Voith's CODE - Notes Fields Extractor Sample

Last updated; April 19th, 2011

Notes Fields Extractor is still undergoing development, and thus both the application and the documentation are likely to change.

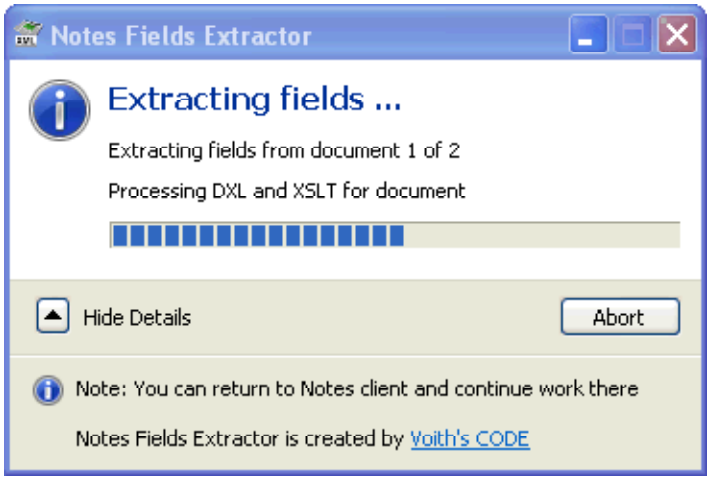
Also note that the trial-version of Notes Fields Extractor has some limitations;

- Maximum 5 documents will be extracted for each run
- All extracted images will be watermarked by Voith's CODE logo
- The XML depth *may* be altered.

What is Notes Fields Extractor?

Notes Fields Extractor (NFE) is a tool to aid you, the programmer, to extract content from Lotus Notes documents. It is shipped as a Windows 32-bit DLL with an accompanying dialog box executable.

You control the operation and what to extract via either LotusScript or @Formula-language functions. As soon as LotusScript or @formula commands tell NFE to extract documents, it will launch the small user interface seen below;



Version History

Release	Date	Description
1.0.0.x	Initial shipdate	The initial release
1.0.0.342	19. APR. 2011	Introduced the options ImageFileNameFormula and AttachmentFileNameFormula. These options allow the user to place images and attachments in specified directories and specify the naming of the files. Since the formulas are evaluated against the active Notes documents, the user can use any valid @-formula! This means that you can name your images with employee-numbers or whatever content you may have in your documents.

Below you will see a short description of the LotusScript functions. Please refer to the Agents within this database for practical implementations of the functions.

Name	Declaration	Description
NFEGetMajorVersion	Declare Function NFEGetMajorVersion Lib "NDBVCNFE.DLL" Alias "NFEGetMajorVersion" () As Long	Get the major version from the DLL. This function is typically used to identify the NFE version if you need to have separate LotusScript logic according to evolving capabilities in newer versions of NFE. The format of the version number is; <major version>.<minor version>.<feature version>.<buildnumber>, a full example of a complete version number could be 1.0.0.286 or 1.1.0.567
NFEGetMinorVersion	Declare Function NFEGetMinorVersion Lib "NDBVCNFE.DLL" Alias "NFEGetMinorVersion" () As Long	Get the minor version from the DLL. See NFEGetMajorVersion for more details
NFEGetBuildNumber	Declare Function NFEGetBuildNumber Lib "NDBVCNFE.DLL" Alias "NFEGetBuildNumber" () As Long	Get the build number from the DLL. See NFEGetMajorVersion for more details
NFEGetVersion	Declare Function NFEGetVersion Lib "NDBVCNFE.DLL" Alias "NFEGetVersion" (_ rstrVersion As String, _ Byval iBufferLen As Long) As Long	Get the complete version string. Remember that you must allocate a fixed number of characters in your LotusScript string before calling this function
NFEClear	Declare Function NFEClear Lib "NDBVCNFE.DLL" Alias "NFEClear" (_ Byval pstrOptions As String) As Long	Clear any existing settings. This function is typically one of the absolutely first functions to call, to ensure that you don't try to process old left-overs from previous processing. The parameters are; pstrOptions -See section *Valid Options* below

