

- **After batch is completely validated** – manual QA will be generated only after automatic tests are performed. In this case operators will not have any selection to check until automatic tests are complete.

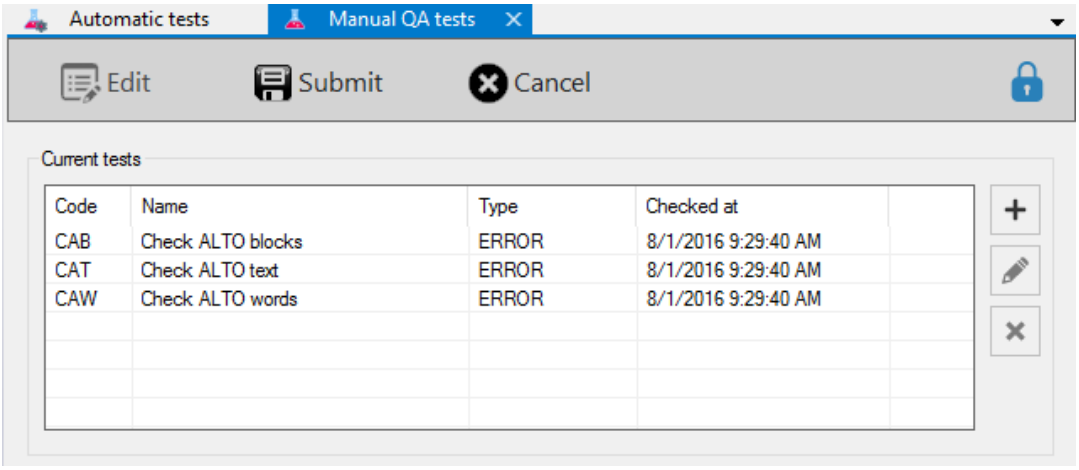
Based on the options above, the controls for defining the selection are enabled or disabled. There are 3 options to define the selection:

- **Default** – this method splits a batch of the project is sub-batches with the number of documents specified in **Sub-batch size** control. Once the sub-batches are generated, from each sub-batch a number of documents specified in **Selection size** is selected randomly and will be marked for manual QA.
- **TCL procedure** – you can write your own TCL script to generate the ManualQA selection. Once you have the script, you just need to add the procedure name
- **SQL stored procedure** – you also have the possibility to write a stored SQL procedure to generate the selection.

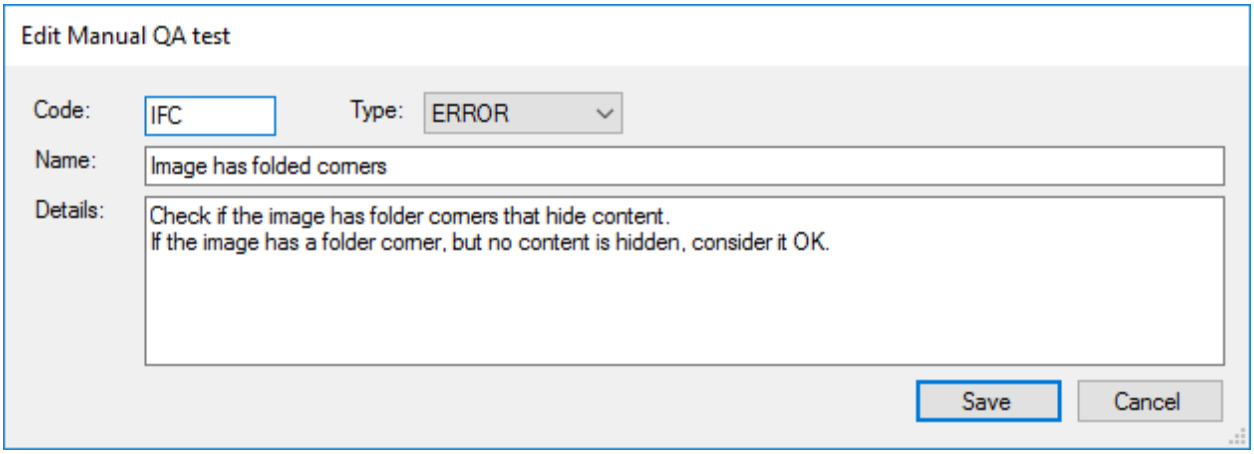
Add manual QA tests

Manual QA tests can be set for each part of a project: on batch level, document level and each docblock level. To start editing or adding manual QA tests, lock the **Manual QA** view for editing, using **Edit** button from Editing toolbar.

The view contains a list with the tests and three buttons on the right-hand side: **Add test**, **Edit test** and **Delete test**.



To add a new test, click **Add test** button  or double click an empty row from the list, which will open a separate dialog, where the details of the test can be specified.



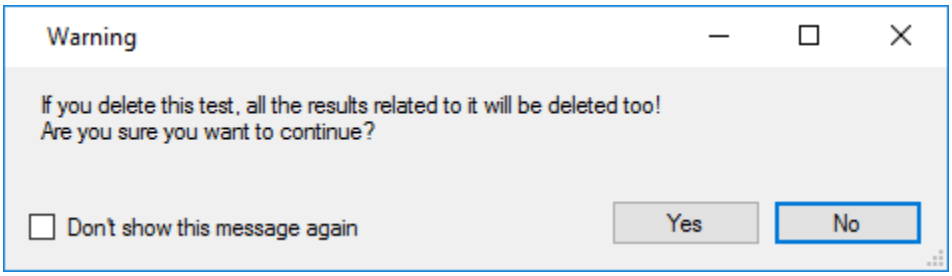
A manual QA test needs to have:

- **Test code** – a three letter code that will be used to identify the test in the database
- **Test type** – the severity of test (Warning or Error). This can be used to determine if a batch will be accepted or rejected.
- **Test name** – a short description of the test
- **Test details** – a detailed description of the test (optional).

Use **Save test** button to save the changes made for the test – this will close the dialog and add / update the test in the list. To also add the test in database, press **Submit** button on Editing toolbar. After the test is saved in database the date when the test was created is added in the list.

Use **Edit test** button  or double click on a test from the list to make changes to an existing test. The same dialog as for **Add test** button will be opened, but having all the details of the selected test filled.

You can delete a test from the list using **Delete test** button  . This will open a Warning dialog, announcing you that deleting a test implies deleting any Manual QA results related to this test.

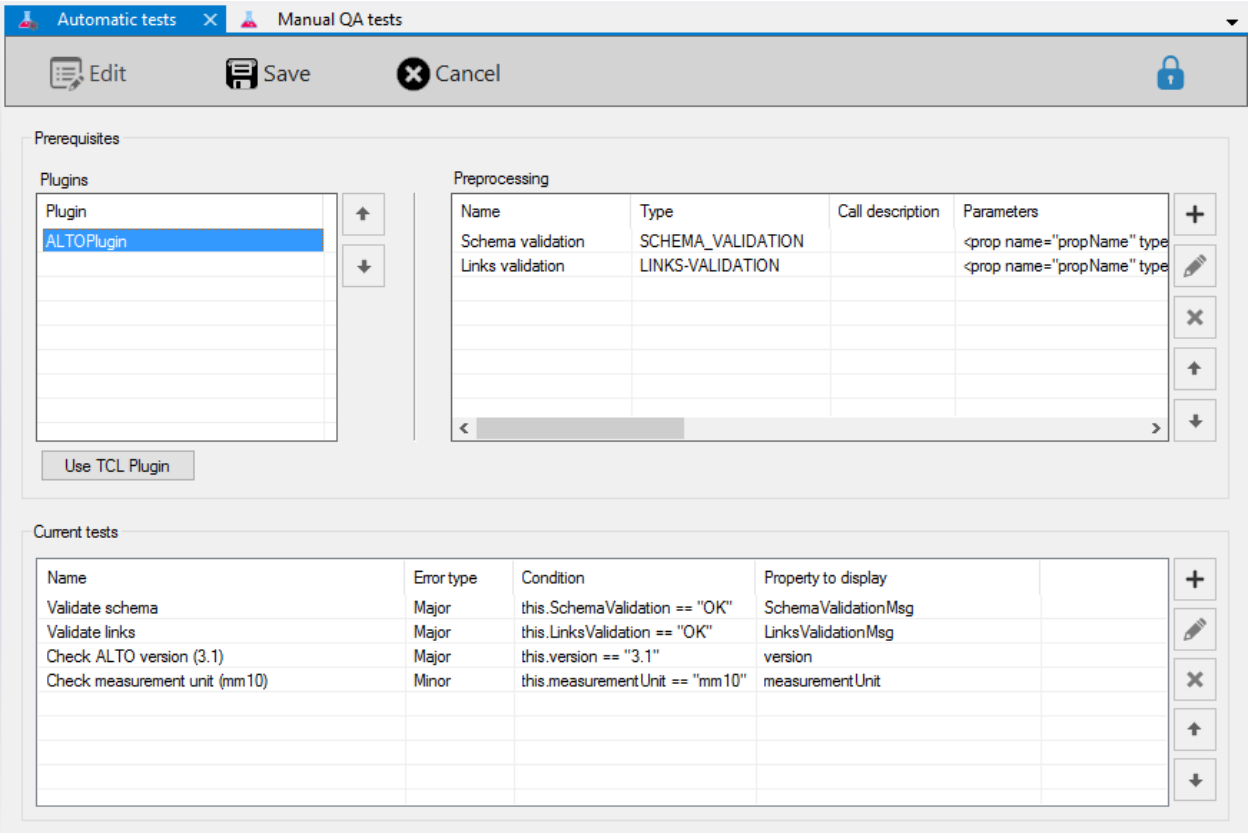


All manual QA tests defined here will appear in Tasks list in dWValidator, after Manual QA selection is generated. *For more details, see Define manual QA selection Help*.

Add automatic tests

Same as for manual QA tests, automatic tests can be set for each part of a project: on batch level, document level and each docblock level. To start editing or adding automatic tests, lock the **Automatic tests** view for editing, using **Edit** button from Editing toolbar.

The view contains two groups: **Prerequisites** and **Current tests**.



The first group contains the Plugins list and the Preprocessing list. These two will be used to gather the required data in order for tests to be run. *For more details, see Plugins Help and Preprocessings Help*.

The second group contains the list with the tests that will be performed on the files. *For more details, see Defining automatic tests Help*.

Plugins

A plugin is a software component that extracts properties of a file, which will be used in an automated test condition.

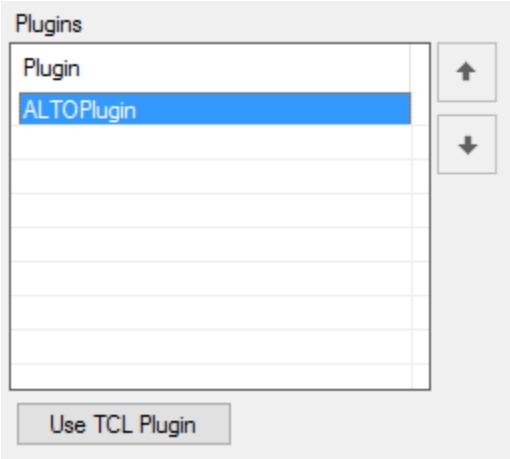
The plugins available are:

- ImagePlugin – used to extract properties of all supported image types
- ALTOPlugin – used to extract properties specific to ALTO format files
- METSPlugin – used to extract properties specific to METS format files

- PDFPlugin – used to extract properties of PDF files
- XMLPlugin – used to extract properties of HTML and XML files
- EPUBPlugin – used to extract properties of ePUB files
- TCLPlugin – used for any file type.

For more details, see Plugins, Preprocessings and Properties Help.

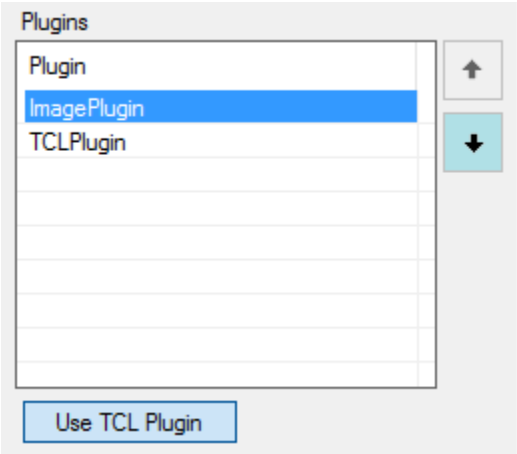
The plugin that you have selected when defining the docblocks will appear in the Plugins list.

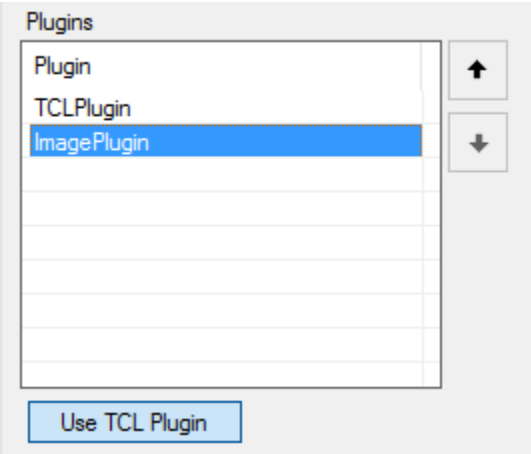


You can combine the selected plugin with the TCLPlugin, to extract properties that the standard plugin cannot produce. The TCLPlugin is able to run custom defined TCL procedures that will export a set of properties, which can be tested by conditions. You can add the TCLPlugin with **Use TCL Plugin** button.

When using a plugin combination, it's important to define their order in the list. For example, if you use TCLPlugin to define a property that depends on another property generated by ImagePlugin, the ImagePlugin must be first in list, otherwise the TCLPlugin will not be able to generate its property, leading to failed or crashed tests.

To set the correct order of the plugins in the list, use the arrow buttons from the right-hand side of the Plugins list:



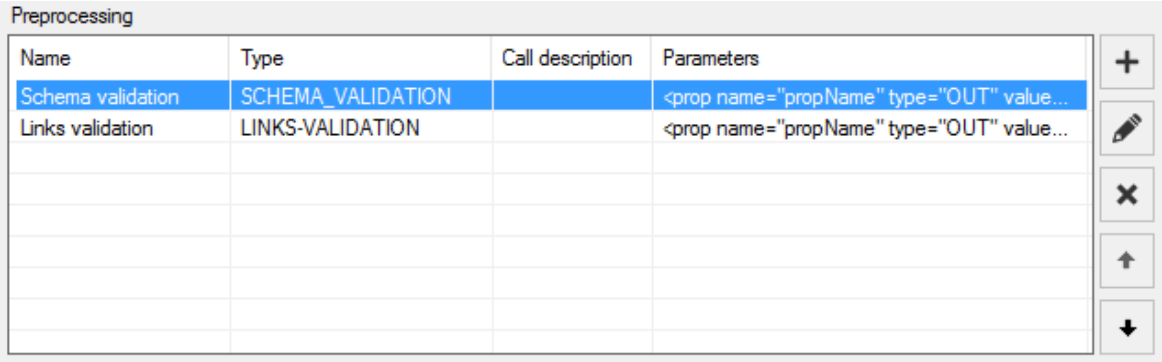


To remove the TCLPlugin from the list, click again the **Use TCL Plugin**.

