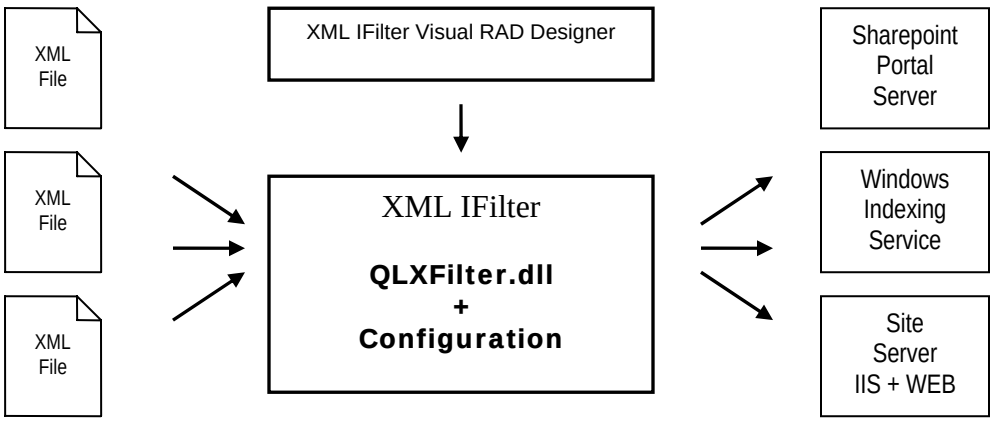
XML IFilter

for easy XML file indexing

User's Guide – V 4.03

www.quilogic.cc



Part No DOC-2007-19
Version 4.03

Contact

You can contact us via any of the following paths

Web	www.quilogic.cc
Support	support@quilogic.cc
Sales Inquiries	sales@quilogic.cc
Executive Office	office@quilogic.cc
FAX	+43 (533) 93544
Telephone	+43 (533) 93544

Before requesting support, it would save both your time and ours if you could do the following:

- Make sure you have read any relevant portions of the manual
- Isolate the problem to a small test case
- Have the version number ready (see readme.txt)
- Have the type of environment, version number and operating system ready.
- Give us an example of the faulting Query statement including

License Agreement

This software package and its documentation are subject to the following license agreement. By installing and using the package, you are implicitly accepting these terms and conditions:

END-USER LICENSE AGREEMENT FOR XML IFilter SOFTWARE

IMPORTANT-READ CAREFULLY.

This QuiLogic, Inc**. XML IFilter** End-User License Agreement ("EULA") is a legal AGREEMENT between you (either as a registered individual user or as the registered user/representative and on behalf of a single entity, “Licensee”) and QuiLogic Software Corporation for the XML IFilter software product identified above, which product includes computer software and may include associated media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, then DO NOT install or use the SOFTWARE PRODUCT; in such event the original purchaser may, however, return it to the place of purchase within thirty days of the date of original purchase for a full refund.

SOFTWARE PRODUCT LICENSE

1) GRANT OF LICENSE.

Subject to the payment of the applicable license fees, and subject to the terms and conditions of this Agreement, QuiLogic hereby grants to you a non-exclusive, non-transferable right to use one copy of the specified version of the Software and the accompanying documentation (the "Documentation"). You may install one copy of the Software on one computer, workstation, server or other electronic device for which the Software was designed (each, a "Client Device"). If the Software is licensed as a suite or bundle with more than one specified Software product, this license applies to all such specified Software products, subject to any restrictions or usage terms specified on the applicable price list or product packaging that apply to any of such Software products individually.

Use: The Software is licensed as a single product; it may not be used on more than one Client Device or by more than one user at a time, except as set forth in this Section 1. The Software is "in use" on a Client Device when it is loaded into the temporary memory (i.e., random-access memory or RAM) or installed into the permanent memory (e.g., hard disk, CD-ROM, or other storage device) of that Client Device. This license authorizes you to make one copy of the Software solely for backup or archival purposes, provided that the copy you make contains all of the Software's proprietary notices.

Server-Mode Use: You may use the Software on a Client Device as a server ("Server") within a multi-user or networked environment ("Server-Mode") only if such use is permitted in the applicable price list or product packaging for the Software. A separate license is required for each Client Device or "seat" that may connect to the Server at any time, regardless of whether such licensed Client Devices or seats are concurrently connected to, accessing or using the Software. Use of software or hardware that reduces the number of Client Devices or seats directly accessing or utilizing the Software (e.g., "multiplexing" or "pooling" software or hardware) does not reduce the number of licenses required (i.e., the required number of licenses would equal the number of distinct inputs to the multiplexing or pooling software or hardware "front end").

If the number of Client Devices or seats that can connect to the Software can exceed the number of licenses you have obtained, then you must have a reasonable mechanism in place to ensure that your use of the Software does not exceed the use limits specified for the license you have obtained. This license authorizes you to make or download one copy of the Documentation for each Client Device or seat that is licensed, provided that each such copy contains all of the Documentation's proprietary notices.

Volume License Use: If the Software is licensed with volume license terms specified in the applicable product invoicing or packaging for the Software, you may make, use and install as many additional copies of the Software on the number of Client Devices as the volume license terms specify. You must have a reasonable mechanism in place to ensure that the number of Client Devices on which the Software has been installed does not exceed the number of licenses you have obtained. This license authorizes you to make or download one copy of the Documentation for each additional copy authorized by the volume license, provided that each such copy contains all of the Documentation's proprietary notices.

Enterprise License Use: If the Software is licensed with enterprise license terms specified in the applicable product invoicing or packaging for the Software, you may make, use and install as many additional copies of the Software on the unlimited number of Client Devices within Licensee's organization. You must have a reasonable mechanism in place to ensure that the number of Client Devices on which the Software has been installed is controlled for reference and audit purposes. This license authorizes you to make or download one copy of the Documentation for each additional copy authorized by the enterprise license, provided that each such copy contains all of the Documentation's proprietary notices.

This material is sold "as is". QuiLogic, Inc. makes no warranties, either expressed or implied, regarding the enclosed software package, its merchantability, or its fitness for any particular purpose. Information in this document is subject to change without notice and does not represent a commitment on the part of QuiLogic, Inc. While every effort is made to insure that the above mentioned product and its documentation is free of defects, QuiLogic, Inc. shall NOT be held responsible for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential or other damages occasioned by the use of this product.

It is assumed that purchasers of this product are familiar with basic programming skills. This is a highly technical product, offered in a rapidly evolving programming environment. QuiLogic, Inc. will provide support to purchasers of this product for 365 days after its purchase and receipt (bug reports and comments are always welcome). Support questions may be submitted either by e-mail or fax. QuiLogic, Inc. reserves the right to respond to questions in responding by e-mail or fax.

2) DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

Limitations on Reverse Engineering, Decompilation, and Disassembly. You may not modify, reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation. The SOFTWARE PRODUCT is licensed as a single product. Except with respect to the Redistributables, its component parts may not be separated for use on more than one computer.

Not for Resale Software. If the SOFTWARE PRODUCT is labeled "Not for Resale" or "NFR" or "Evaluation Copy", then, notwithstanding other sections of this EULA, you may not use the SOFTWARE PRODUCT for commercial purposes nor sell, or otherwise transfer it for value. Commercial purposes include the use of the SOFTWARE PRODUCT to create publicly distributed computer software.

Rental. You may not rent, lease, or lend the SOFTWARE PRODUCT to any party.

Software Transfer. You may permanently and wholly transfer all of your rights under this EULA, provided you (a) retain no copies (whole or partial), (b) permanently and wholly transfer any and all of the SOFTWARE PRODUCT (including all component parts, the media and printed materials, any upgrades, this EULA, and, if applicable, the Certificate of Authenticity) to the recipient, and (c) the recipient first

agrees to abide by all of the terms of this EULA. If the SOFTWARE PRODUCT is an upgrade, any transfer must include any and all prior versions of the SOFTWARE PRODUCT and any and all of your rights therein, if any.