Continue on next page

Name	Declaration	Description
NFESetDatabase	Declare Function NFESetDatabase Lib "NDBVCNFE.DLL" Alias "NFESetDatabase" (_ Byval pstrNotesDatabase As String, _ Byval pstrNotesServer As String,_ Byval pstrOptions As String) As Long	Set the database you intend to extract documents from. The parameters are; pstrNotesDatabase - The database filename or replica id. The filename can be specified either as a full path to a local database, or to a relative filename if database is on a server. pstrNotesServer - The server the database is on. If blank or "Local", the database spesified by the paramater above, is local. pstrOptions - See section *Valid Options* below
NFEAddDocument	Declare Function NFEAddDocument Lib "NDBVCNFE.DLL" Alias "NFEAddDocument" (_ Byval pstrNoteID As String, _ Byval pstrOptions As String) As Long	Add a single document to the list of documents to extract. This can be useful if you are in a loop and for example just want to process the selected documents in a view. If you intend to process a lot of documents, please consider using the NFEAddAllDocumentsInView or NFEAddAllDocumentsInViews which allows you to specify view names and process all documents within views. The parameters are; pstrNoteID - A single NoteID for the document to extract pstrOptions - See section *Valid Options* below
NFEAddDocuments	Declare Function NFEAddDocuments Lib "NDBVCNFE.DLL" Alias "NFEAddDocuments" (_ Byval pstrNoteIDs As String, _ Byval pstrOptions As String) As Long	Add a list of semicolon-separated note ids to the list of documents to extract. This is just a variant of the function above, which let you add a semicolon-separated list of noteids in one go. The parameters are; pstrNoteIDs - A list of semicolon-separated note ids for the documents to extract pstrOptions - See section *Valid Options* below
NFEAddAllDocumentsInView	Declare Function NFEAddAllDocumentsInView Lib "NDBVCNFE.DLL" Alias "NFEAddAllDocumentsInView" (_ Byval pstrViewName As String, _ Byval pstrOptions As String) As Long	Another way to specify documents to extract, and this time you specify a view name or view alias, which tells NFE to extract all documents within that view! The parameters are; pstrViewName - A view name or view alias of the view to enumerate for documents to extract. pstrOptions - See section *Valid Options* below

Name	Declaration	Description
NFEAddAllDocumentsInViews	Declare Function NFEAddAllDocumentsInViewsLib "NDBVCNFE.DLL" Alias "NFEAddAllDocumentsInViews" (_ Byval pstrViewNames As String, _ Byval pstrOptions As String) As Long	A list of semicolon-separated view names/aliases to extract documents from. In other words, another powerful way of specifying lots of documents. The parameters are; pstrViewNames - A list of semicolon-separated view names/aliases of the views to enumerate for documents to extract. pstrOptions - See section *Valid Options* below
NFESetIncludeFields	Declare Function NFESetIncludeFields Lib "NDBVCNFE.DLL" Alias "NFESetIncludeFields" (_ Byval pstrListOfFields As String, _ Byval pstrOptions As String) As Long	By default NFE will extract all available fields from the documents. You can however limit the fields you want by specifying a list of semicolon-separated field names. The parameters are; pstrListOfFields - A list of semicolon-separated field names to extract from the documents pstrOptions - See section *Valid Options* below
NFESetOutput	Declare Function NFESetOutput Lib "NDBVCNFE.DLL" Alias "NFESetOutput" (_ Byval pstrBaseDirectory As String, _ Byval pstrOutputType As String, _ Byval pstrOptions As String) As Long	Where do you want to store the extracted content? By specifying a so-called base directory, NFE will store the main XML in this directory, and create any necessary subdirectories per document if they contain images, attachments or OLE objects. Note that this version of NFE only support XML as output type, and that only images and attachments are extracted. OLE objects are not supported in this version. The parameters are; pstrBaseDirectory - The base directory where NFE should store the output pstrOutputType - Leave as a blank string, for future use. pstrOptions - See section *Valid Options* below
NFEExtractFields	Declare Function NFEExtractFields Lib "NDBVCNFE.DLL" Alias "NFEExtractFields" (_ Byval pstrOptions As String) As Long	Finally, this function starts the extraction process accoring to the preparation you have done above. Note that NFE will launch a small dialog box informing you about the progress. NFE is completely multitasking, so it won't lock your Notes client while extracting data. You may therefore switch back to Lotus Notes and continue with other work The parameters are; pstrOptions - See section *Valid Options* below

Valid Options

Almost all functions in NFE accept extra options as the last parameter. The table below specifies the valid options and how to use them. Also note that each option has a release number when the option was introduced. This means that you won't find support for the specified options in older releases of NFE.

How to specify options?