WARNING! Removing the TCLPlugin from the list will also delete the preprocessing and tests defined for it.

Preprocessings

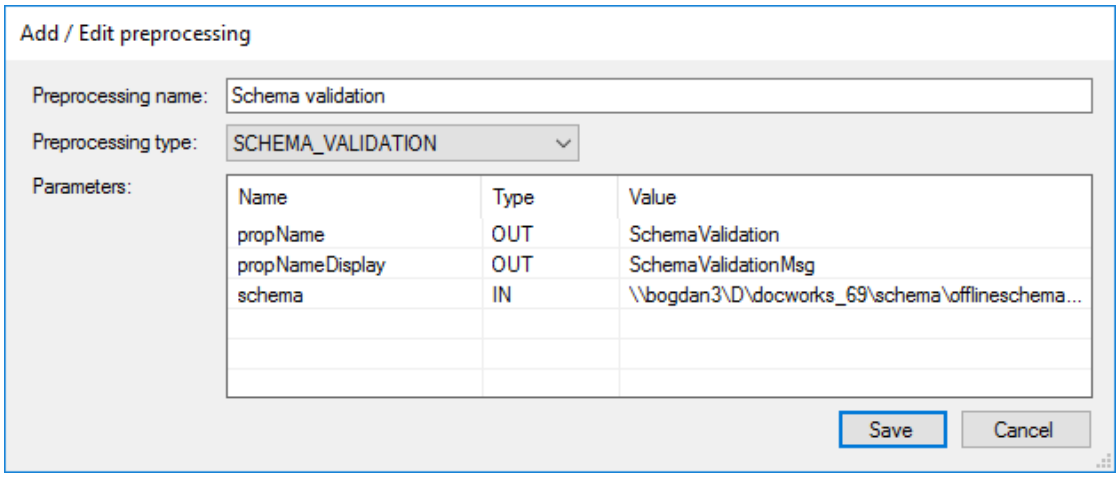
Some properties of files need additional processing to be extracted.



Default set of properties is enhanced by additional preprocessing defined per plugin. The properties generated after a Preprocessing step are not included into standard set of properties mainly because of two reasons:

- either the process of extracting those properties is time consuming and these will be computed only if they are explicitly requested
- or additional information is needed (for example to validate a schema using XMLPlugin you need to add as input the schema used for validation).

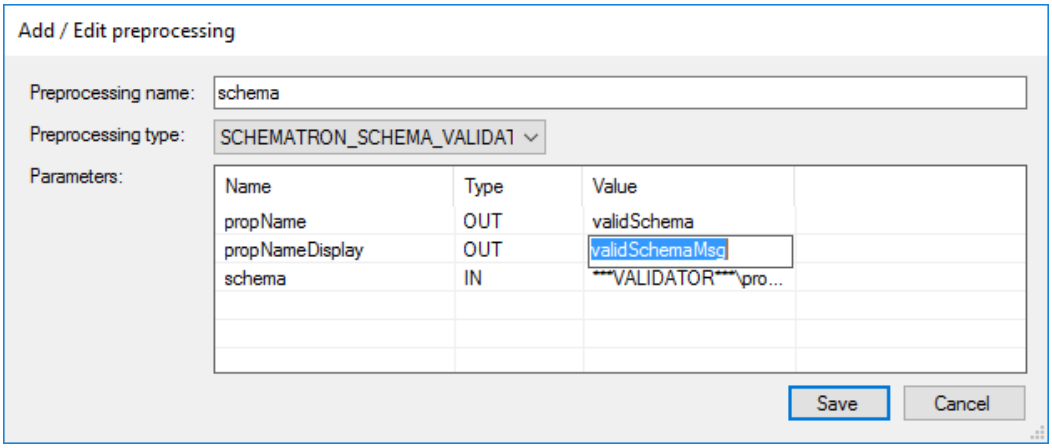
To add a new preprocessing, select the desired plugin from Plugins list and then press **Add preprocessing** button  from the right-hand side of the preprocessing list (or double click on the list). This will open a dialog where the preprocessing details can be filled.



Add a **Preprocessing name**, that will be used only for display purpose.

The **Preprocessing type** dropdown is automatically filled based on the selected plugin. Each plugin has a list of possible preprocessing. *For more details, see Plugins, Preprocessings and Properties Help.*

Once the Preprocessing type is selected, the value for each **Parameter** needs to be added. To do this, double click on the row of the parameter, under the **Value** column.



The values allowed depend on the type of the parameters:

- **OUT** parameter – accepts only alphanumeric characters and underscore "_"
- **IN** parameter – accepts all characters
- **INDEX** parameter – accepts only alphanumeric characters and underscore "_" for one parameter and comma "," as separator.

For more details, see Plugins, Preprocessings and Properties Help.

Press **Save** button to add the preprocessing to the list and **Submit** from **Editing toolbar** to add it to database.

Use **Edit preprocessing** button  to make changes to an existing preprocessing. The same dialog as for **Add preprocessing** button will be opened, but having all the details on the selected preprocessing filled. You can also double click on an item from the list to edit it.

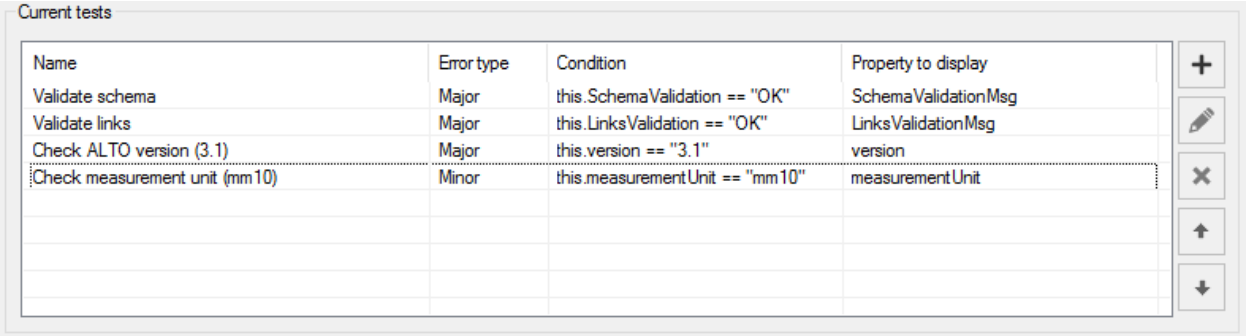
As in the case of plugins, preprocessings may require a specific order in which to be performed, in case properties from one preprocessing are used in another one. Use the arrow buttons  on the right-hand side of the list to change the order.

Use **Remove preprocessing** button  to delete a preprocessing from the list.

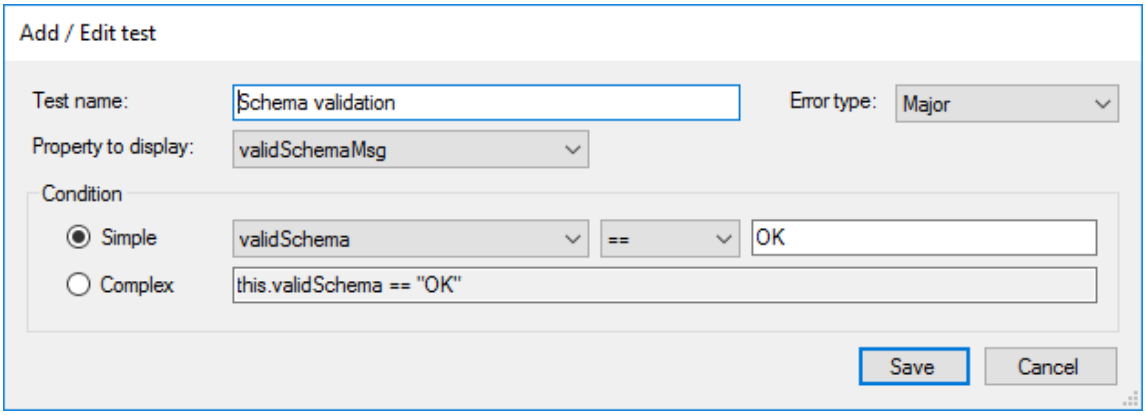
WARNING! Deleting a preprocessing from the list will also delete any test that was defined using the properties extracted with the help of that preprocessing.

Defining automatic tests

The tests list is in the second group of the **Automatic tests** view. The tests defined here will be run on all files on the docblock for each document in the batch. You can also define tests that can be run on documents or on batch by selecting the level from Projects tree.

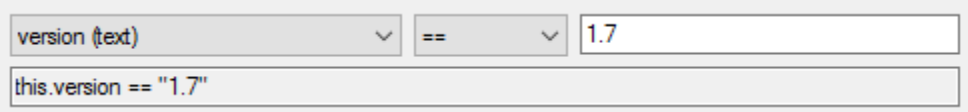


To add a new test, use **Add test** button  from the right-hand side of the list with tests or double click on an empty row from list. This will open a separate dialog where the test details can be filled:



- **Test name** – the name of the test. This is how the test will appear in the validation report
- **Error type** – the severity of the test (Minor or Major). Just as for manual QA tests, this can be used to determine if the batch will be accepted or rejected.
- **Property to display** – based on the plugin selected, the properties dropdown will be prefilled with the default properties and the ones generated by added preprocessing. *For more details, see Plugins, Preprocessing and Properties Help.* This is used to display the validation result in validation report. Usually there are defined pairs *propertyName* and *propertyNameMsg* and the second one is recommended to be used as display property, because it provides additional information is special cases (For example, when testing that PDF version is 1.4, the test can fail either because version is not 1.4, or the PDF is not valid and version cannot be extracted. Using *versionMsg* will display in validation report the reason why it failed). In regular cases these two properties have identical values if test is performed properly. If *propertyNameMsg* is missing for one of the properties, use *propertyName* for display.

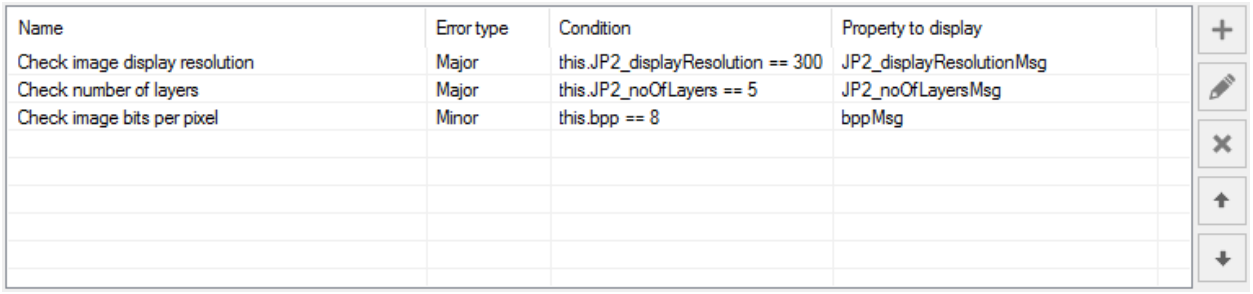
- **Condition –** the statement that will be evaluated in the validation process. It can be a simple condition, where you can select the value and the operator from dropdowns, or a complex condition, where you have to type the entire statement.
  a. The **simple condition** is composed of:
     - The **tested property** that can be selected from a list automatically filled, based on plugin and preprocessings. Use *propertyName* to create a condition – for example, if you need to test that PDF version is 1.7, use *version* from the list to generate "version == 1.7"
     - An **operator** from the available list of operators:
       - **!=** (inequality)
       - **<** (less than)
       - **<=** (less than or equal)
       - **==** (equal)
       - **>** (greater than)
       - **>=** (greater than or equal)
       - **Contains**
       - **in**
     - The **value** that the property will be compared to. For standard properties of a plugin, there are validation rules – the value added should correspond with the value type, that is displayed in brackets).
     - And how the generated condition will look like.



  b. For **complex condition**, you can evaluate multiple properties in the same test, using **&&** (AND) and **||** (OR) logical operators to combine the conditions.

You can even compare properties against other properties. For example, you want to make sure that all images are portrait – in this case, the complex condition will look like this:

The tests list has arrow buttons to arrange the tests in the desired order. Tests order has no impact on the validation process, it's only used to define the sequence in which the tests will be displayed in validation reports.



Name	Error type	Condition	Property to display
Check image display resolution	Major	this.JP2_displayResolution == 300	JP2_displayResolutionMsg
Check number of layers	Major	this.JP2_noOfLayers == 5	JP2_noOfLayersMsg
Check image bits per pixel	Minor	this.bpp == 8	bppMsg

Use **Edit test** button  to edit selected test and **Remove test** button  to delete it.

Click **Submit** button from **Editing toolbar** to save the tests in database.

Plugins, Preprocessing and Properties

Each plugin has a number of default properties that can be used in tests conditions, but also properties that can be generated using preprocessing.