Support Services. QuiLogic, Inc. may provide you with support services related to the SOFTWARE PRODUCT ("Support Services"). The provision and use of Support Services is governed by the QuiLogic, Inc. policies and programs described in the SOFTWARE PRODUCT user manual and/or in "online" documentation. Any supplemental software code provided to you as part of the Support Services shall be considered part of the SOFTWARE PRODUCT and subject to the terms and conditions of this EULA. With respect to technical information you provide to QuiLogic, Inc. as part of the Support Services, QuiLogic, Inc. may use such information for its business purposes, including for product updates and development.

Termination. Without prejudice to any of QuiLogic's other rights, QuiLogic, Inc. may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy any and all copies of the SOFTWARE PRODUCT and all of its component parts.

3) UPGRADES. If the SOFTWARE PRODUCT is labeled or otherwise identified by QuiLogic, Inc. as an "upgrade", you must be properly licensed to use a product identified by QuiLogic, Inc. as being eligible for the upgrade in order to use the SOFTWARE PRODUCT. A SOFTWARE PRODUCT, labeled or otherwise identified by QuiLogic, Inc. as an upgrade, replaces and/or supplements the product that formed the basis for your eligibility for such upgrade. You may use the resulting upgraded product only in accordance with the terms of this EULA. If the SOFTWARE PRODUCT is an upgrade of a component of a package of software programs that you licensed as a single product, the SOFTWARE PRODUCT may be used and transferred only as part of that single product package and may not be separated for use on more than one computer.

4) COPYRIGHT AND TRADEMARKS.

All title, trademarks and copyrights in and pertaining to the SOFTWARE PRODUCT, the accompanying printed materials, and any copies of the SOFTWARE PRODUCT, are owned or licensed by QuiLogic, Inc. or its affiliated companies. The SOFTWARE PRODUCT is protected by copyright and trademark laws and international treaty provisions. You may make one copy of the SOFTWARE PRODUCT for back-up and archival purposes. You may not copy the printed materials accompanying the SOFTWARE PRODUCT.

You may not remove, modify or alter any QuiLogic, Inc. copyright or trademark notice from any part of the SOFTWARE PRODUCT, including but not limited to any such notices contained in the physical and/or electronic media or documentation, in the QuiLogic, Inc. Setup Wizard dialogue or ‘about’ boxes, in any of the runtime resources and/or in any web-presence or web-enabled notices, code or other embodiments originally contained in or otherwise created by the SOFTWARE PRODUCT.

5) DUAL-MEDIA SOFTWARE. You may receive the SOFTWARE PRODUCT in more than one medium. Regardless of the type or size of the medium you receive, you may use only that one medium that is appropriate for your single computer. You may not use or install the other medium on another computer, including but not limited to portable computers under the exclusive control of the registered developer. You may not loan, rent, lease, or otherwise transfer the other medium to another user, except as part of the permanent transfer (as provided above) of the SOFTWARE PRODUCT.

6) AUSTRIAN GOVERNMENT RESTRICTED RIGHTS. The SOFTWARE PRODUCT and documentation are provided with RESTRICTED RIGHTS. This EULA shall be construed, interpreted and governed by the laws of the Austrian country.

7) HIGH RISK ACTIVITIES. The Software is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software

could lead directly to death, personal injury, or severe hysical or environmental damage ("High Risk Activities"). QuiLogic and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

8) LIMITED WARRANTY. QuiLogic, Inc. warrants that (a) the SOFTWARE PRODUCT will, for a period of ninety (90) days from the date of delivery, perform substantially in accordance with QuiLogic's written materials accompanying it, and (b) any Support Services provided by QuiLogic, Inc. shall be substantially as described in applicable written materials provided to you by QuiLogic, Inc.

CUSTOMER REMEDIES. In the event of any breach of warranty or other duty owed by QuiLogic, Inc., QuiLogic's and its suppliers' entire liability and your exclusive remedy shall be, at QuiLogic's option, either (a) return of the price paid by you for the SOFTWARE PRODUCT (not to exceed the suggested U.S. retail price) if any, (b) repair or replacement of the defective SOFTWARE PRODUCT or (c) re-performance of the Support Services. This Limited Warranty is void if failure of the SOFTWARE PRODUCT has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE PRODUCT will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

NO OTHER WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, QUILOGIC, INC. AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE PRODUCT AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THE LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION. Some states and jurisdictions do not allow disclaimers of or limitations on the duration of an implied warranty, so the above limitation may not apply to you. To the extent implied warranties may not be entirely disclaimed but implied warranty limitations are allowed by applicable law, implied warranties on the SOFTWARE PRODUCT, if any, are limited to ninety (90) days.

9) LIMITATION OF LIABILITY. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL QUILOGIC, INC. OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF QUILOGIC, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, QUILOGIC's ENTIRE LIABILITY UNDER ANY PROVISION OF THIS EULA SHALL BE LIMITED TO THE AMOUNT YOU ACTUALLY PAID TO QUILOGIC, INC. FOR THE SOFTWARE PRODUCT OR SERVICE THAT DIRECTLY CAUSED THE DAMAGE. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU

Table of Contents

Introduction **1**
Supported Systems: 1
Technical Requirements 1
Features **2**
Operation Principle **3**
4 Simple Steps to Setup XML Indexing **4**
Query Technology Inside **6**
RAD Designer **7**
Working with the Designer, Step by Step **8**
Create a new filter description 8
Customizing a filter 9
Assign a file extension 9
Define a condition 10
Add property to index 11
Define property content 13
Adding a comment 15
Automatic generation of definition data 15
Testing the filter 16
Deploying the filter 17
Query Basics **18**
XPath 20
FILE keyword 20
WHERE clause 21
DISTINCT 23
Aggregate Functions 23
Return clause 25
Data types 25
Availability **27**
References **27**

Introduction

QuiLogic's XML IFilter enables crawling of documents containing XML based data. For the first time ever, our filter technology enables users to search and index arbitrary structured xml files based on content. Likewise VISIO® or Excel® files can be saved as xml files and indexed with the help of QuiLogic's XML IFilter!

XML IFilter extends the standard functionality of Microsoft Internet Information Server (IIS), Microsoft Share Point Portal Server (SPS), and all other products based on top of Microsoft Indexing Service Technology.

XML IFilter contains a Rapid Application Development (**RAD) tool** which let you **visually design and test** your indexing application. No coding is required. The RAD tool takes care to generate all necessary registry entries and definition files for you.

Supported Systems:

XML IFilter can be used with the following products:

- Microsoft Share Point Portal Server®
- Microsoft Share Point Team Services®
- Microsoft Windows® Indexing Service
- Microsoft Site Server®
- All other products based on Microsoft Indexing Technology

Technical Requirements

XML IFilter is self-contained and requires no other additional products to install.

The following information is extracted from an xml file:

- Name and text content of any element.
- Name and text content of any attribute.
- Date, time, numeric and boolean content from any element and attribute.
- Recognizes all usual character sets used by xml files (UTF8, UTF16 …).

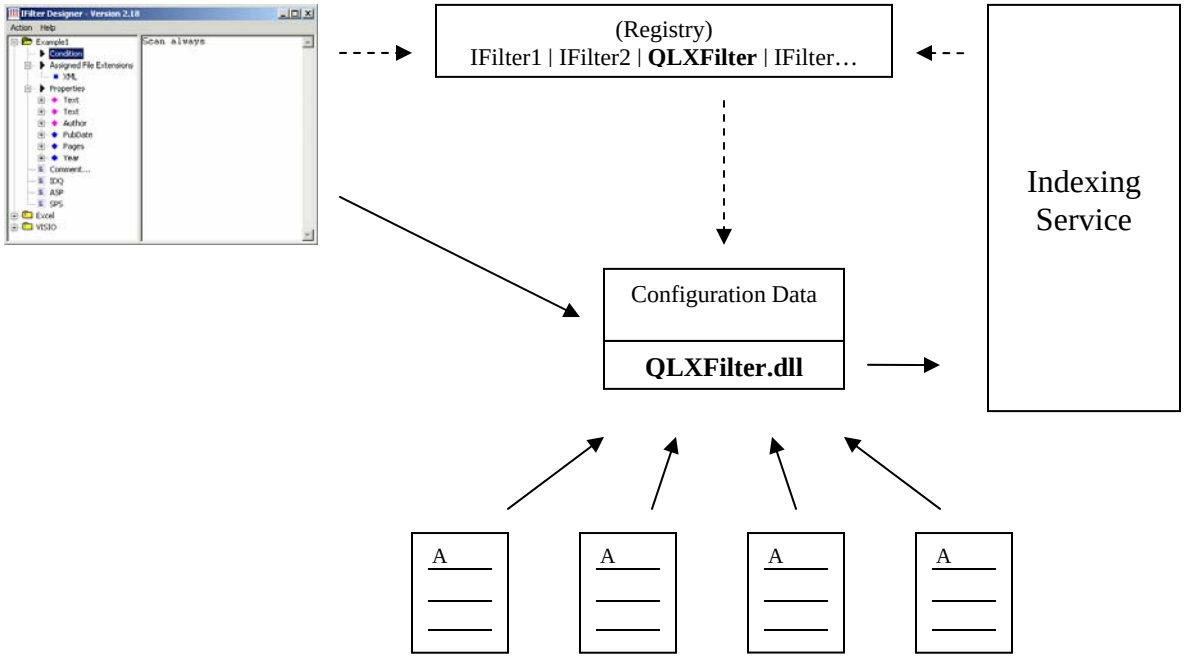
The included RAD tool let you visually design and specify which data to extract for each returned property. It might be either the content of a single element or any combination of attribute and element values including the names of elements in any order.