Options is a text string, where each option and value is separated by a colon, like this;

option:value

For example:

```
ImageFileNameFormula:"D:\\Employee Images\\" + @Text(EmployeeNbr)) + ".jpg"
```

The option is blue while the value is green. If the option needs more than one value, each value is separated with a comma.

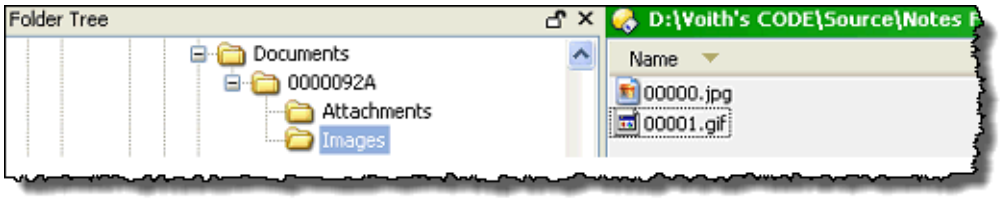
Multiple option and value pairs are separated with semi-colon.

ImageFileNameFormula - 1.0.0.342

Prior to this release, NFE would dump all images in the Images-sub directory per document Note ID. The images was given a filename corresponding to the sequence-number in the document. Think of a formula equal to;

<Base Directory>\<Note ID>\Images\<sequencenumber>.<filetype>

An example would be:



In the screenshot above you see that the <Base Directory> has some initial directory to start with, ending with "Documents" above. Then comes the <Note ID> subdirectory "0000092A", the hardcoded "Images" subdirectory, and finally two sequence-numbered images with their original filetypes.

With the ImageFileNameFormula you can change this completely by specifying a full @-formula. The formula will be executed against the current document, which means that you can use any field value or anything that end up being a valid @-formula!

For example;
`ImageFileNameFormula:"C:\\Temp\\" + @Text(@NoteID)) + ".jpg"`
... will always same images to "C:\Temp\<Note ID>.jpg" filename.

`ImageFileNameFormula:"D:\\Employee Images\\" + @Text(EmployeeNbr)) + ".jpg"`
... will always same images to "D:\Employee Images\<the content of the field EmployeeNbr>.jpg" filename.

Please note how I must use double backslashes to denote a single backslash in the formula. This is common requirement in @-formula. This means that you should be *sure* that the formula evaluates to something sensible before using it in NFE.

To assist you creating filenames, some special variables can be used in the formulas too. Always remember that the variables must be placed inside quotes since they are strings. The variables are:

$(BASEPATH) - The current base directory, without trailing backslashes
$(NOTEID) - The current Note ID with leading zeros
$(SEQ) - The sequence number of the current image, with leading zeros
$(DEFFILETYPE) - The default filetype of the current image. Smart to use so you don't mess up the file types.

Some examples with variables, the first one recreate the default path from scratch:

```
ImageFileNameFormula:"$(BASEPATH)\\$(NOTEID)\\Images\\$(SEQ).$(DEFFILETYPE)
"
```

.
A final sample;

```
ImageFileNameFormula:"$(BASEPATH)\\All
images\\$(NOTEID)_$(SEQ).$(DEFFILETYPE)"
```

... will place all images in the base directory's All images-subdirectory. The image names has a combination of the Note ID plus the sequence number, with the default file type, such as "0000092A_00000.jpg"