ImagePlugin

Default properties

Property	Property to display	Type
bpp	bppMsg	Numeric
compression	CompressionMsg	Text
fileSize	fileSizeMsg	Numeric
height	Height	Numeric
ICCProfile	ICCProfileMsg	Text
name	Name	Text
path	Path	Text
resolution	ResolutionMsg	Numeric
width	Width	Numeric
XMPMetadata	XMPMetadataMsg	Text
JP2_captureResolution	JP2_captureResolutionMsg	Numeric
JP2_codeBlockHeight	JP2_codeBlockHeightMsg	Numeric
JP2_codeBlockWidth	JP2_codeBlockWidthMsg	Numeric
JP2_colourTransformation	JP2_colourTransformationMsg	Text
JP2_comment	JP2_commentMsg	Text
JP2_decompositionLevels	JP2_decompositionLevelsMsg	Numeric
JP2_displayResolution	JP2_displayResolutionMsg	Numeric
JP2_endOfPacket	JP2_endOfPacketMsg	Text
JP2_errorResilience	JP2_errorResilienceMsg	Text
JP2_ICCProfileMethod	JP2_ICCProfileMethodMsg	Text
JP2_noOfLayers	JP2_noOfLayersMsg	Numeric
JP2_precInctHeight	JP2_precInctHeightMsg	Text
JP2_precInctWidth	JP2_precInctWidthMsg	Text
JP2_progressionOrder	JP2_progressionOrderMsg	Text
JP2_segmentationSymbols	JP2_segmentationSymbolsMsg	Text
JP2_startOfPacket	JP2_startOfPacketMsg	Text
JP2_tileHight	JP2_tileHeightMsg	Numeric
JP2_tileWidth	JP2_tileWidthMsg	Numeric
JP2_waveletTransformation	JP2_waveletTransformationMsg	Text

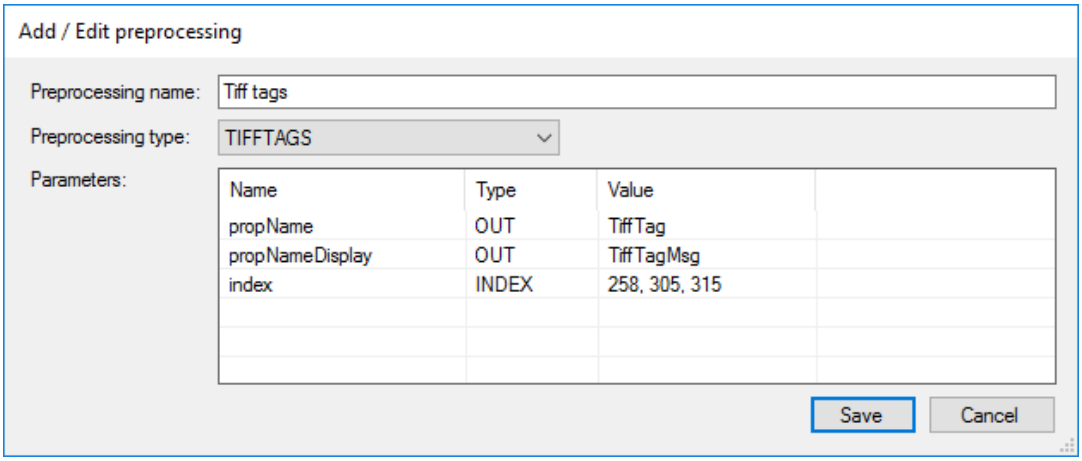
Preprocessing

Only one preprocessing is available for ImagePlugin: TIFFTAGS. You can use it to generate properties for the tags you need to test for your tiff files (*for details about tiff tags, visit* http://www.awaresystems.be/imaging/tiff/tifftags.html).

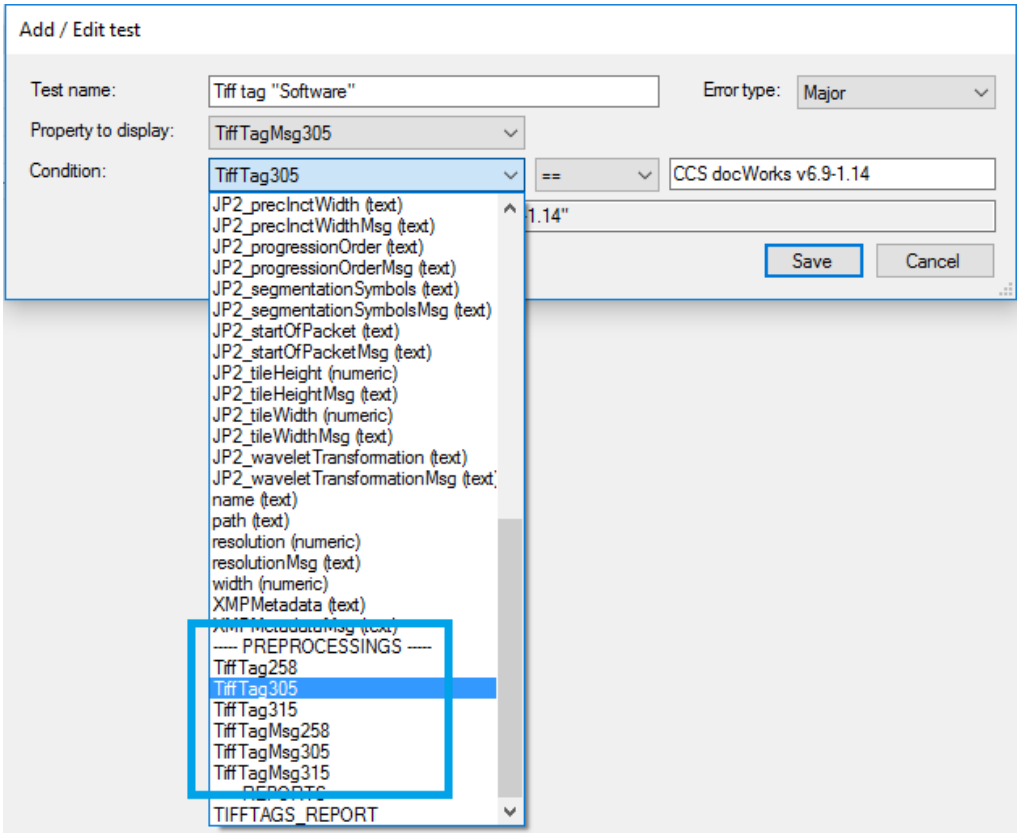
TIFFTAGS needs three parameters in order to generate the properties:

- **propName** – used as the property in test condition.
- **propNameDisplay** – used to display the validation result in the report
- **index** – a list of tiff tags indexes. You can add here the tags you are interested in, separated by comma ",". For each value added here, there will be generated a property and a property to display, having the names as combination between propName / propNameDisplay and an index.

For example:



Setting the preprocessing like in the screenshot, with value *258, 305, 315* for index will generate three properties and three properties to display. These will be found in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under "PREPROCESSINGS" group:



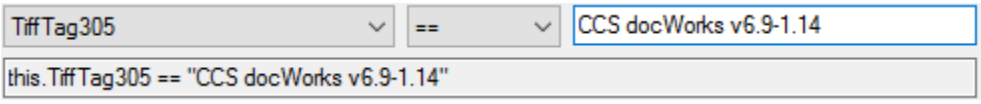
Condition

The properties generated with the help of preprocessing need to be compared to a value, but because the property is generated, its type is unknown. All the generated properties will be treated as *string* and the quotation marks will be added automatically.

For example:

- tag 258 (BitsPerSample) is a numeric tag, but the condition will look like this:

- tag 305 (Software) is a text tag, but the condition will look like this:



For default properties, check the type of the property displayed in brackets. You will notice that the conditions will be formatted based on the type:

- for text (strings), the quotation marks will be automatically added
- for numeric, no quotation mark will be added, but there will be validation rules to allow only numbers
- for bool, only *true* or *false* are allowed as values.

ALTOPlugin

Default properties

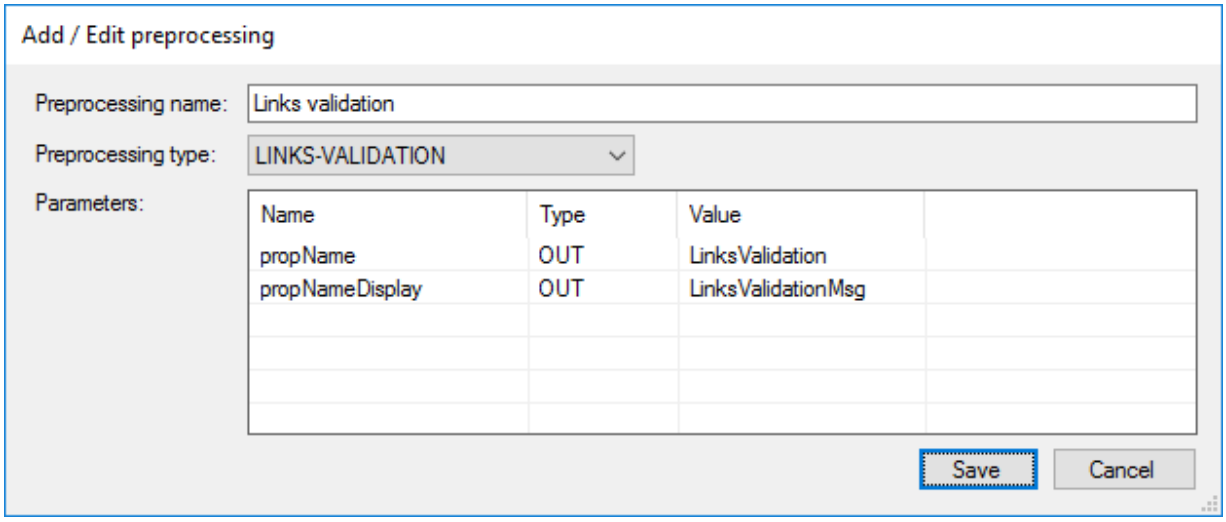
Property	Property to display	Type
errorMsg	errorMsg	Text
fileSize	fileSizeMsg	Numeric
height	height	Numeric
measurementUnit	measurementUnit	Text
name	name	Text
pageId	pageId	Text
path	path	Text
position	position	Text
schema	schema	Text
srcImageName	srcImageName	Text
valid	valid	Bool
version	version	Text
width	width	Numeric
xLink	xLink	Text
xsi	xsi	Text

Preprocessing

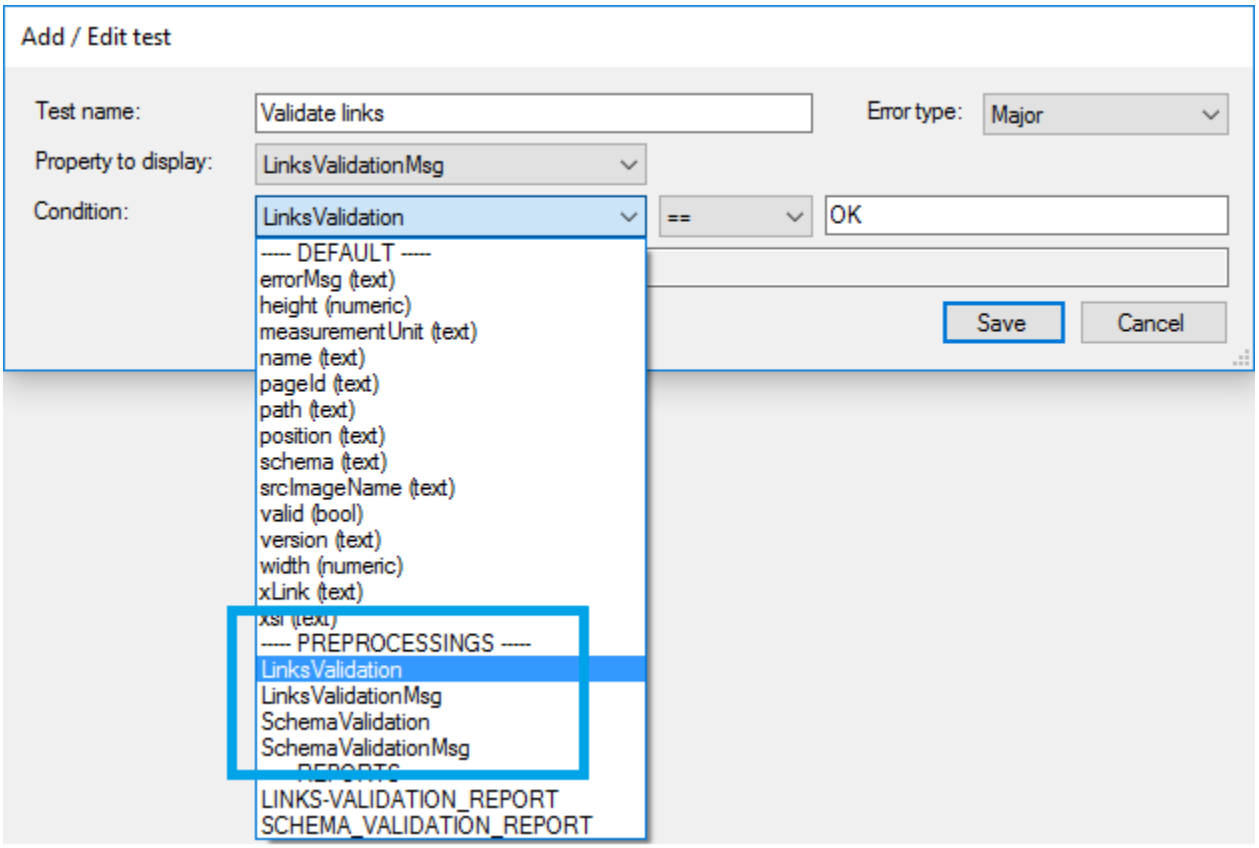
For ALTOPlugin, there are four preprocessings available: LINKS-VALIDATION, SCHEMA_VALIDATION, SCHEMATRON_SCHEMA_VALIDATION and ENCODING_CHECK.

LINKS-VALIDATION can be used to validate the links from the ALTO file. This preprocessing has two parameters:

- **propName** – the name of the property used in test condition
- **propNameDisplay** – used to display the validation result in the report.