Features

- Build in “Probing Technology”. Enables crawling and indexing of multiple, arbitrary structured xml files with totally different content but having the same file name extension.
- Build in facility to “normalize” the returned property text to a common string. A typical example might be the indexing of telephone numbers as shown below.
- **343-6790-555** or **#343-6790-555** or ‘**343 6790 555**’ can be normalized to: **3436790555**
- Assign more than one file extension to “common structured” XML files.
- Indexing can be fine tuned by parameters for controlling exactly what data to index from your XML files.
- Option Switch to ignore <XML> or <HTML> tags during indexing.
- Visual RAD Designer included for developing indexing solutions without coding.
- Designer takes care to set the required registry entries as needed.
- Designer creates required property definition files for SPS, ASP and WEB search applications.
- Build in facility to create “Abstracts” from totally different locations in the XML Data.
- Precisely specify and fine tune the returned content for “full text” indexing.
- Test-Mode available to check the data outcome of your indexed properties.
- Installation program included for quick and easy setup.
- Enterprise edition contains full C++ Source code for the IFilter implementation.
- Debugging Aid available for better troubleshooting your indexing application.
- Ultrahigh performance, makes the indexing of even hundreds of thousands XML documents a snap.

Operation Principle

XML IFilter is a dynamic link library (dll) file that provides a bridge between any Microsoft indexing client and files containing the xml data. When an indexing client needs to index content from documents it will look in the registry for an appropriate filter dll based on the type of the document file name extension (.doc, .xml …). By using the included RAD tool, users can visually specify the file extensions assigned as well as the content and properties to extract. That information is written into a configuration file to drive the filter dll at indexing time and to update the registry based association between document extensions and indexing filters.



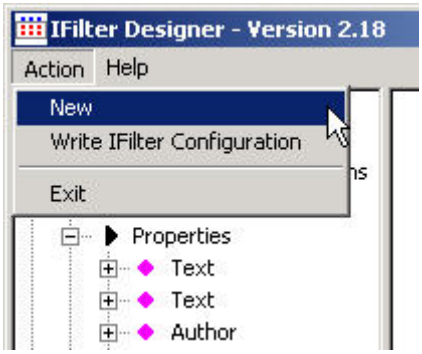
The Design tool let you create multiple filter descriptions. Each description is assigned an individual indexing condition. The condition is used by the filter dll to “probe” an xml file during the file scan. The file is indexed with a particular filter description only when the probing passed successfully the condition. In this respect, it is possible to scan xml files having the **same extension**, but totally **different xml content** and structure !

The user assigns each filter description a set of individual properties to be retrieved during the xml document scan. The data content of the returned property is individually laid down by the user, taking into account the structure of the underlying xml document.

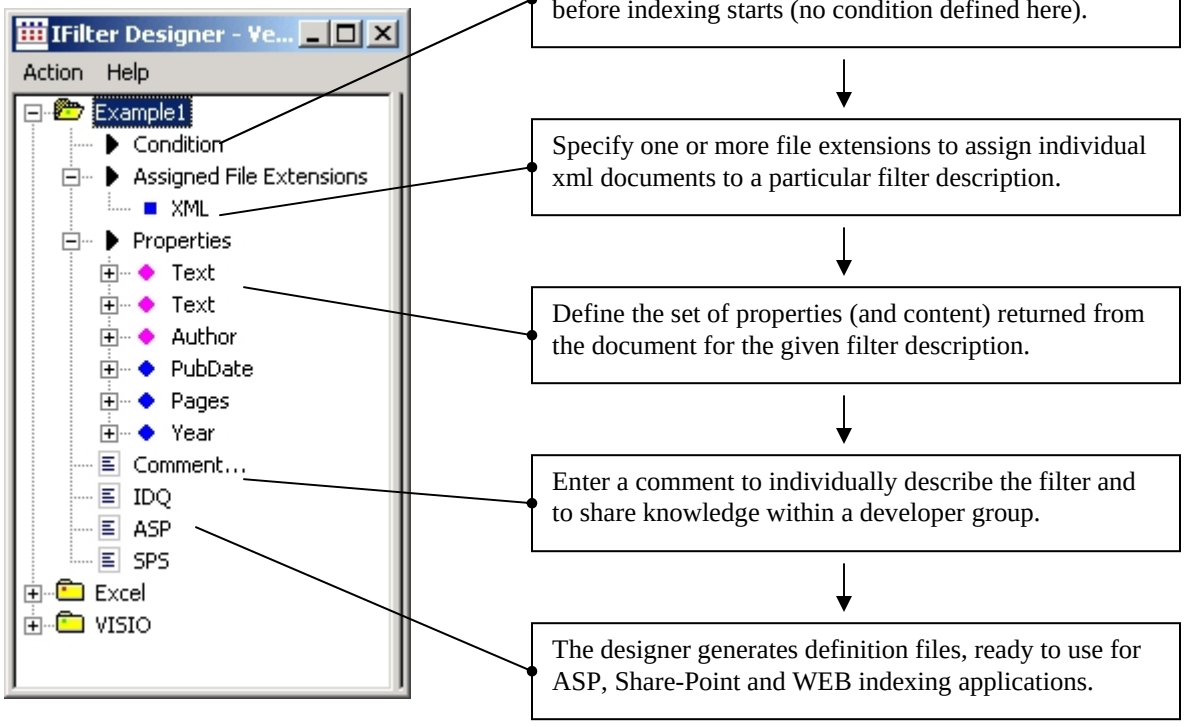
With the above described methods it is possible to fine tune the indexing of *any* xml based document with individual internal structure and file name extension.