AttachmentFileNameFormula - 1.0.0.342

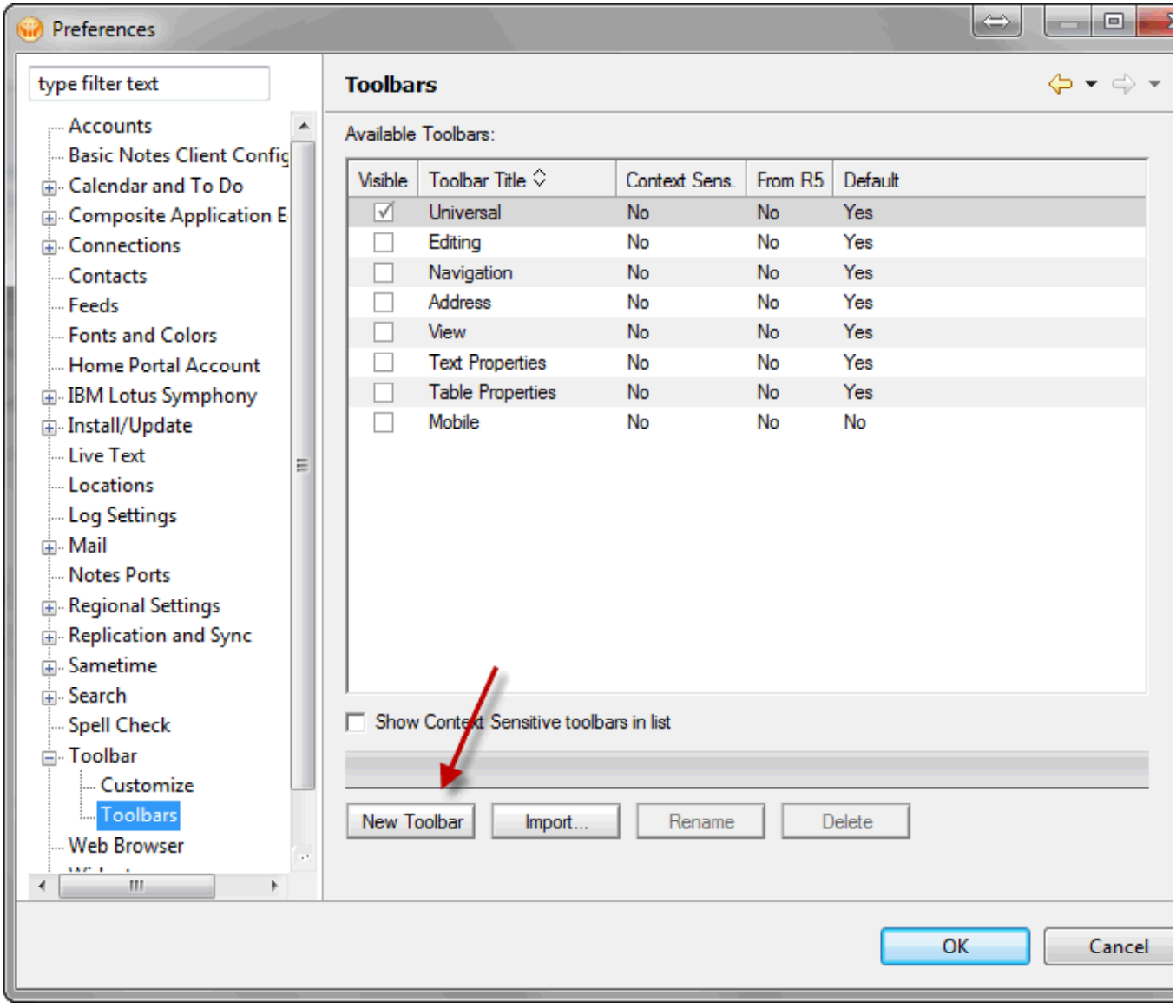
The same as ImageFileNameFormula, but now you address any attachments which normally was placed in the hardcoded Attachments-subdirectory. The valid variables for this options are;

$(BASEPATH) - The current base directory, without trailing backslashes
$(NOTEID) - The current Note ID with leading zeros
$(SEQ) - The sequence number of the current attachment, with leading zeros
$(ORGATTFILENAME) - The original file name as it was attached with in the Notes document

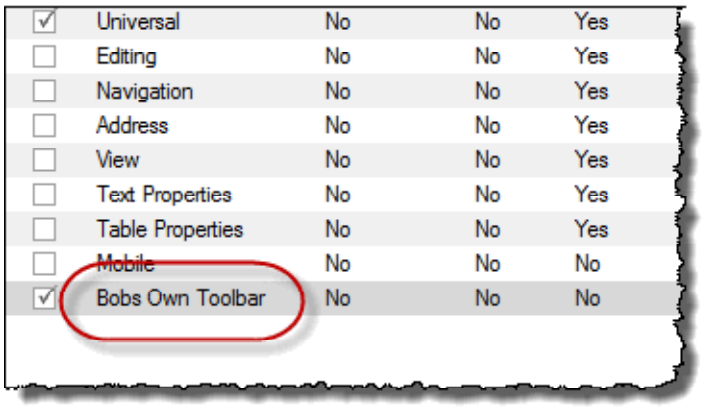
Add Notes Fields Extractor as a button on your toolbar!

One of the ways to use Notes Fields Extractor is from a generic button in a toolbar. This means that you can extract any selected document (note, document in singular form, not multiple documents - this is a limitation by the toolbar-implementation in the Notes client) from any database, without further programming.