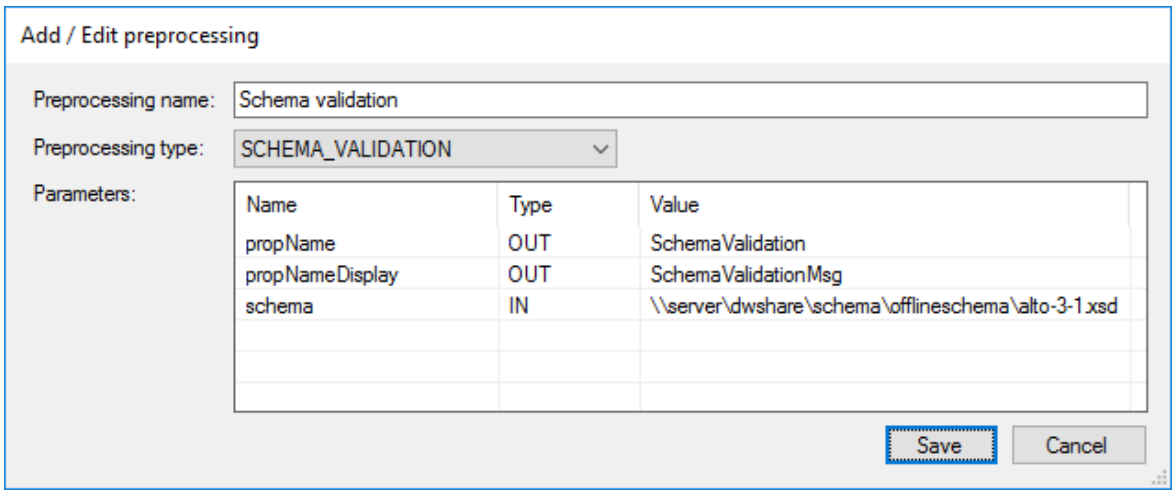


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

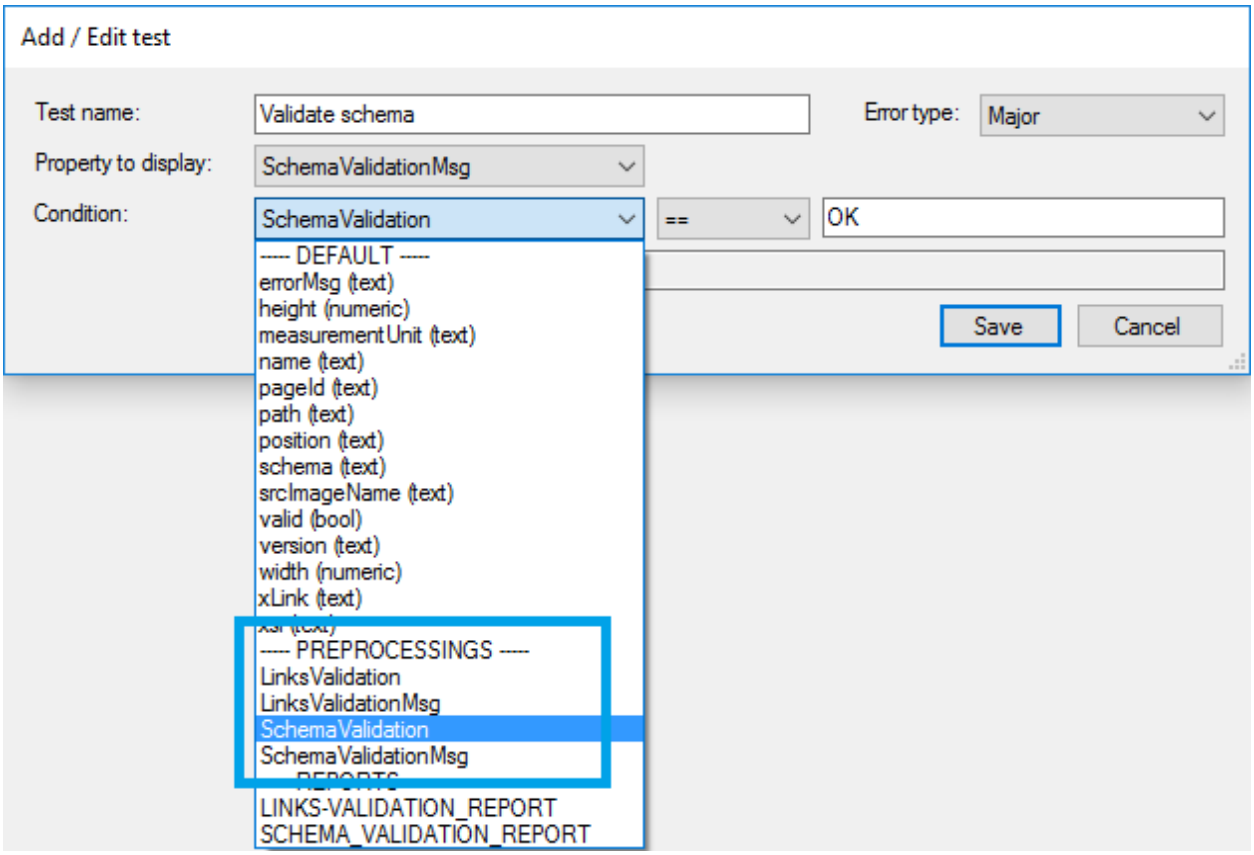


SCHEMA_VALIDATION can be used to validate the ALTO file against a standard schema. The parameters for this preprocessing are:

- **propName** – used as the property in the test condition
- **propNameDisplay** – used to display the result of the validation in the report
- **schema** – the .xsd file that the ALTO file will be compared with. The entire path, together with the file name are needed for this parameter.

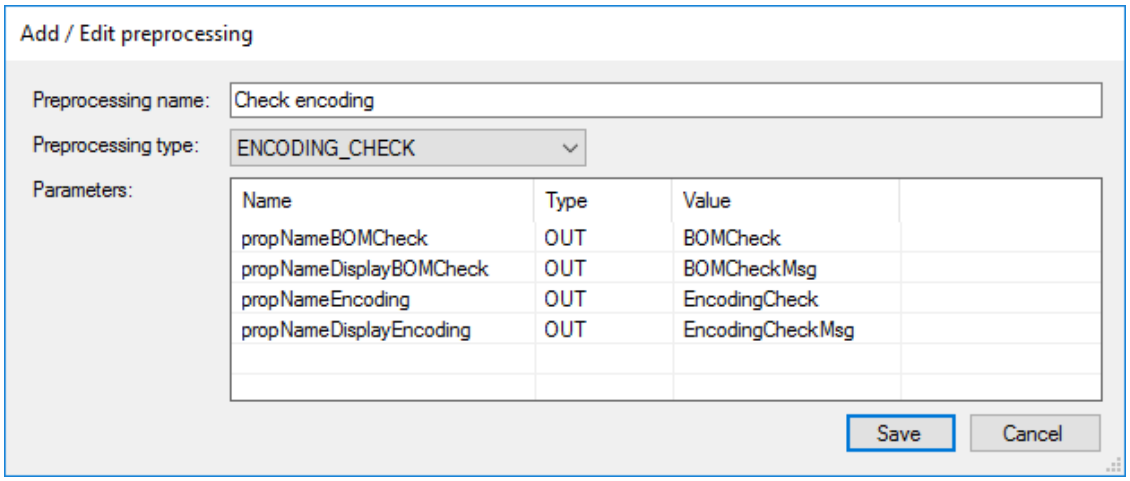


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

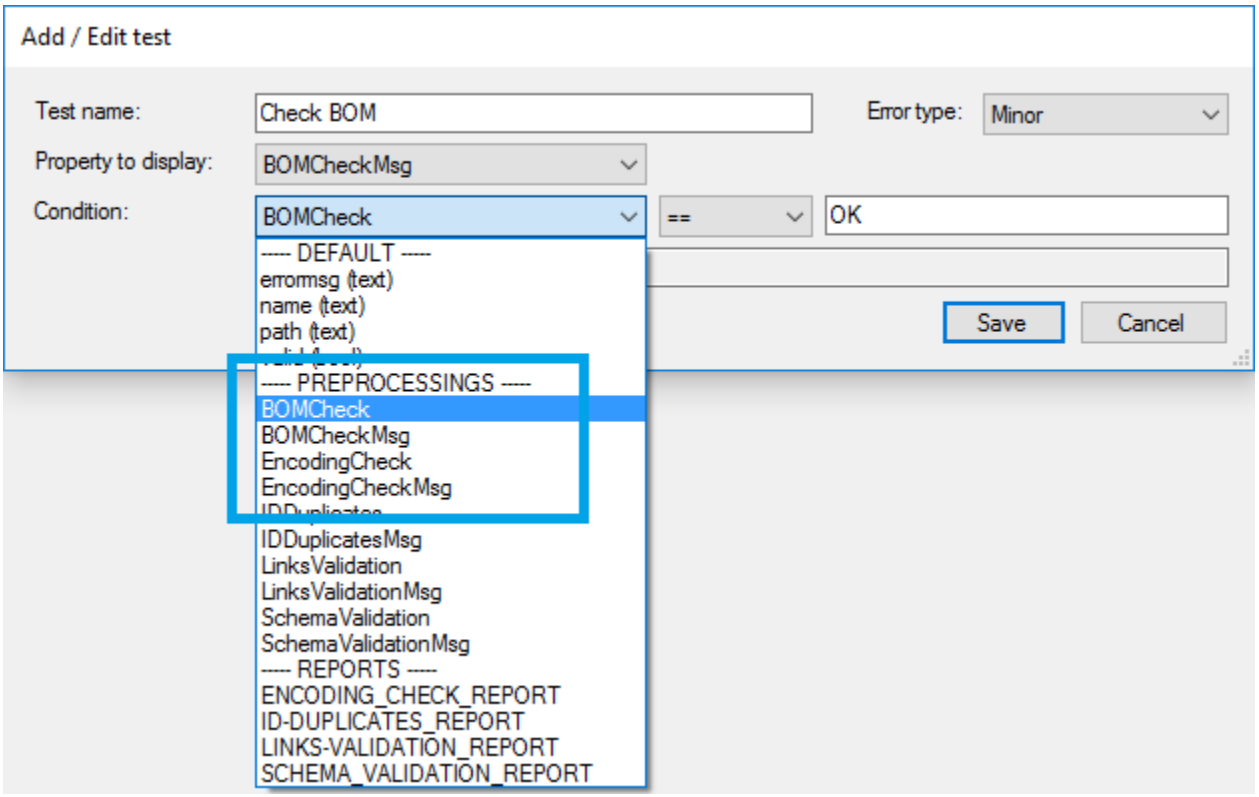


ENCODING_CHECK can be used to check the file encoding and BOM property. The parameters for this preprocessing are:

- **propNameBOMCheck** – used as the property in the test condition for BOM
- **propNameDisplayBOMCheck** – used to display the validation result for BOM in the report
- **propNameEncoding** – used as the property in the test condition for encoding
- **propNameDisplayEncoding –** used to display the validation result for encoding in the report

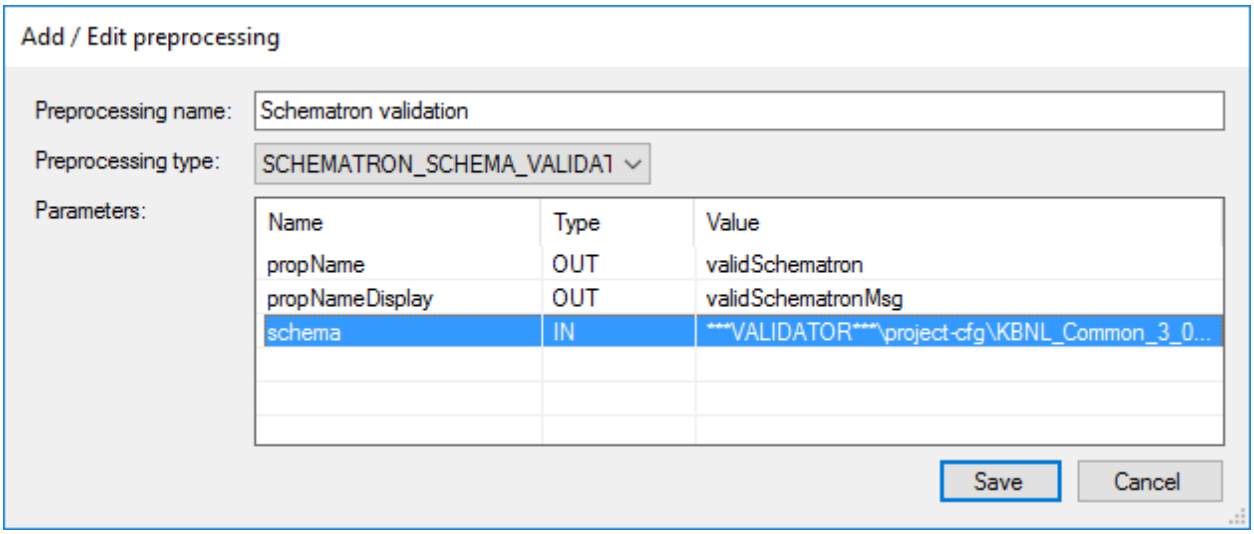


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

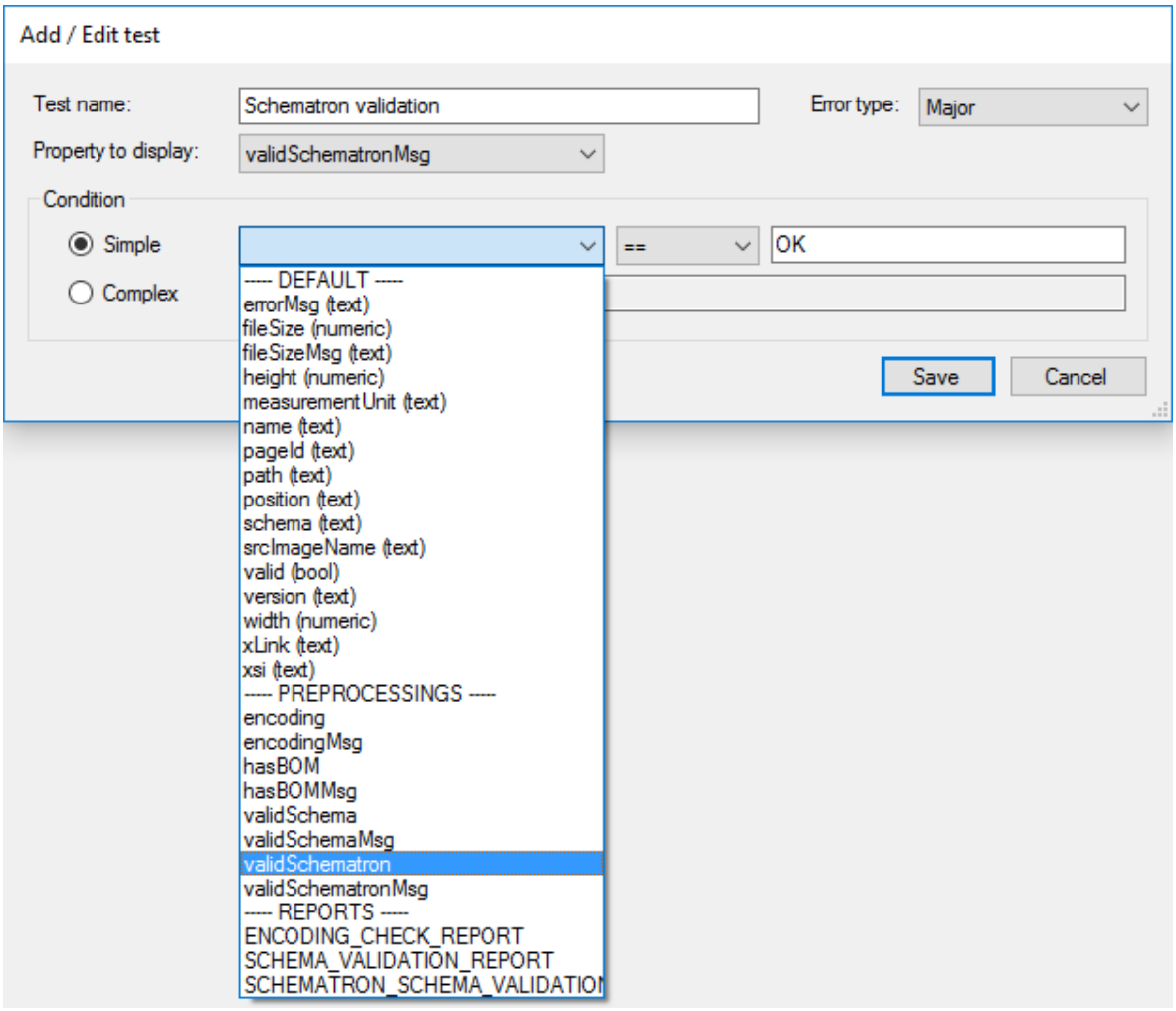


SCHEMATRON_SCHEMA_VALIDATION validates the file against a .sch schema. The parameters for this preprocessing are:

- **propName** – used as the property in the test condition
- **propNameDisplay** – used to display the result of the validation in the report
- **schema** – the .sch file that the ALTO file will be compared with. The entire path, together with the file name are needed for this parameter

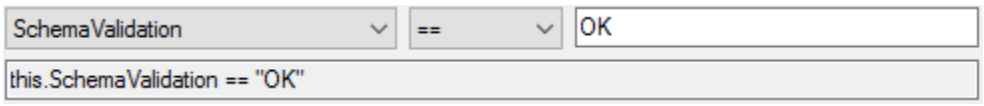


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:



Condition

All properties generated by the preprocessings for ALTOPlugin need to be compared against *OK* in the condition. The tests will return *OK* if the validation was done successfully and other strings if it failed (which will be displayed in the report by *propNameDisplay*). So the condition must look like this:



WARNING! The validation process is case-sensitive, so adding values like *Ok* or *ok* will lead to failed tests.