4 Simple Steps to Setup XML Indexing

1. Create a new filter description

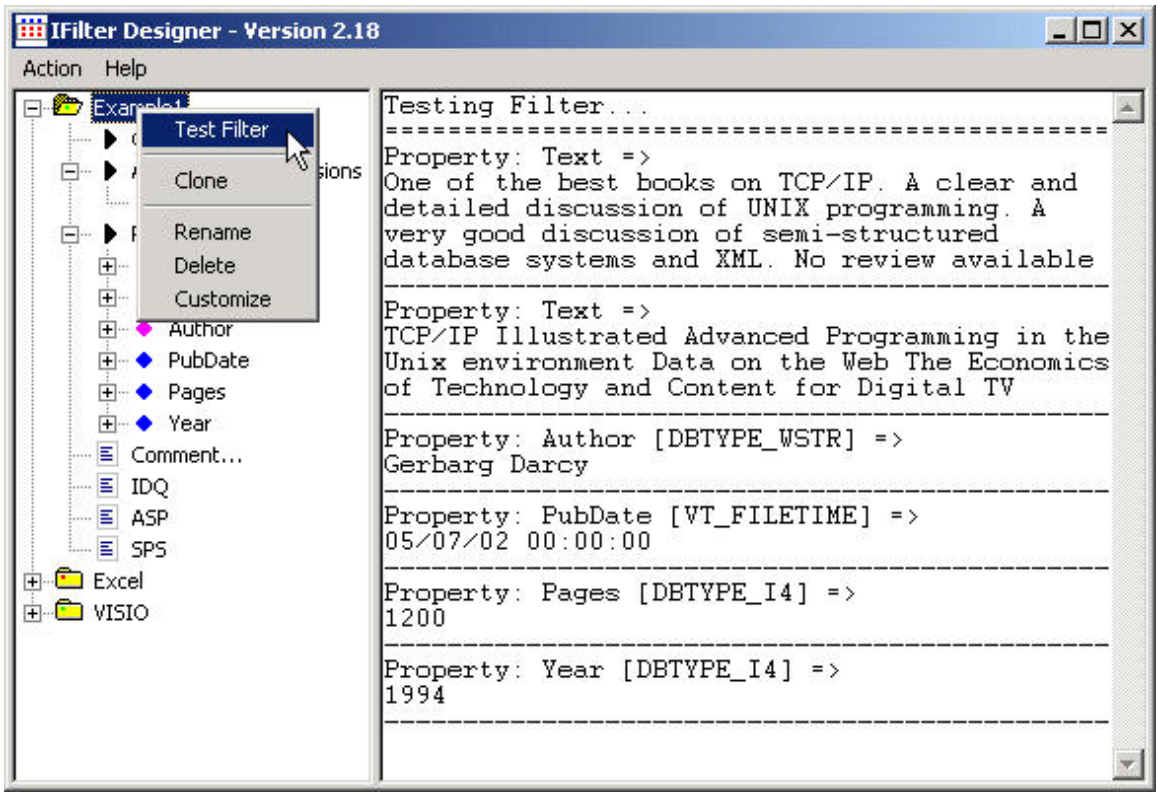


2. Edit filter description

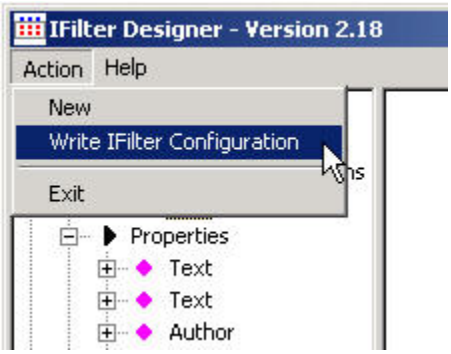


During indexing, the XML IFilter (QLXFilter.dll) is called file by file from the Indexing Service. The filter applies *all* defined filter-descriptions one by one, probing individually and returning the property data for which a match was found. Each property again has a query condition assigned which must be true to return any content for that property. Properties can be selected from a set of predefined standard properties (content, author, title …) or individually defined to match the user requirements.

3. Test filter



4. Deploy



Within the last step, the filter configuration is written to the configuration file, the registry updated and the indexing service stopped and restarted. At this point, full-text queries and SPS dashboard site simple search (‘contains’ and ‘freetext’) will work.

Query Technology Inside

Behind the scene QuiLogic's XML Database and Query Technology provides the heart beat to drive the QLXFilter indexing engine. Even multi megabyte sized xml files with arbitrary complexity and tree depth can be indexed fast and easy.

QuiLogic provides a visual RAD tool which enables the user to specify exact the individual content to be retrieved for each property. To do so, the user formulates simple queries which describe the data that should be returned. A simplified version of XQuery (a W3C draft specification for querying xml data; see References) is used.

For example, to return the content of *all* xml attribute tags, use the following simple XQuery statement for a content property:

```
For $X in FILE//@* Return $X/text();
```

… or to query an author property use:

```
For $X in FILE/abc/author WHERE … Return $X/text();
```

XQuery contains the well known XPath language (see References) to select sub trees out of a larger xml tree. In the return clause, you specify what to return. To return data types other then text (numeric, bool, date, time), a simple data type specification is append to the referenced variable in the return statement (/bool(), /text(), /date()…).

The design tool let you assign individual queries for all defined properties and in this way you specify the returned content. At run time the indexing engine applies these queries to retrieve the associated content. This works very fast, because during file load a special index is build on the fly, taking into account the predefined queries.

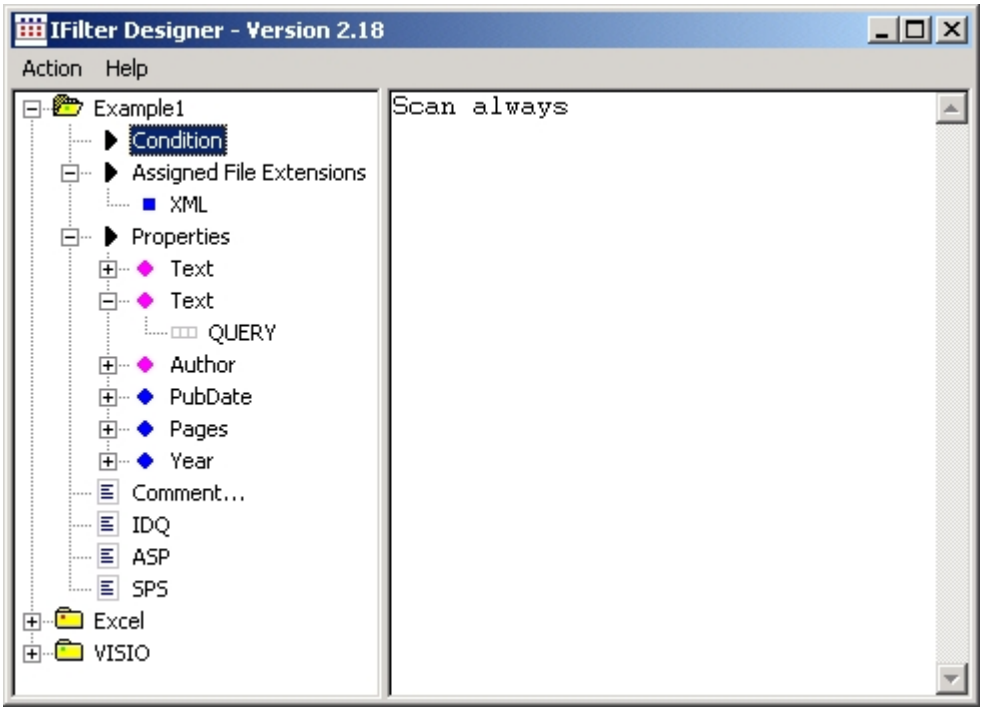
XML content of any complexity can be indexed by this method. Even numeric data, build up from the content of several elements, can be indexed due to the possibility to apply aggregate functions like SUM, AVG, … on the XQuery statement.

QuiLogic has developed another product, **SQL/XML-IMDB** a universal in-memory database engine which (to our knowledge) is the only software product available on the marked for managing sql and xml based data within one component. SQL/XML-IMDB is a combined native SQL and XML database and part of QuiLogic's information integration strategy to unify structured and unstructured data from sources such as relational databases, xml documents, flat files and Web services (SOAP).

The engine is available for NET, VB, Delphi, C++ and Perl from QuiLogic. A white paper can be downloaded from our website, which describes the database in more detail (see next section).

RAD Designer

The RAD design tool let you visually design and test your indexing application. The RAD tool takes care to generate all necessary registry entries and definition files for you. The designer has a simple and intuitive user interface.



The working area consists of two panes. The left pane is the design pane where the main work takes place. The right pane is the information output side, showing additional information and user feedback for selected items from the left side.

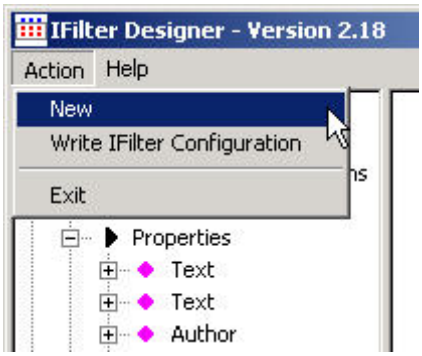
The entire program is menu driven. Additional actions can be selected by right clicking the mouse over an item which pops up context sensitive menus.