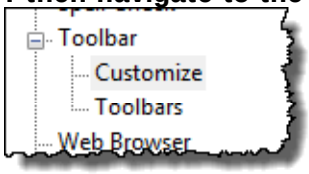
First, open up your *Preferences* dialog box and select the *Toolbar* -section. Personally I like to place my buttons in my own toolbar, so I can quickly locate and find my own tools. But you can of course place the button in whatever toolbar you'd like. Below you see how I create my own toolbar ...



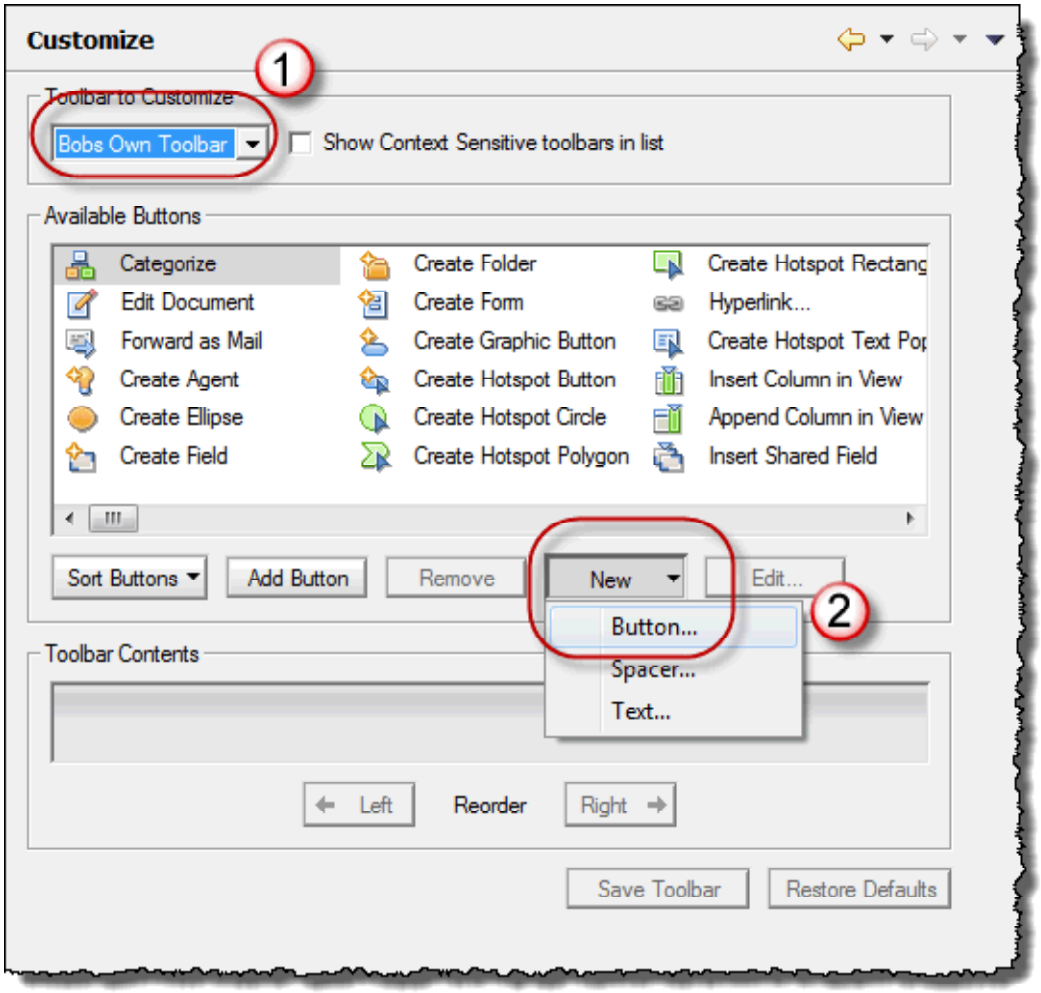
... and name it *Bobs Own Toolbar* ...