For default properties, check the type of the property displayed in brackets. You will notice that the conditions will be formatted based on the type:

- for text (strings), the quotation marks will be automatically added

- for numeric, no quotation mark will be added, but there will be validation rules to allow only numbers
- for bool, only *true* or *false* are allowed as values.

METSPlugin

Default properties

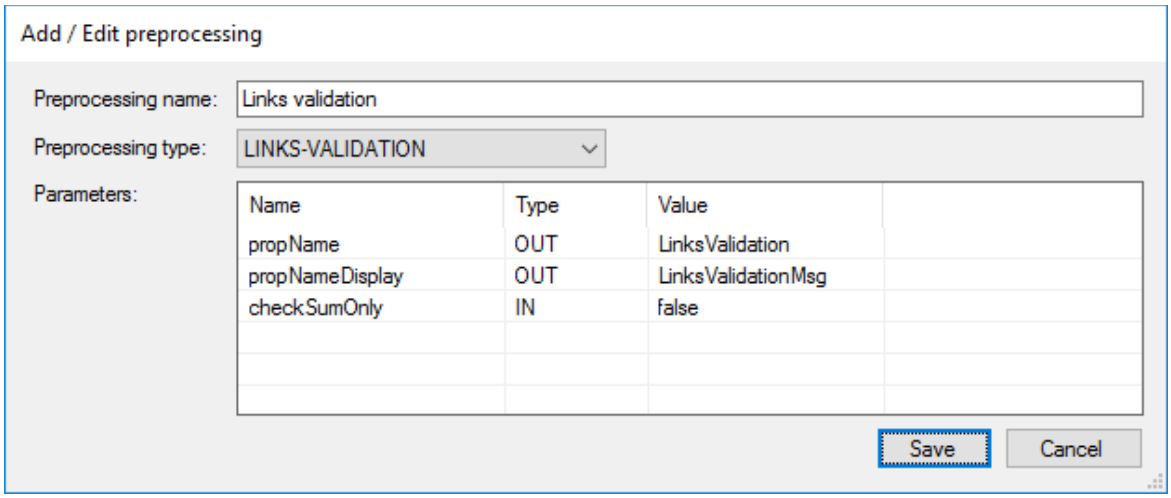
Property	Property to display	Type
errorMsg	errorMsg	Text
fileSize	fileSizeMsg	Numeric
name	name	Text
path	path	Text
valid	valid	Bool

Preprocessing

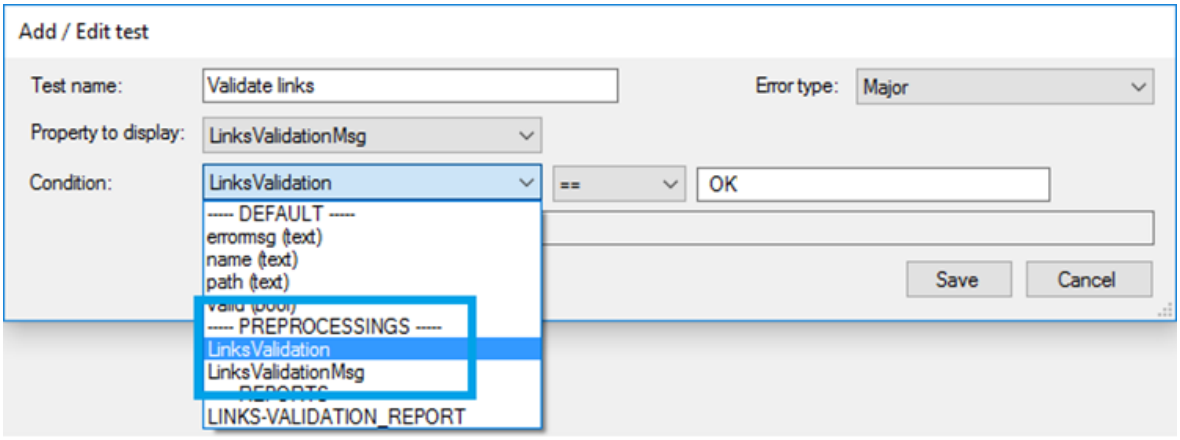
For METSPlugin, there are six preprocessings available: LINKS-VALIDATION, SCHEMATRON_SCHEMA_VALIDATION, SCHEMA_VALIDATION, ID-DUPLICATES, ESCAPE-SEQ and ENCODING_CHECK.

LINKS-VALIDATION can be used to validate the links from the METS file. This preprocessing has three parameters:

- **propName** – the name of the property used in test condition
- **propNameDisplay** – used to display the validation result in the report.
- **checkSumOnly** – is a Boolean parameter, used to specify if only the checksum will be validated, or all the links of the METS file.
    - If is set to “false” or “0”, all the links of the METS file will be validated
    - If is set to “true” or “1”, only the checksum will be validated

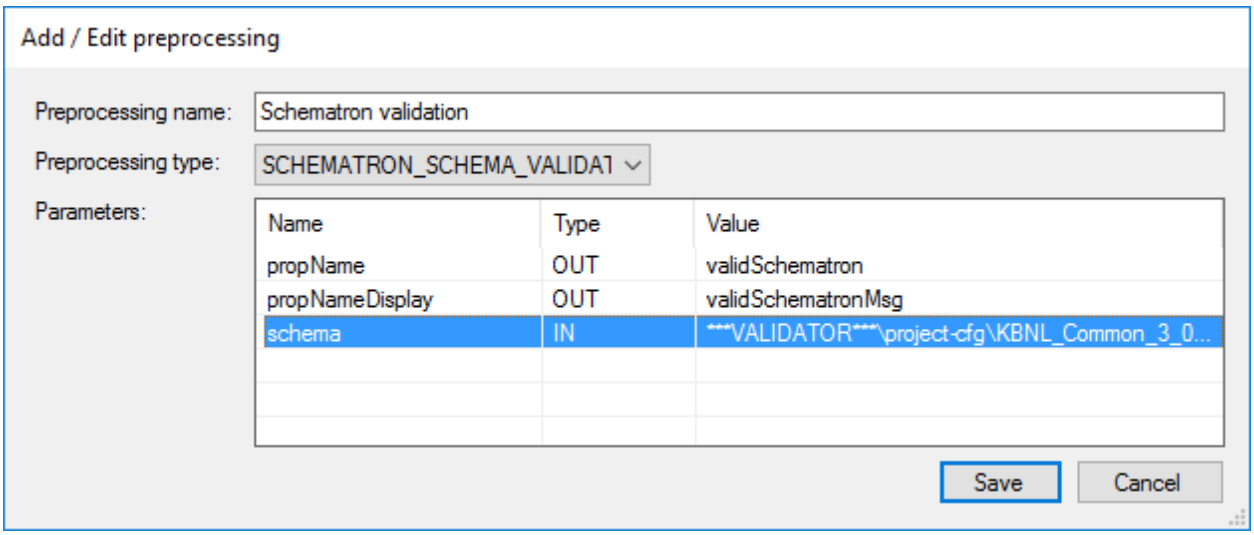


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

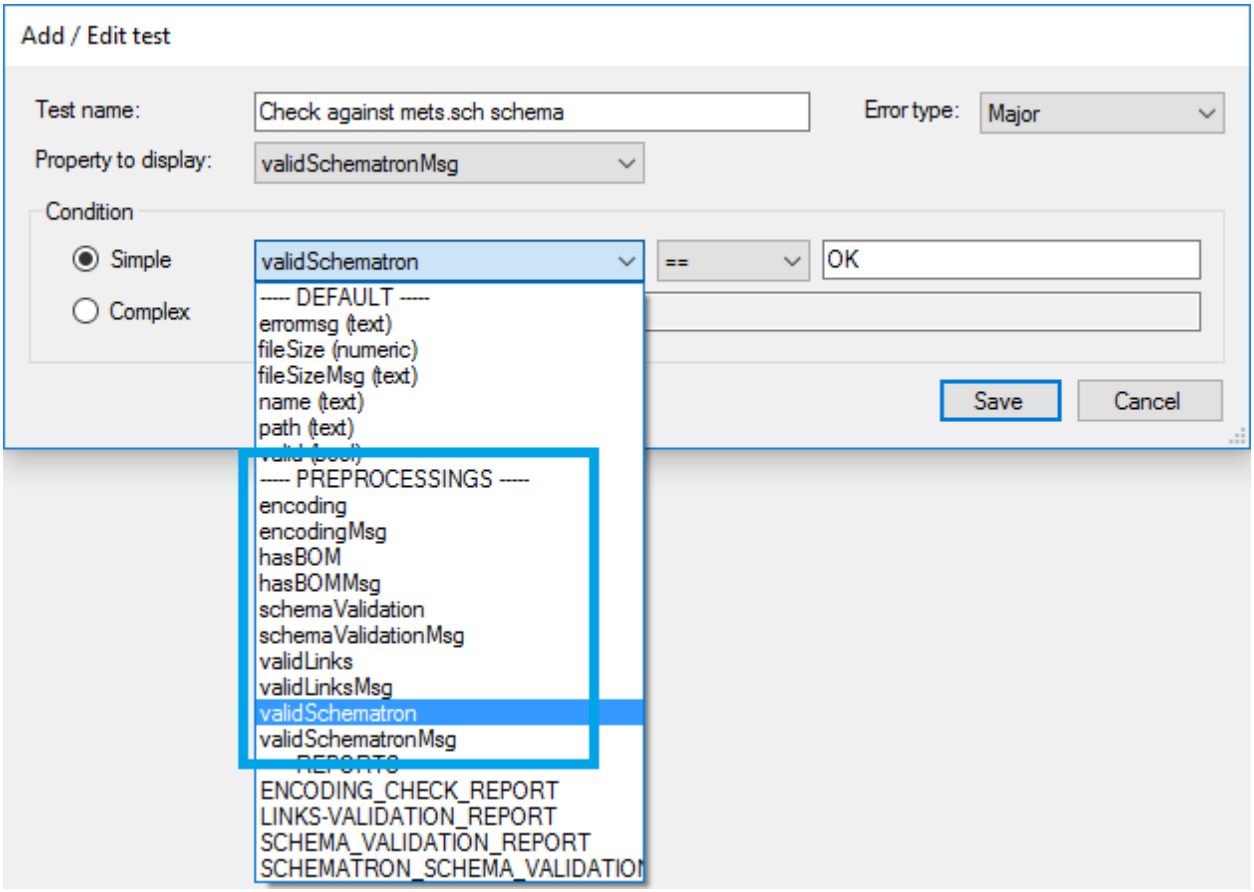


SCHEMATRON_SCHEMA_VALIDATION validates the file against a .sch schema. The parameters for this preprocessing are:

- **propName** – used as the property in the test condition
- **propNameDisplay** – used to display the result of the validation in the report
- **schema** – the .sch file that the METS file will be compared with. The entire path, together with the file name are needed for this parameter

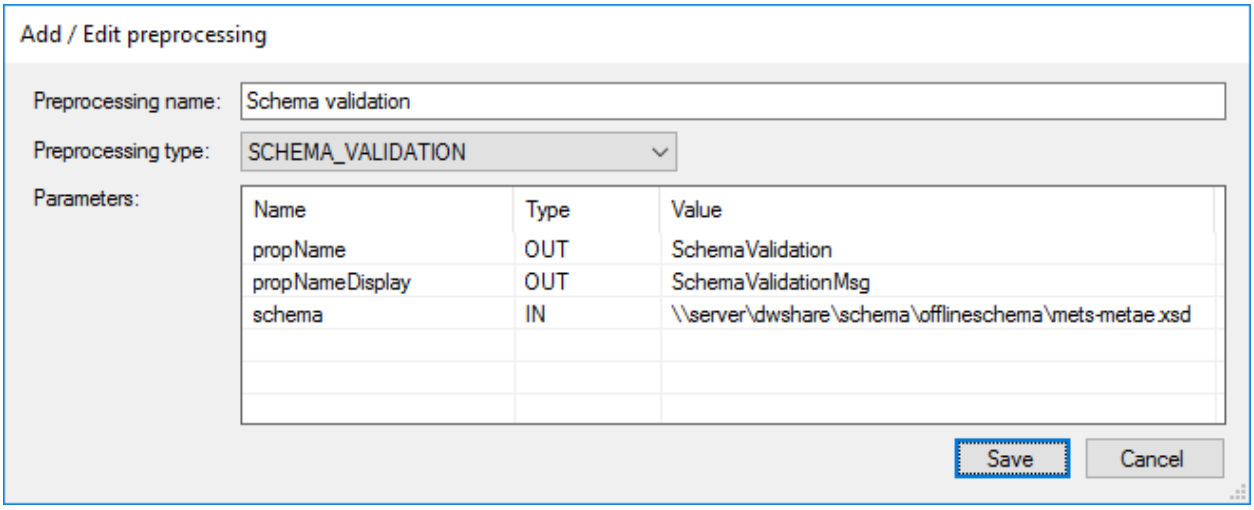


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

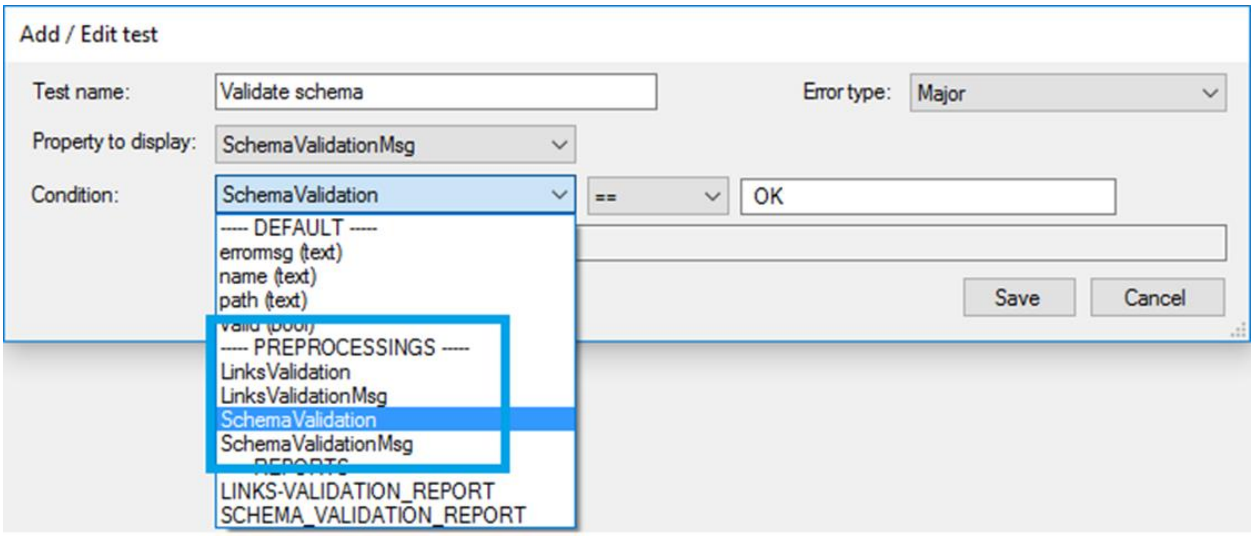


SCHEMA_VALIDATION can be used to validate the METS file against a standard schema. The parameters for this preprocessing are:

- **propName** – used as the property in the test condition
- **propNameDisplay** – used to display the result of the validation in the report
- **schema** – the .xsd file that the METS file will be compared with. The entire path, together with the file name are needed for this parameter.