With the Design tool you create one or more filter descriptions, edit all the necessary information and parameters, to correctly drive the scanning dll during the indexing step.

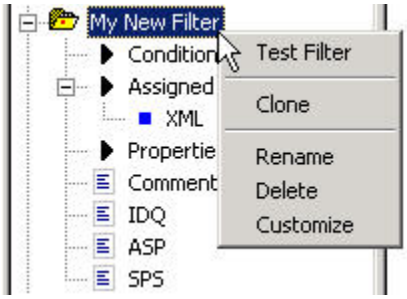
Working with the Designer, Step by Step

Create a new filter description

To crate a new filter description, select `Action->New` from the menu:



A new filter entry will be created with some parameters set to default values by the designer. To edit the newly created entry simple right click the mouse over the new entry:



The context menu enables you to

- Test the Filter (described in more detail below).
- Clone an entire filter description.
- Rename the filter description.
- Delete a filter description.
- Customize a filter description.

Customizing a filter, let you set the filter active/inactive or whether the xml scanner should honor any possible “robot” tag entry in the xml file.

Customizing a filter

Click "**Filter is active**" to set the filter active. Only active filter descriptions will be recognized by the XML IFilter indexing engine (QLXFilter.dll). Deactivated filters are useful to create "template" filter descriptions which can be cloned and set to active after modifying the necessary parameters. The state of the filter is visually reflected by the red or green dot inside the yellow directory symbol.

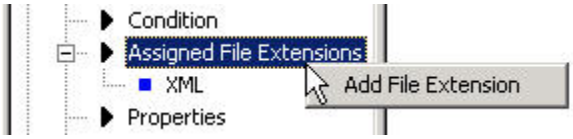
If the xml file contains the element:

```
<meta name="robots" content="noindex"/>
```

and "**Robot Meta Tag**" is checked, then the file will be **excluded** from indexing regardless of any additional "Condition" defined or not.

Assign a file extension

Right click to add a file extension:



When an indexing client needs to index content from documents it will look in the registry for an appropriate filter dll based on the type of the document file name extension (.doc, .xml …). The Designer registers all file extensions found in *all* (active!) filter descriptions for the QLXFilter.dll. When the Indexing Service is looking into the registry it will find a certain file extension associated with QLXFilter.dll and then call the QLXFilter Dll to scan the file. The filter will receive the file name and extension from the Index Service and based upon this information he can select the **corresponding** filter description to use during the file scan. If you have **more** than one filter description associated with the same extension, **all** that filter descriptions will be applied during the file scan. Of course, you can even assign more then one extension to a particular filter description. Extensions can be deleted and renamed at any time by right clicking on it.

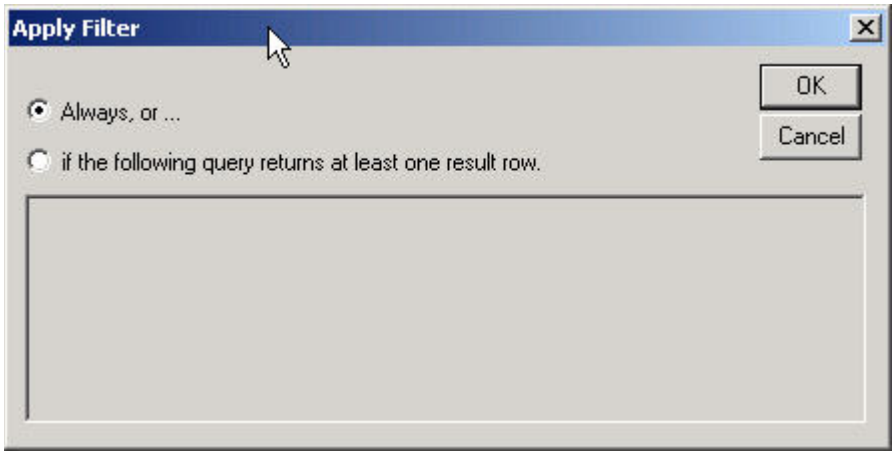
Define a condition

The Indexing Service calls a specific IFilter dll based on the extension class it founds on the file name. As a matter of fact most xml documents have the same core extension (.xml) but can contain very different data and structure. The problem is now, how to index documents with varying xml structure inside.

The Design tool let you create multiple filter descriptions and for each description it is possible to assign an individual indexing condition which is applied during file parsing.

The specified condition is used by the IFilter implementation (QLXFilter.dll) to "**probe**" an xml file during the file scan. The file is indexed only, when the probing passed successfully the condition. In this respect, it is possible to scan xml files having the same extension, but totally different xml content and structure !

Conditions can be assigned to any filter description. Simple right- or double click on the "Condition" item, which will bring up the following dialog:

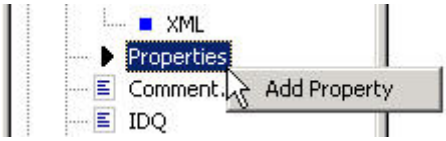


Selecting the button "Always" will set the condition permanently to true. The given filter description is always applied, regardless of any probing. To choose a selective file scan based on document content enter the adequate query into the edit field below the buttons.

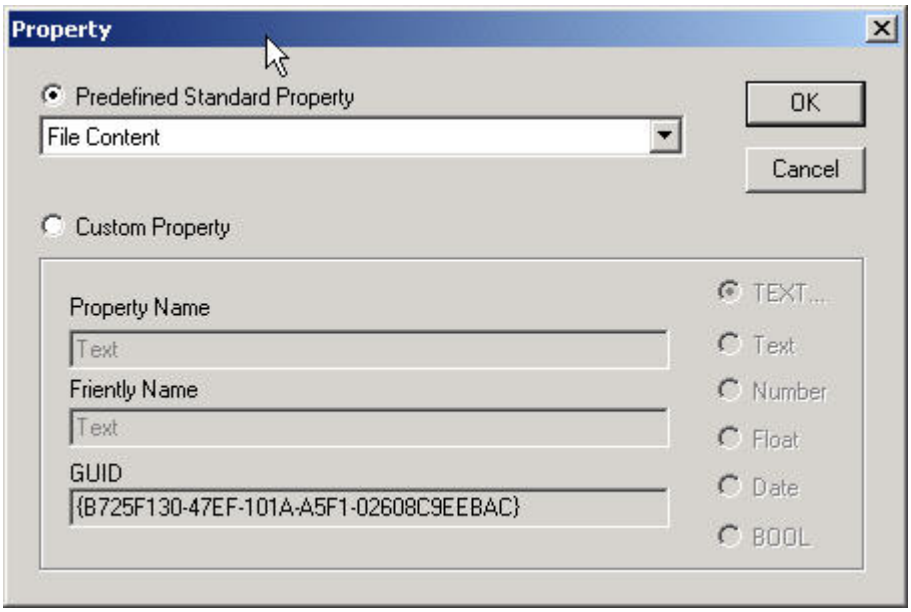
The formulation of a query condition is based on a simplified form of the XQuery language. The detailed usage and syntax is covered in the chapter "Query Basics".

Add property to index

IFilters extract the content of documents through properties (author, title, and so on). This step let you define what properties and content you will extract for the given filter description. To do so, right click on the “Properties” item:



This will pop up the following property specification dialog:



Xml documents contain text and values. Sure, even the value content is in text form, but it can be interpreted as having a value of a certain type. For example, the string “07/12/1999” can be interpreted as being a sequence of characters (text) or being of type date having the value of a given date. Thus, each property is said to be either a **text-type** or a **value-type** property. Text-type properties are stored in the **word list** index of Indexing Service and represent the (possible complex) content of a document. Text type properties are typically queried by “contains” and “freetext” searches. Value-type properties represent a **single** property only, taken from the entire document (e.g.: Author).

A text type property has a list of words (as unformatted text) associated with it. A value-type property has a type (text, date, time, number…) and a corresponding value and is

stored in the property cache of the Indexing Service. The type of the property therefore determines how Indexing Service indexes the property and what kind of querying Indexing Service can perform with the property.

Each property has a unique name and a GUID associated with it. The GUID is either automatically created by the RAD design tool, or predetermined in the case of "well known" standard properties (see below). The friendly name is optional and can be used as a substitute in the search applications build upon Indexing Service.