I then navigate to the *Customize* -part in the Preferences dialog box;

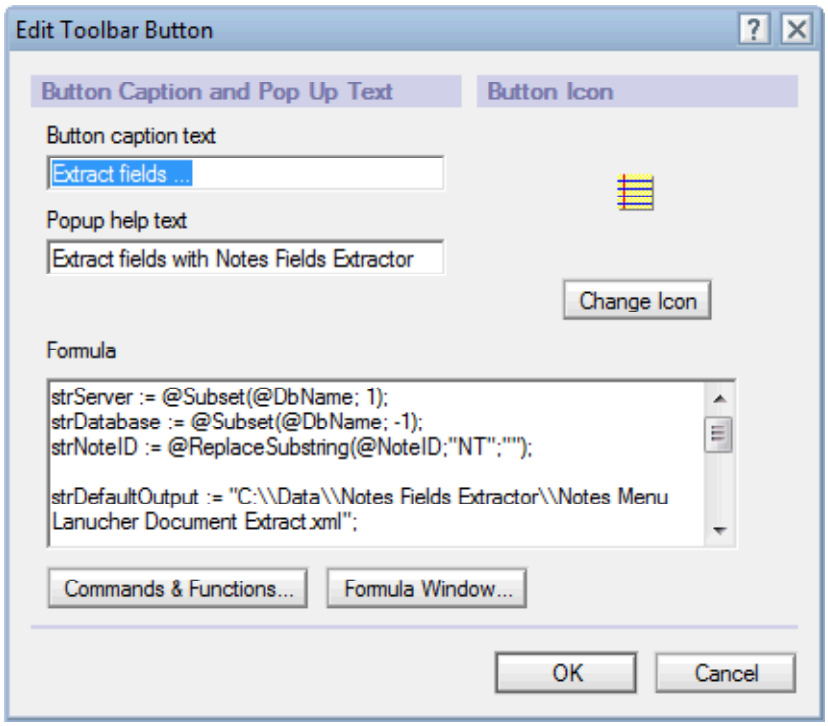


Ensure you have selected the toolbar you want to place your button in in the field *Toolbar to Customize* ; Then click on the *New* -button and select the *Button-* menu line....



This brings up the *Edit Toolbar Button* dialog box, where you add some descriptive text,

select an icon and finally add some formula-code, like this;



The code that I have used so far is this;

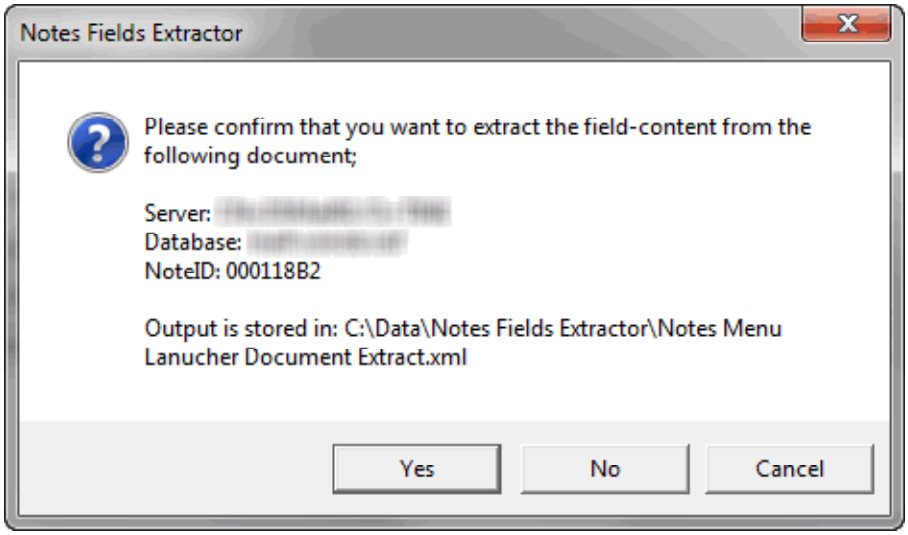
```
strServer := @Subset(@DbName; 1);
strDatabase := @Subset(@DbName; -1);
strNoteID := @ReplaceSubstring(@NoteID;"NT";"");

strDefaultOutput := "C:\\Data\\Notes Fields Extractor\\Notes Menu Lanucher
Document Extract.xml";
strMsg := "Please confirm that you want to extract the field-content from
the following document;" + @NewLine + @NewLine + "Server: " + strServer +
@NewLine + "Database: " + strDatabase + @NewLine + "NoteID: " + strNoteID +
@NewLine + @NewLine + "Output is stored in: " + strDefaultOutput;

rc := @Prompt([YesNoCancel];"Notes Fields Extractor"; strMsg);
@If(rc != 1; @Return("");"");

@DbCommand("VCNFE":"NoCache";"GetFields"; strServer ; strDatabase ;
strNoteID ; ""; strDefaultOutput; "")
```

This code will first present a dialog box like this;



....which gives me a chance to abort. Note how I have specified the output folder too with the parameter strDefaultOutput. This is the default folder and XML file for the result.