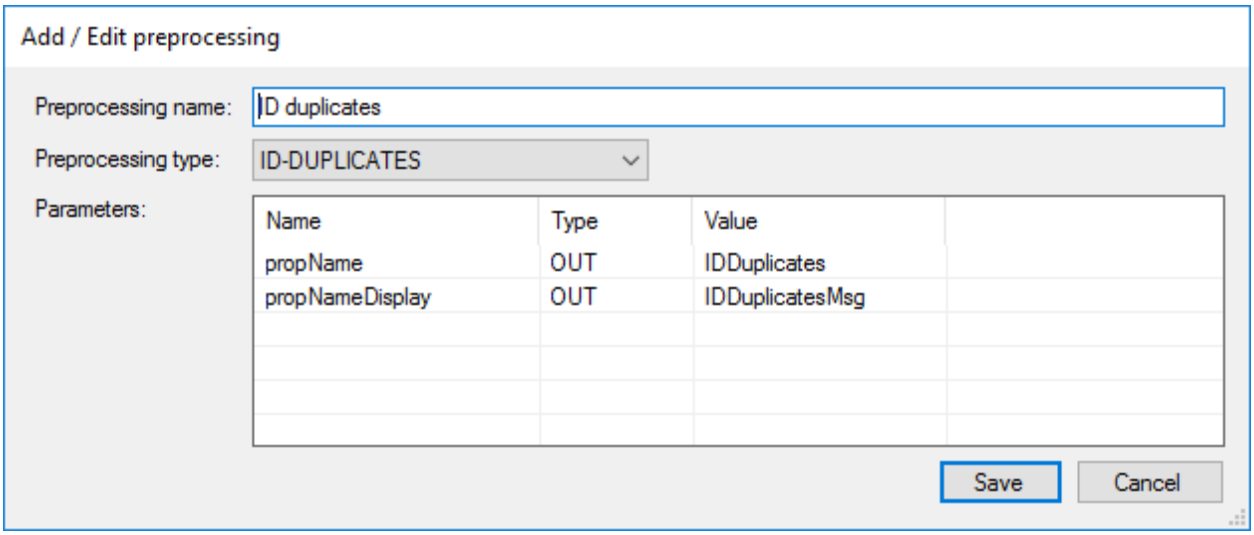


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

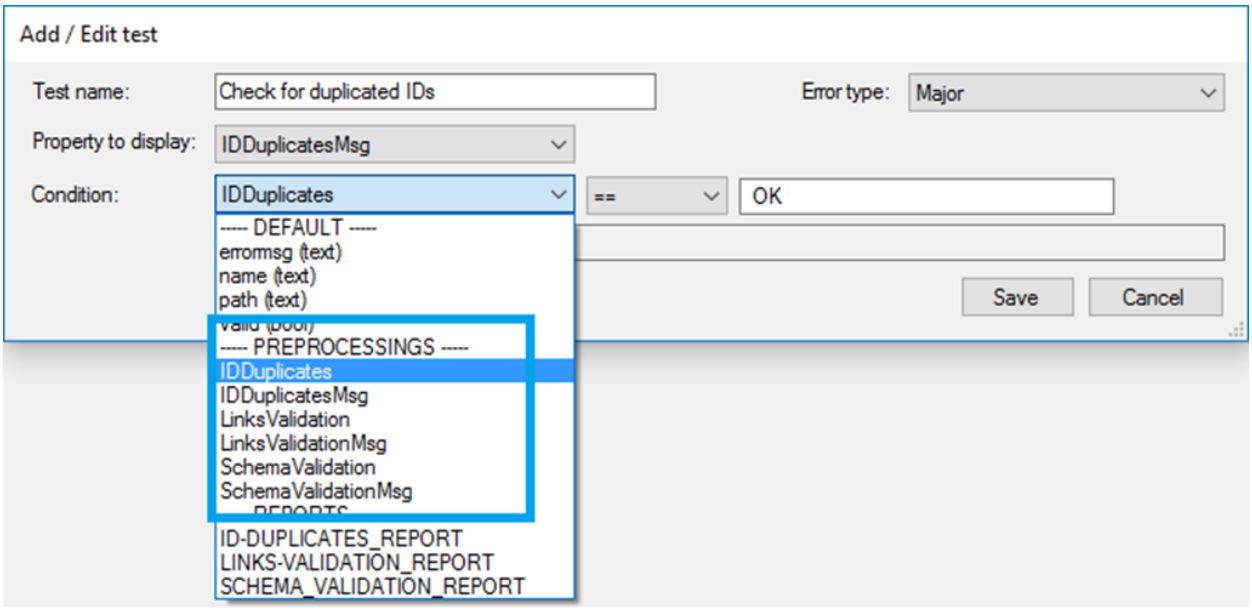


ID-DUPLICATES is used to check the METS file for duplicated IDs. The parameters necessary for this preprocessing are:

- **propName** – used as the property in the test condition
- **propNameDisplay** – used to display the result of the validation in the report



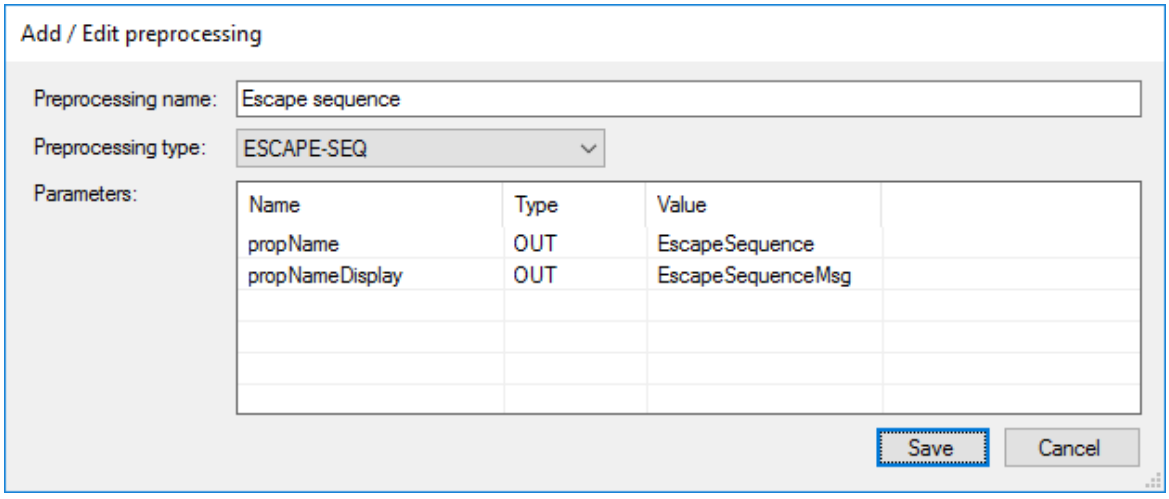
The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:



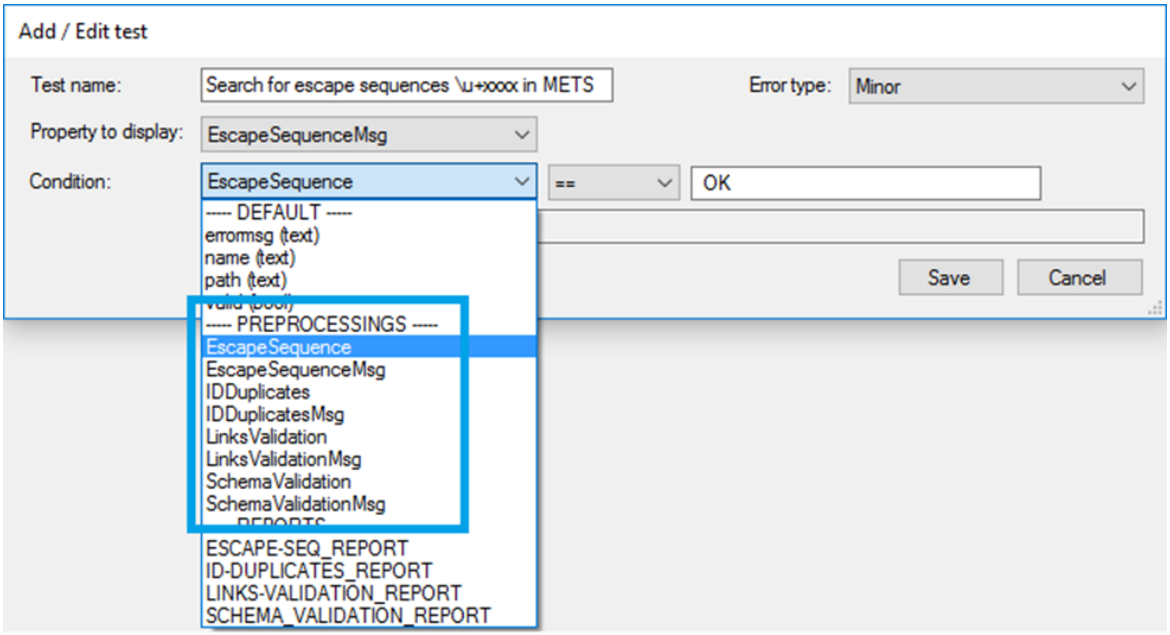
ESCAPE-SEQ can be used to search for special combination of characters, consisting of a backslash (\) followed by a letter and a combination of digits (ex: "\u+xxxx", where "x" is a digit). If no escape sequence is found within the file, the test will be valid, otherwise will fail.

The parameters necessary for this preprocessing are:

- **propName** – used as the property in the test condition
- **propNameDisplay** – used to display the result of the validation in the report

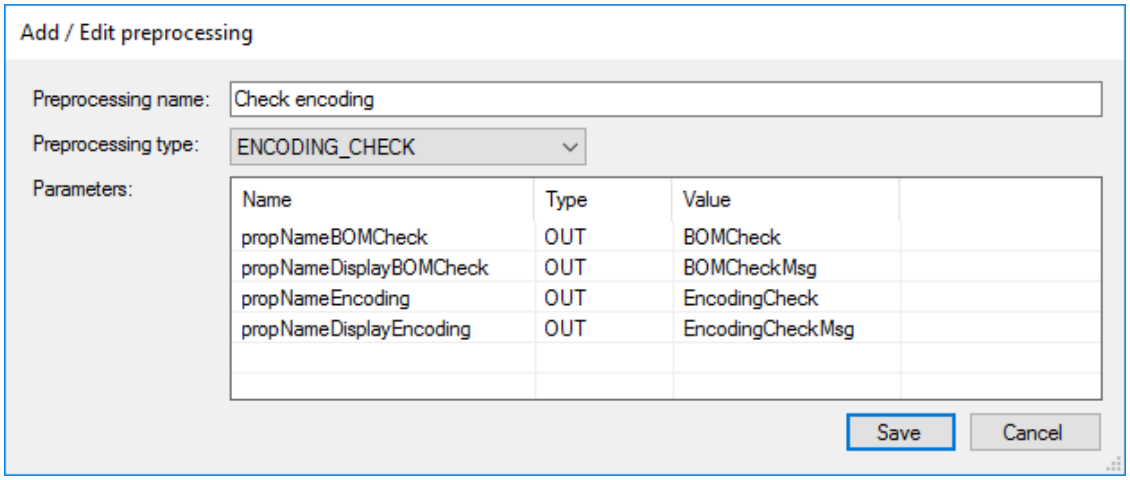


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under "PREPROCESSINGS" group:

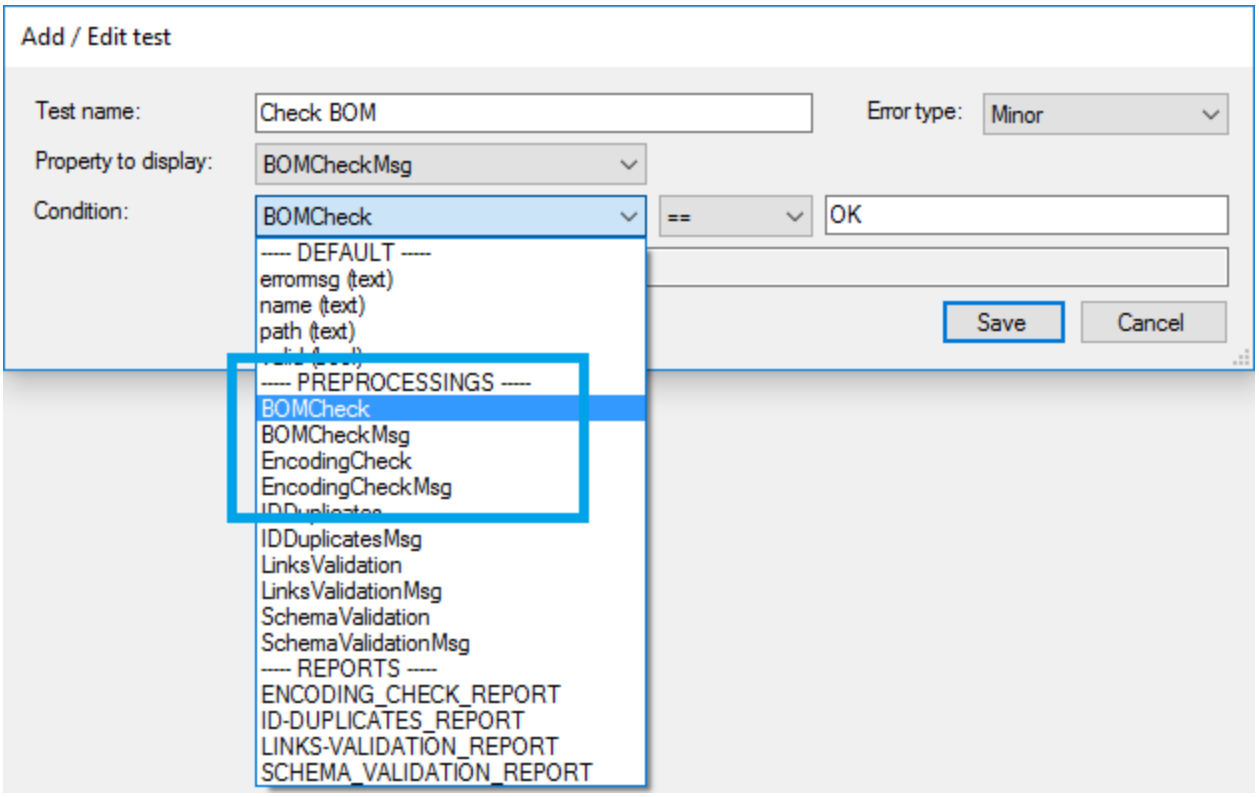


ENCODING_CHECK can be used to check the file encoding and BOM property. The parameters for this preprocessing are:

- **propNameBOMCheck** – used as the property in the test condition for BOM
- **propNameDisplayBOMCheck** – used to display the validation result for BOM in the report
- **propNameEncoding** – used as the property in the test condition for encoding
- **propNameDisplayEncoding –** used to display the validation result for encoding in the report

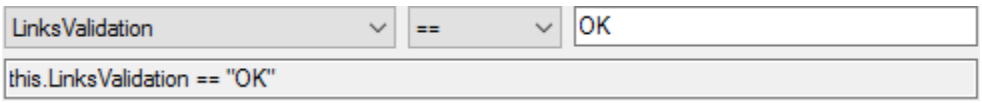


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:



Condition

All properties generated by the preprocessings for METSPlugin need to be compared against *OK* in the condition. The tests will return *OK* if the validation was done successfully and other strings if it failed (which will be displayed in the report by *propNameDisplay*). So, the condition must look like this:



WARNING! The validation process is case-sensitive, so adding values like *Ok* or *ok* will lead to failed tests.

For default properties, check the type of the property displayed in brackets. You will notice that the conditions will be formatted based on the type:

- for text (strings), the quotation marks will be automatically added
- for numeric, no quotation mark will be added, but there will be validation rules to allow only numbers
- for bool, only *true* or *false* are allowed as values.

PDFPlugin

Default properties

Property	Property to display	Type
author	authorMsg	Text
creationDate	creationDateMsg	Text
creator	creatorMsg	Text
errorMsg	errorMsg	Text
fileSize	FileSizeMsg	Numeric
jhoveXml	jhoveXmlMsg	XML Text
modDate	modDateMsg	Text
name	name	Text
pageLayout	pageLayoutMsg	Text
pageMode	pageModeMsg	Text
pageCount	pageCountMsg	Numeric
path	path	Text
producer	producerMsg	Text
profile	profileMsg	Text
status	statusMsg	Text
title	titleMsg	Text
valid	valid	Boolean
version	versionMsg	Text
XMP	XMPMsg	Text

Preprocessing

PDFPlugin has no available preprocessing.

Condition

For default properties, check the type of the property displayed in brackets. You will notice that the conditions will be formatted based on the type:

- for text (strings), the quotation marks will be automatically added
- for numeric, no quotation mark will be added, but there will be validation rules to allow only numbers
- for bool, only *true* or *false* are allowed as values.

XMLPlugin

Default properties

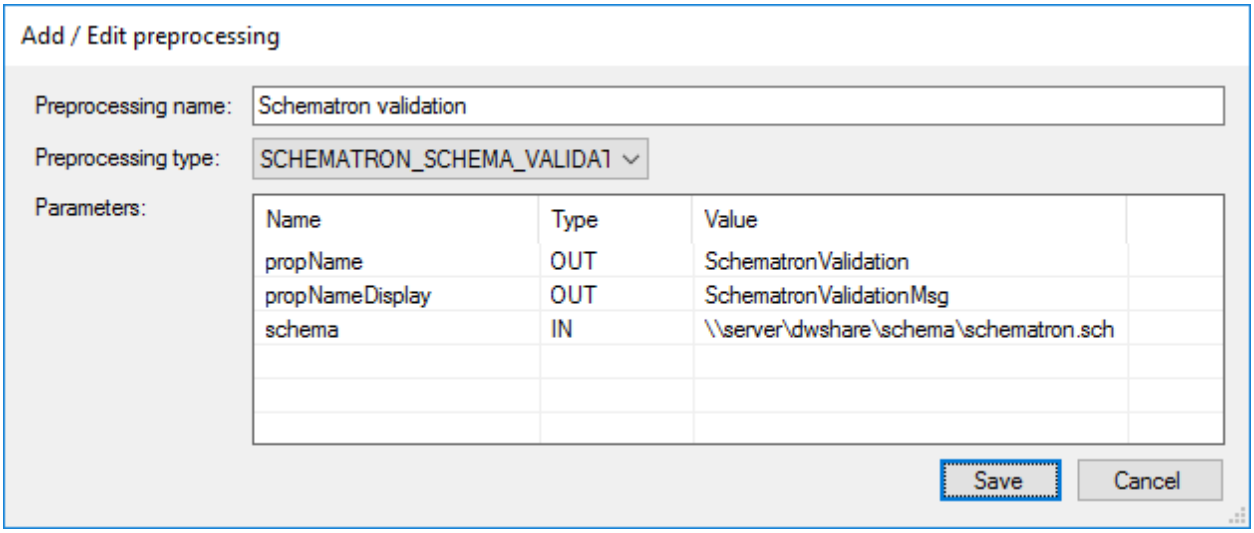
Property	Property to display	Type
characterCount	characterCountMsg	Numeric
creationDate	creationDateMsg	text
creator	creatorMsg	Text
errorMsg	errorMsg	Text
fileSize	fileSizeMsg	Numeric
fontFile	fontFileMsg	Text
fontName	fontNameMsg	Text
identifier	identifierMsg	Text
language	languageMsg	Text
name	name	Text
path	path	Text
publisher	publisherMsg	Text
references	referencesMsg	Text
status	statusMsg	Bool
title	titleMsg	Text

Preprocessing

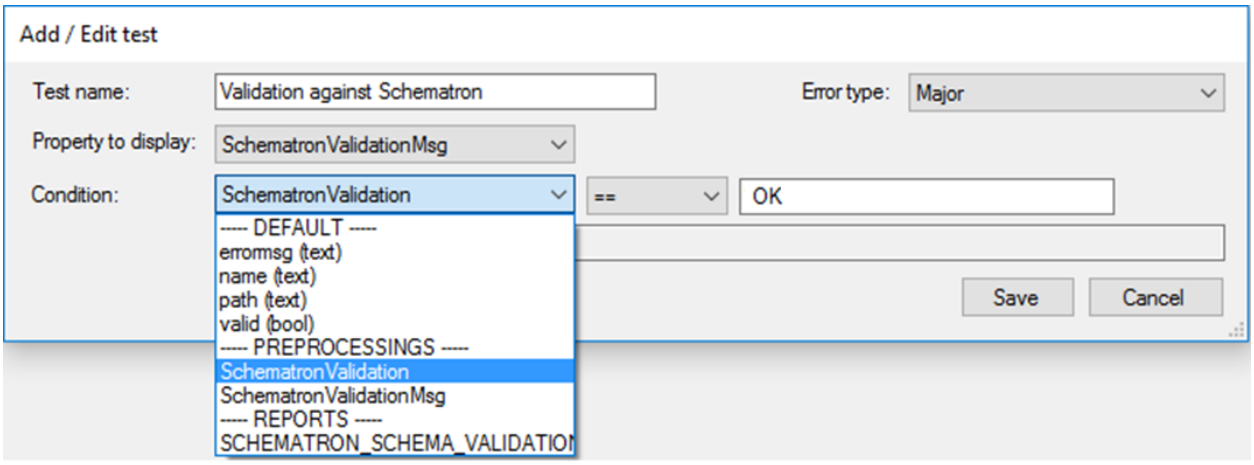
For XMLPlugin, there are four preprocessings available: SCHEMATRON_SCHEMA_VALIDATION, XSD_SCHEMA_VALIDATION, DTD_SCHEMA_VALIDATION and ENCODING_CHECK.

SCHEMATRON_SCHEMA_VALIDATION can be used to validate the XML file against a Schematron file. It uses three parameters:

- **propName** – the name of the property used in test condition
- **propNameDisplay** – used to display the validation result in the report
- **schema** - the .sch file that the XML file will be compared with. The entire path, together with the file name are needed for this parameter.

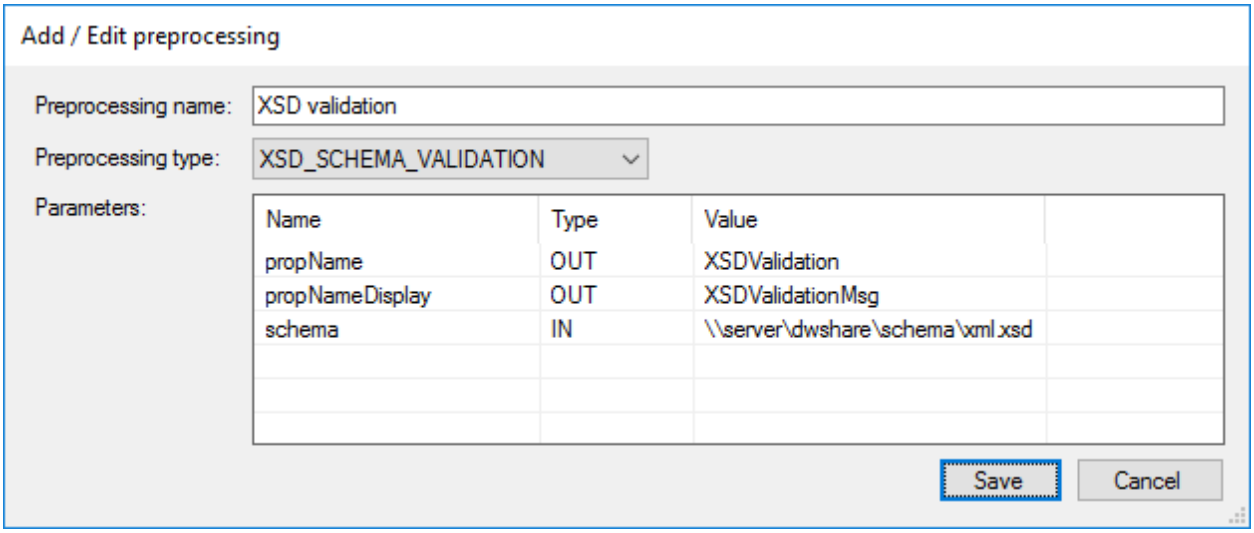


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

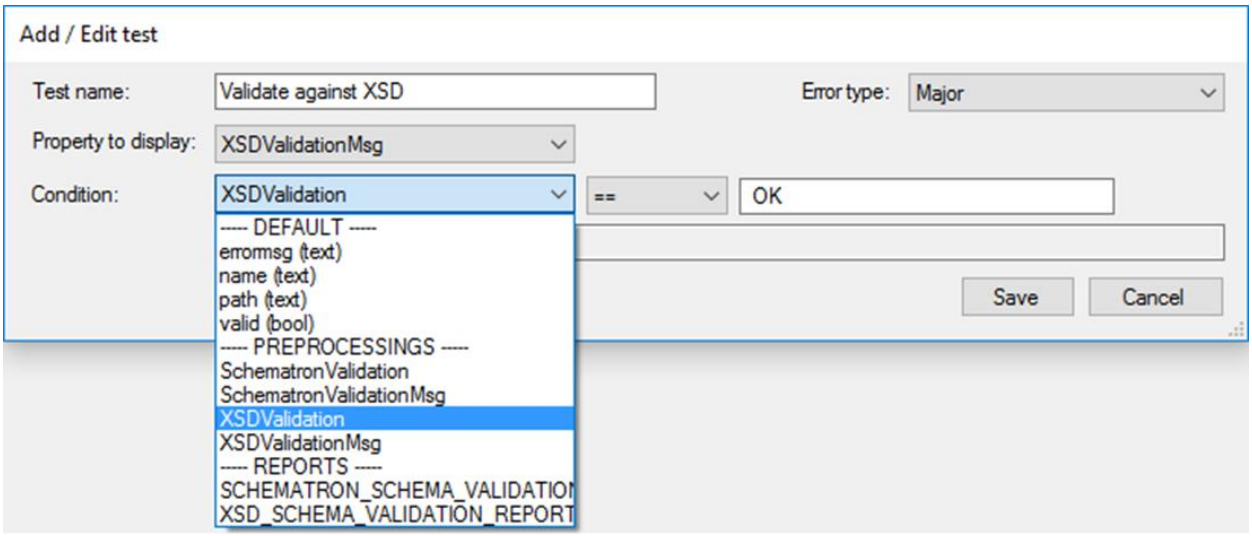


XSD_SCHEMA_VALIDATION will validate the XML file against a .xsd schema. As parameters, you need:

- **propName** – the name of the property used in test condition
- **propNameDisplay** – used to display the validation result in the report
- **schema** - the .xsd file that the XML file will be compared with. The entire path, together with the file name are needed for this parameter.