Microsoft encourages all users of Indexing Service to adopt sets of well known properties so that client applications can use **one** query to search for a certain property across **all** file classes. Therefore it is recommended to select, whenever possible, one of the properties contained in the standard-list by clicking on "Predefined Standard Property".

To create a custom property click on "Custom Property" to enable the input fields below the radio button.

You must provide a name and data type for a custom property. To create a text-type property select the radio button labeled "TEXT…", to create a value-type property check one of the other buttons to create a property of a certain type. Remember that text-type properties represent (possible long) unformatted text (word list) whereas value-type properties represent a single value (which could be even a text string, of course).

You can define and add more than once the same property (same name and GUID) to a given filter description!

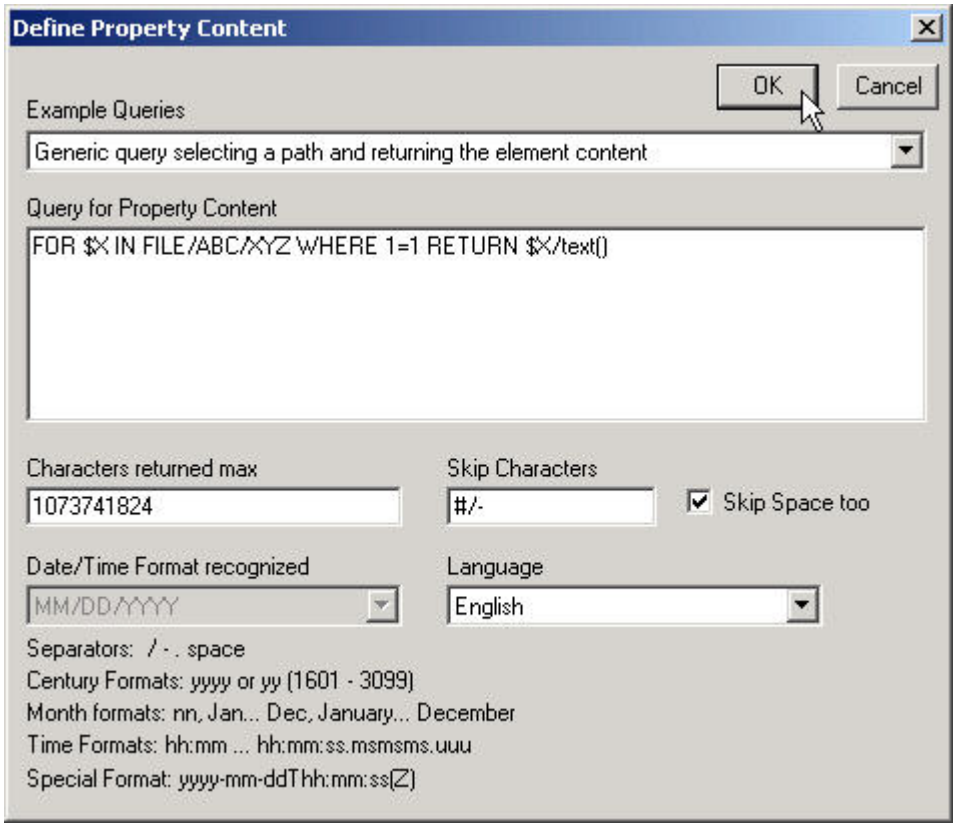
This makes sense (only) for text-type properties which can then return concatenated content from **different locations** in your xml file. You simple assign different queries (see below) to each property. The scanning engine, when detecting the same multiple defined properties, simple concatenates the output of each query to return only a single result.

Right clicking on the property item, let you edit or delete the property.

Define property content

To define what exactly the property should return from the xml document you need to define a query. Right- or double-clicking the QUERY item just below the property item…

… show's up the query editor dialog:



XML IFilter uses a simplified form of the XQuery language. The detailed usage and syntax is covered in the chapter "Query Basics". A number of predefined example queries, representing common tasks, are available by selecting one of the entries in the combo box labeled "Example Queries". It is possible to extend and customize the example list, by simple adding an entry in the "<examplequeries>" section of the designer configuration file (qlfiltxml.xml) from which the combo box gets filled.

On the dialog, you will find the following entry fields:

- **Query for property content**

Enter the property specific query here. Accepts only one query at a time. See the chapter "Query Basics" for syntax details.

- **Characters returned max**

This field let you specify how many characters should be returned from the query. Enter a value between 1 and 1073741824. The field is ignored and disabled if you have a property with a data type other than text. The value entered in this field my be important and useful if you have a property, which should represent for example an abstract of the document content build up from the say first 1000 characters of a particular xml element.

- **Language**

It is very likely that your site has documents written in several languages, some of which have multiple languages interspersed within them. The language information is important for the Indexing Service to choose the correct word breaker and stemmer component.

This field let you set the language in which the document is written in. You can even have different languages in the same document! Simple specify the same property more than once but use different, selective queries for every property targeting a different language. The scanning engine assigns the correct language identifier (LCID) for every property fetched from the document. The field is ignored and disabled if you have a property with a data type other then text.

- **Date/Time Format recognized**

This field configures the build-in date/time parser to correctly parse and recognize date/time typed properties. The build in parser is quite flexible in recognizing various string formats. Even exotic formats like that one used in Excel and Visio files (yyyy-mm-ddThh:mm:ssZ) will be recognized.

- **Skip Characters**

Enter up to 7 characters (no space between them!) which should be skipped when returning text-type content for a property. This feature is a facility to "normalize" the returned property text to a common string. A typical example might be the indexing of telephone numbers as shown below.

> **343-6790-555** or **#343-6790-555** or '**343 6790 555**' can be normalized to: **3436790555**

by entering "-#" in the field and checking the check-box "Skip space too".

Adding a comment

Right- or double clicking on the “Comment…” item lets you enter a new comment.

Comments are useful for a short description of the filter or for sharing knowledge in a group of developers.

Automatic generation of definition data

Before queries can be issued over a custom property, Index Server needs to be given a name for the property. Adding a property definition to the [names] section of an IDQ file accomplishes this. An example line might be look like:

```
Author_fn (DBTYPE_WSTR) = d1b5d3f0-c0b3-11cf-9a92-00a0c908dbf1 Author
```

… where the left side word “Author_fn” represents the **friendly name** entered in the Property definition dialog.

Defining a property in Active Server Pages (ASP) is similar. An example line might be look like:

```
Q.DefineColumn " Author_fn (DBTYPE_WSTR) = d1b5d3f0-c0b3-11cf-9a92-
00a0c908dbf1 Author"
```

These definitions tell Index Server that the custom property named **Author** will be referred to as **Author_fn**. The property is defined as a wide (Unicode) string value, and the long string of letters and numbers is the GUID (generated by the designer) that uniquely identifies all properties.

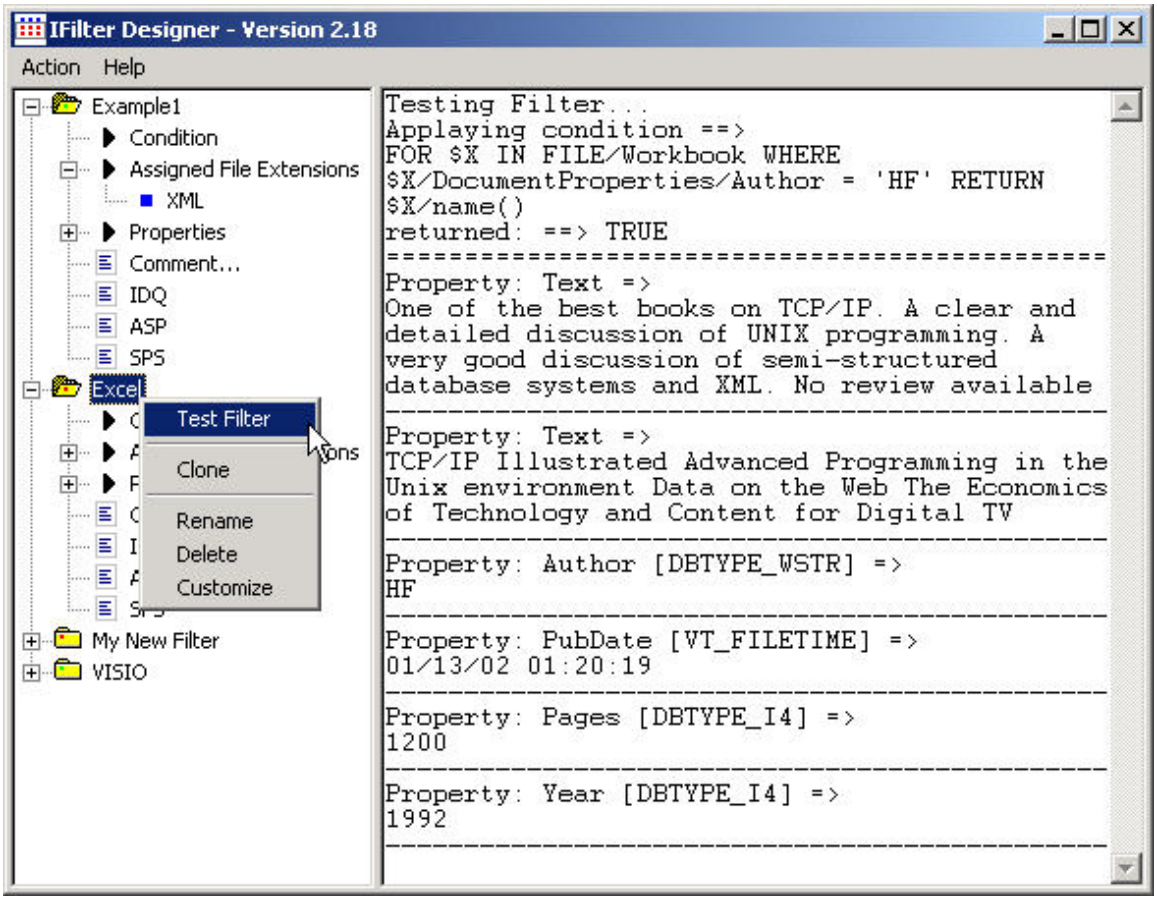
Once the property name is defined, it can be used to issue content queries.

The designer generates 3 definition files ready to use in your search application. One for IDQ/HTX, one for ASP and one for the Share Point Portal Server. Simple copy the generated data to the clipboard and paste it in your application.

Testing the filter

Having successfully entered all properties, queries and file associations it's time to test if it works as expected. In preparation for testing, you should locate an example xml file on which the filter description can be tested.

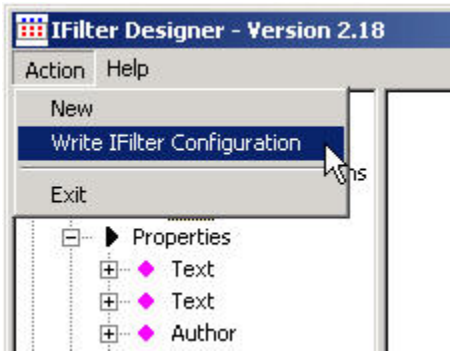
Right click on the filter description and select "Test Filter". After selecting the test file the designer will load the file and apply the condition and properties defined, showing the result output in the right pane.



In this way, you should run your filter description against a number of selected example files to make sure all is working well. If there are any problems with a query it will be reported on the output pane.

Deploying the filter

If all tests passed successfully you are now ready to enter the final state, filter deployment. From the Action menu select “Write IFilter Configuration”.



Within the last step, the filter configuration is written to the configuration file, the registry updated and the indexing service stopped and restarted. At this point, full-text queries and SPS dashboard site simple search (‘contains’ and ‘freetext’) will work.

Remember that custom value-type properties must be added to Index Server's **property cache** to make them available for display after a successful search.

To add custom properties to the property cache, invoke the Index Server Microsoft Management Console (MMC) administration tool. Open the catalog and select **Properties** from the tree view. Select the property to be added and then right-click it. Then click **Properties** and check the **Cached** box. Set the suitable data type. Save the property cache changes by right-clicking the **Properties** item in the tree pane and then clicking **commit**.

After the property is added to the schema of the property cache, each document is given a null value for the property. Documents must be re-indexed so that the values from each document are written to the property cache, since cache values are updated when a document is indexed.

To re-scan a directory, use the Index Server MMC administration tool, select the directory containing your documents, right-click it, and force a full rescan of the files. Once the index is up-to-date again, the meta property will be available in the property cache.

For additional details about the process of adding a property to the property cache, please refer to the Index Server documentation.

Query Basics

The underlying foundation for all property-queries is XQuery. The language is currently being developed by the W3C XML Query Working Group and has working draft status (as of Dec. 2002, see References for details). Even though the current language definition is quite huge based on functional principles and contains at least 7 types of expressions, there is a simple to understand **core principle** behind all the complexities. It is possible to write really simple constructs which, as you will see, satisfies all your needs for querying property data.

The core of the language is based on the FLWR (pronounced "flower") expression, and is very similar to the **SELECT-FROM-WHERE** construction in SQL.

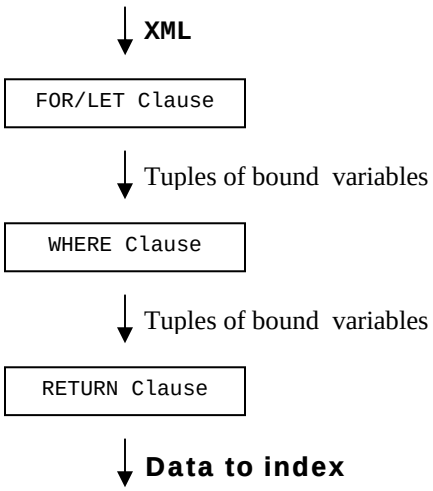
1.) A FLWR expression consists of:

- **FOR**-clause: binds one or more variables ($X…) to a sequence of nodes returned by another expression (usually a path expression, see below) and iterates over the nodes. The variable therefore represents an array of bound nodes.

- **LET**-clause: also binds one or more nodes but without iterating. A single sequence of nodes is therefore bound to the variable.

- **WHERE**-clause: contains one or more predicates that filters or limits the set of nodes as generated by the FOR/LET-clauses.

- **RETURN**-clause: generates the output of the FLWR expression. The RETURN-clause usually contains the references to variables and is executed once for each bound node-reference that was returned by the FOR/LET/WHERE-clauses.



The input to the XQuery expression consists of one or more XML **documents** to index. The result of the FOR and LET clauses is an ordered list of tuples, each containing a value for each of the bound variables. The value of a variable bound by a FOR clause is an array of nodes and its descendants. The value of a variable bound by a LET clause is a (possibly empty) single set of nodes.

The `RETURN` clause is executed for each surviving tuple, generating output nodes from the values of the bound variables. The node(s) generated by the `RETURN` clause represent either a single property value or the linearized stream of text content.

The `FOR` and `LET` clauses work together to generate tuples of variable bindings. Unlike a `FOR` clause, however, a `LET` clause does not affect the number of tuples that are generated. Each `LET` clause binds its variable to exactly one.

- If a query contains a LET clause but no FOR clause, exactly one tuple of variable bindings is generated.
- If there are more than on FOR clauses a Cartesian product of all tuples is formed.
- The `WHERE` clause serves as a filter that discards some of the tuples and retains others.

The result of the FOR/LET clause can be thought of as being equivalent to the rows and columns of a relational table where each column represents a bound variable.

The data model that XQuery uses is based on that of XPath (see References) and defines each XML document as a tree of nodes. Therefore XPath is heavily used in XQuery to select sub trees out of a larger xml tree just as it is used as the path selection language for XSLT. XQuery uses abbreviated XPath expressions.

2.) Path expressions

The second important construct are path expressions. The syntax is similar to the abbreviated syntax of XPath, the XML standard for specifying "paths" in an XML document. For example:

Find all titles of chapters in document books.xml:

```
document("books.xml")//chapter/title
```

Find all books in document bib.xml published by Addison-Wesley after 1991:

```
document(bib.xml")//book[publisher = "WROX" AND @year > "1991"]
```

In general, an XPath expression evaluates to a set of nodes. The FOR clause generates an ordered list of tuples, each containing a value for each of the bound variables. A tuple is generated for each possible way of binding the list of variables to nodes that **satisfy their respective XPath expressions**. When a node is bound to a variable, its descendant nodes are carried along with it.