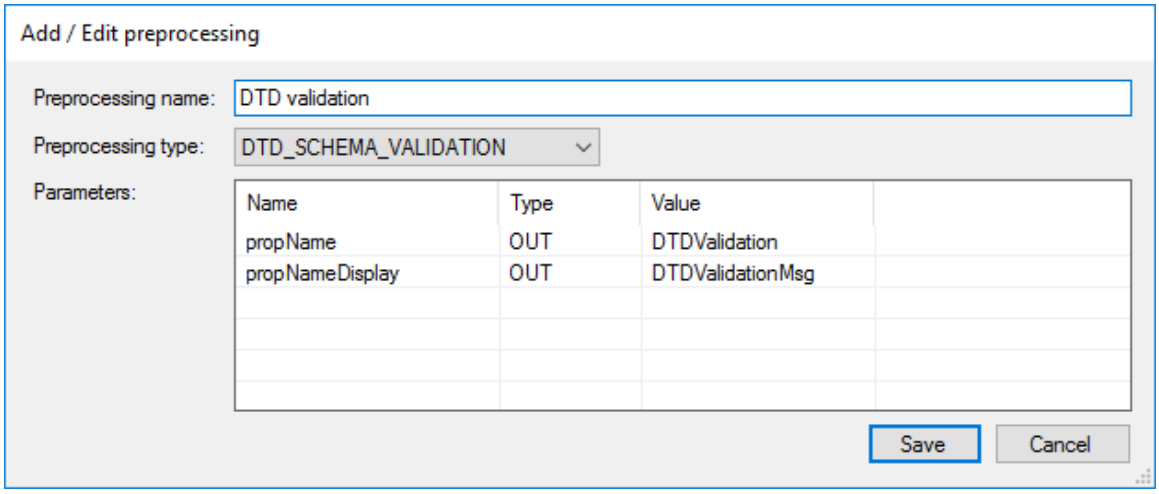


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

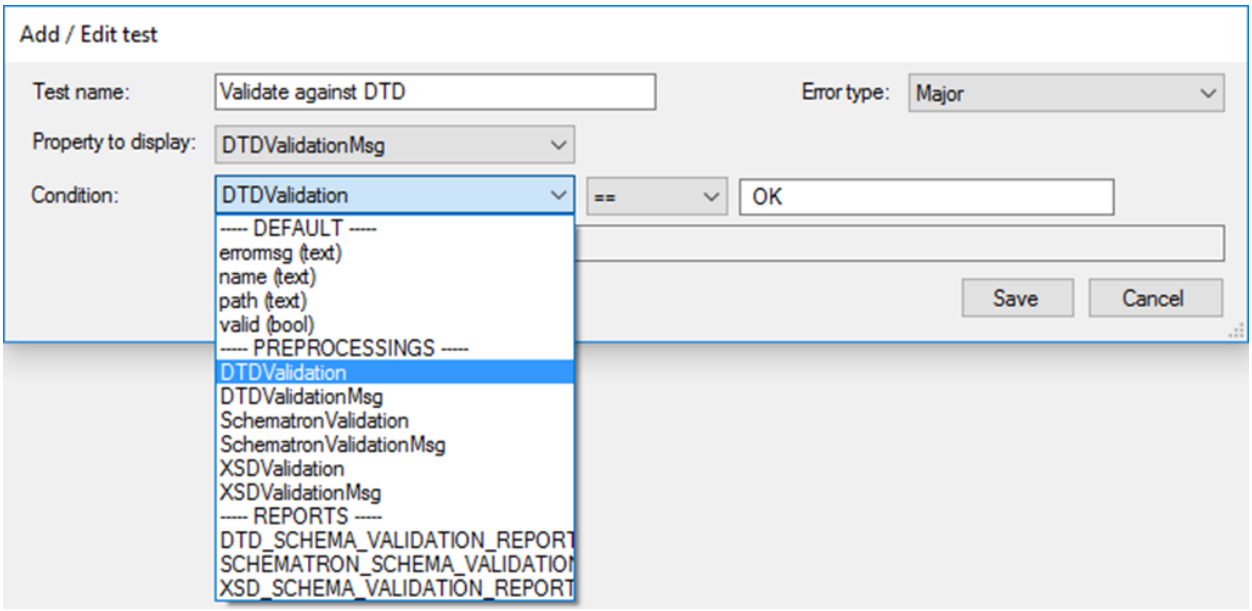


DTD_SCHEMA_VALIDATION will check the XML file against DTD schema. Define the values for the parameters:

- **propName** – the name of the property used in test condition
- **propNameDisplay** – used to display the validation result in the report

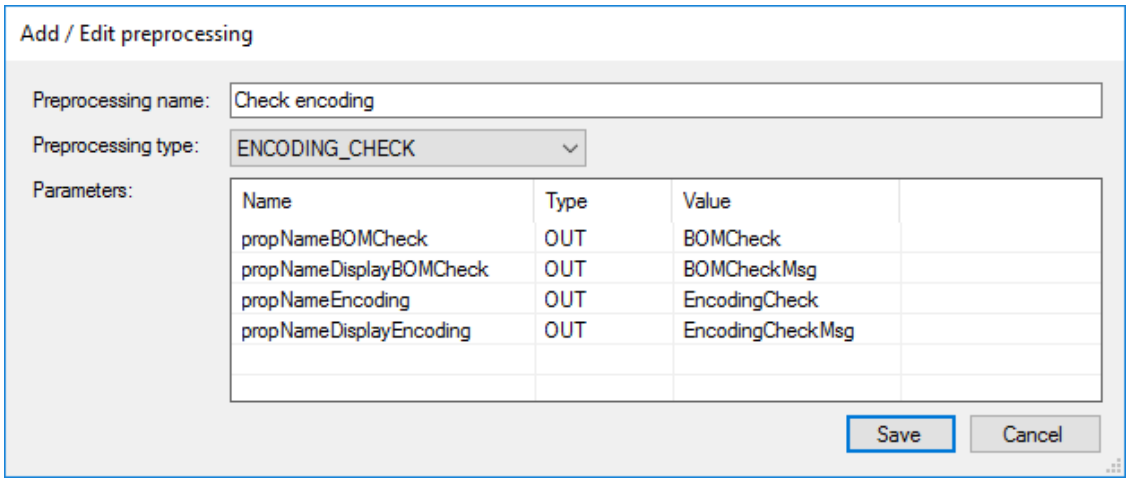


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:

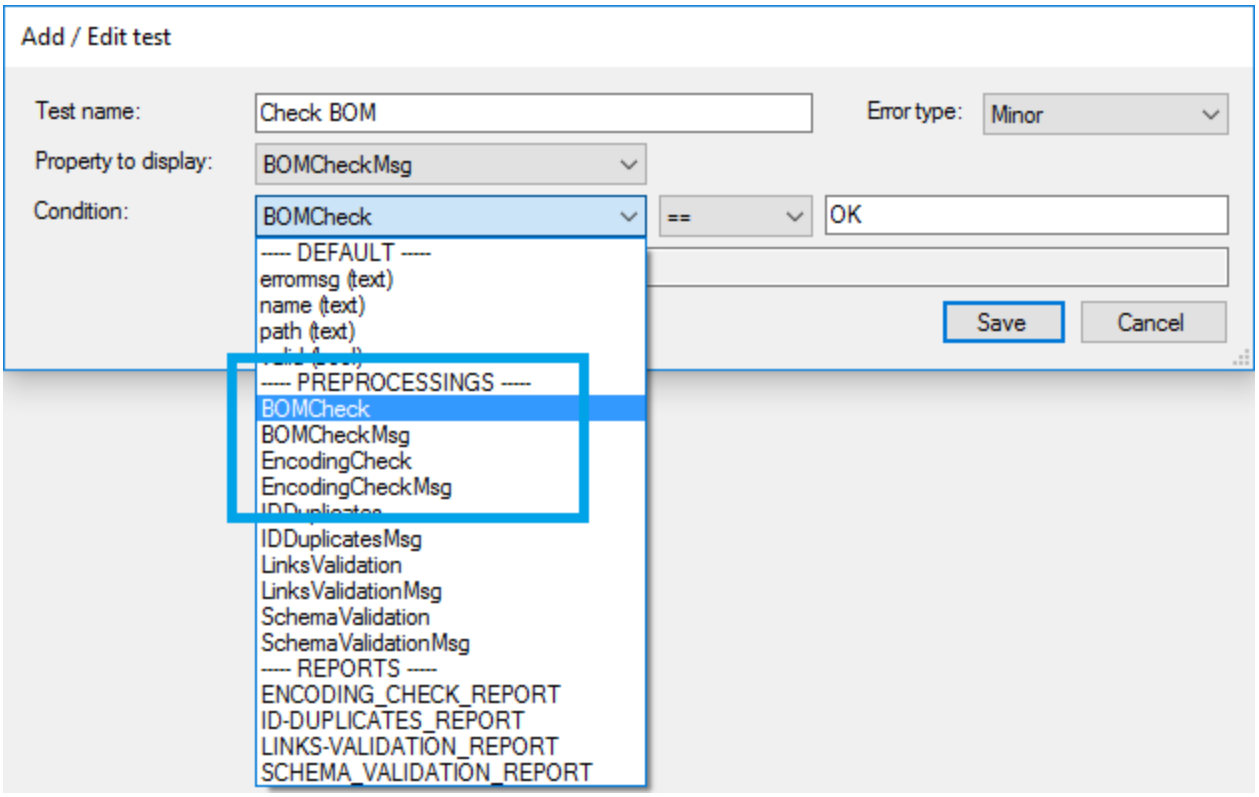


ENCODING_CHECK can be used to check the file encoding and BOM property. The parameters for this preprocessing are:

- **propNameBOMCheck** – used as the property in the test condition for BOM
- **propNameDisplayBOMCheck** – used to display the validation result for BOM in the report
- **propNameEncoding** – used as the property in the test condition for encoding
- **propNameDisplayEncoding –** used to display the validation result for encoding in the report

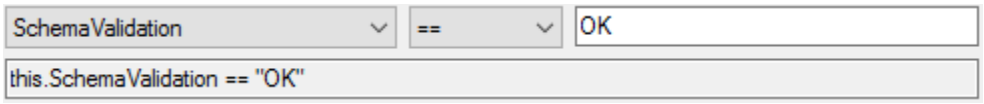


The properties generated will be automatically added in **Property to display** and **Condition** dropdowns of **Add / Edit test** dialog, under “PREPROCESSINGS” group:



Condition

All properties generated by the preprocessings for XMLPlugin need to be compared against *OK* in the condition. The tests will return *OK* if the validation was done successfully and other strings if it failed (which will be displayed in the report by *propNameDisplay*). So, the condition must look like this:



WARNING! The validation process is case-sensitive, so adding values like *Ok* or *ok* will lead to failed tests.

For default properties, check the type of the property displayed in brackets. You will notice that the conditions will be formatted based on the type:

- for text (strings), the quotation marks will be automatically added
- for numeric, no quotation mark will be added, but there will be validation rules to allow only numbers
- for bool, only *true* or *false* are allowed as values.

TCLPlugin

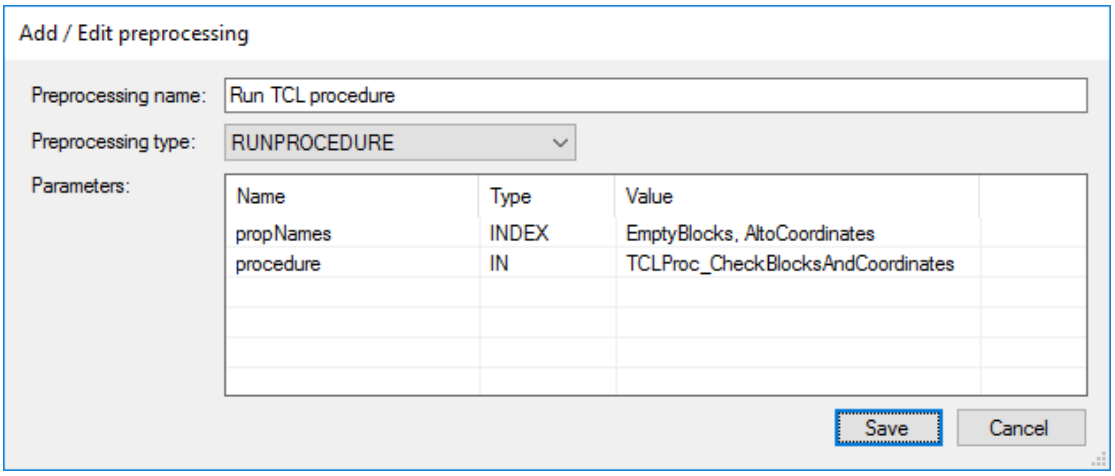
Default properties

Property	Property to display	Type
name	name	Text
path	path	Text

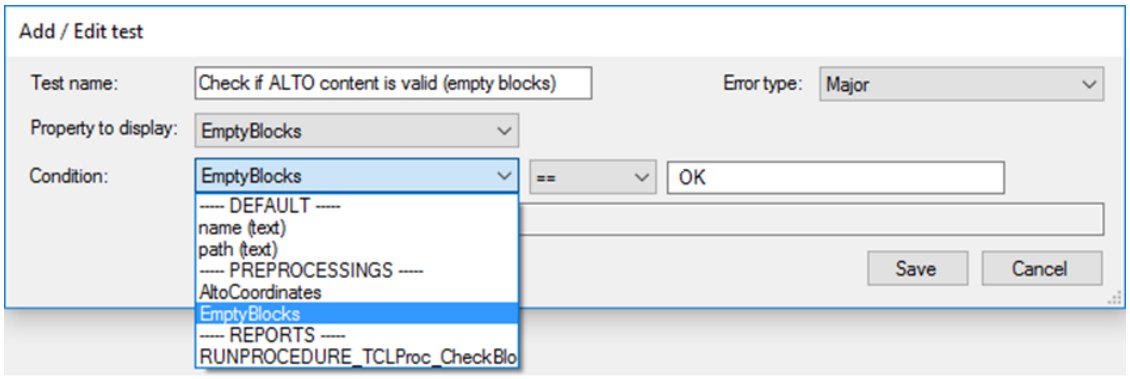
Preprocessing

Only one preprocessing is available for TCLPlugin: **RUNPROCEDURE**. This will get as IN parameter a TCL procedure that will return some properties.

- **propNames** – a list with properties that the TCL procedure returns. If there are more than one properties returned, they need to be separated by comma ","
- **procedure** – The name of the TCL procedure that will be run.



The properties returned will all be added in the **Add / Edit test** dialog:



Condition

Because the RUNPROCEDURE preprocessing works with a custom procedure, you need to know what the properties returned are and their type, but also what to compare them to. As a standard rule, we return *OK* when the test is successful also in TCL procedures, but feel free to define the procedure in the way that best suits your needs.

For default properties, check the type of the property displayed in brackets. You will notice that the conditions will be formatted based on the type:

- for text (strings), the quotation marks will be automatically added
- for numeric, no quotation mark will be added, but there will be validation rules to allow only numbers
- for bool, only *true* or *false* are allowed as values.