XPath path expressions may contain wildcards:

```
document("books.xml")/books/*/title
document("books.xml")/books/*@isbn
```

The following example returns the title of all books published by Addison-Wesley:

```
FOR $X IN DISTINCT(document("bib.xml")/book/title)
FOR $Y IN document("bib.xml")/book[title = $X]
      WHERE $Y/publisher = 'Addison-Wesley'
RETURN $X/text()
```

Although the XQuery draft specifies more constructs (element constructors, conditional expressions …) for the sake of simplicity and usability, XML IFilter property-queries are restricted to FLWR and path expressions, with some SQL stylish enhancements to aid in query formulation as described below.

XPath

XPath is supported entirely and the work horse for all XQuery queries to select specific elements or sub-trees out of the whole xml tree. For more details on XPath please see the tutorials and specifications found on the website of W3C (see Reference). For example to index the whole content of an xml document simple write:

```
FOR $X IN FILE//* RETURN $X/text()
```

FILE keyword

In XQuery, you specify the document to be queried within the function "document()" given the file name and path as argument.

For XML IFilter queries you must use the special keyword **FILE** instead of document() which acts as a placeholder for the filename to load. The QLXFilter.dll runtime replaces all occurrences of FILE with the physical document name during the document load and enumeration process while scanning the xml documents.

WHERE clause

In the WHERE clause, predicates may be combined using parentheses, AND, OR, and NOT. Predicates are based on **XPath expressions** that contain the variables bound in the FOR and LET clauses. Comparing against values returned by sub-queries is possible too.

Examples:

```
... WHERE $X/last/text() = ‘abc’ AND $X/price/number() = 99

... WHERE $X/book[@isbn = ’12-333-456’]/price/number() = 99

... WHERE $X/pubdate/date() = ‘1994-12-03’

... WHERE $X/price/number() = ( For … Return $X/price/number() )
```

Joins are possible too:

```
... WHERE $X/last/text() = $Y/last/text()
```

The semantics of comparisons is the same as in XPath. For example, consider the comparison $X/last = "abc". In general, an XPath expression such as $X/last evaluates to a **set** of nodes. The comparison therefore is considered to be True if at least **one** of the nodes returned by $X/last has a string-value equal to "abc".

To specify the type of the bound variable in the comparison, use one of the data type modifiers added to the end of the bound variable separated by a “/”. (see RETURN clause section for details)

Modifier text() can be omitted in the WHERE clause as shown below.

```
WHERE $X/last/text() = ‘abc’
WHERE $X/last = ‘abc’
```

Remember that variables bound in a FOR clause are bound to individual nodes (with their descendants), but variables bound in a LET clause are bound to ordered sets of nodes (with their descendants). In the WHERE clause, appropriate predicates must be used with each type of variable. For example, in the following query, $book is bound to a **set** of books (by using LET), and the WHERE clause appropriately applies a count() function to count the number of books in the **set**. The query returns publishers who have published more than 100 books.

```
FOR $pub IN DISTINCT TX//publisher
LET $book := TX//book[pubinfo/publisher = $pub]
WHERE count($book) > 100
RETURN $pub/text()
```

If we require to add an additional condition on books, such as "find publishers who published more than 100 books in 2002", this condition could **not** be added to the WHERE clause, since the WHERE clause has access only to **sets** of books, not to individual books. The proper place to add such a condition would be in the XPath expression that defines $book, as follows:

```
FOR $pub IN DISTINCT TX//publisher
LET $book := TX//book[pubinfo/publisher=$pub AND
pubinfo/year="2002"]
WHERE count($book) > 100
RETURN $pub/text()
```

The WHERE clause may also use several operators taken from SQL. These operators will be illustrated below:

! Note that this is an extension implemented by QuiLogic and not part of the XQuery draft.

- [NOT] LIKE
- [NOT] BETWEEN
- [NOT] IN
- Sub-Query
- ALL, ANY, SOME, EXISTS

Above operators (except for Subquery, All … Exists) may also be used in XPath expressions like: `$X/book[@isbn IN ('554-0772-03','776-1299-01')]/title`

Examples of SQL stylish operators:

```
WHERE $X/last/text() IN ('abc', 'def', 'xyz')

WHERE $X/book[@isbn = '12-333-456']/price/number() IN (XQuery)

WHERE $X/last[3]/text() LIKE 'abc%'

WHERE $X/pubdate/date() BETWEEN '1994-12-03' AND '2002-01-01'

WHERE $X/last/number() = [ANY,ALL,SOME] ( XQuery )

WHERE EXISTS ( XQuery )
```

DISTINCT

Distinct serves the same purpose as found in SQL:

```
FOR $X IN DISTINCT(document('bib.xml')/book/title)…
```

The `DISTINCT` keyword can be applied independently to each expression in a `FOR/LET, WHERE and RETURN` clause, serving to eliminate duplicate values from the node-sets returned by the expression. Equality is defined by equality of value rather than by identity.

When `DISTINCT` is specified and several candidate nodes of equal value are available for binding, SQL/XML-IMDB does not specify which of the candidate nodes is bound to the variable.

Xml Elements having a content value of `NULL` are ignored by `DISTINCT` with the exception when adding the data type specify `/name()` to the variable.

Counts only distinct title elements having a title

```
Count(Distinct(document('bib.xml')/book/title))
```

Counts ALL distinct child elements below book regardless of having a value or not (null)

```
Count(Distinct(document('bib.xml')/book/*/name()))
```

Counts distinct numeric values of child elements (excluding null values)

```
Count(Distinct(document('bib.xml')/book/*/number()))
```

Aggregate Functions

A LET clause is often used to bind a variable to a set of values that is used as the argument of some aggregate function such as avg(). For example, the following query returns the average price of all the books in the table TX:

```
LET $b :=  TX//book/price
RETURN
   <avgprice> {avg($b)} </avgprice>
```

Aggregate functions can be applied in `LET, WHERE and RETURN` clauses. For example the above query could be rewritten as:

```
LET $b := avg(TX//book/price/real())
RETURN
   <avgprice> $b </avgprice>
```

Use of an aggregate function in the WHERE clause:

```
FOR $pub IN DISTINCT TX//publisher
LET $b :=  $pub//book/price
WHERE avg($b) < 100
RETURN
   <publisher> $pup/text() </publisher>
   <avgprice> {avg($b)} </avgprice>
```

Available aggregate functions are:

- COUNT
- SUM
- AVG
- MAX
- MIN

Aggregate functions may be combined with DISTINCT.

Return clause

In the Return clause, you specify what content to return for the property value. In contrast to "official" XQuery, you can only return the content of **one** bound variable, although you can specify as many variables as you need (to meet your query requirements) in the FOR/LET/WHERE section of the query.

As an exception to the above rule you can still have more than one variable in the Return clause if those variables are used within an expression. Therefore its perfect legal to write a query like:

```
FOR $X In FILE/book, FOR $Y In FILE/review
      WHERE $X/@ISBN = $Y/@ISBN
Return { $X/Author + ' ' + $Y/Summary }
```

The above query returns in **one** result the concatenation of two bound variables. Note further the use of a join statement to "connect" (@attribute ISBN) two different sub trees of the xml document tree to create a combined result. Always Use "{ }" if you have expressions in the Return clause, but they can be omitted for simple returns.

Data types

To match the data type of the return value with the type of the property (remember there are text-type and value-type properties) use one of the data type modifiers added to the end of the variable name separated by "/".

…/text()	Text (DBTYPE_WSTR).
…/number()	Integer (DBTYPE_I4).
…/real()	Double (DBTYPE_R8)
…/datetime()	Date/Time(VT_FILETIME)
…/bool()	Bool (DBTYPE_BOOL)

If you omit the data type modifier, type text is assumed and the property returns both the element name **and** content: `<author>Stanislav Lem</author>`

To return an element name only use: **…/name()** which returns only the name of an xml-element as of data type text.

For example, to return the text content of an element enclosed between the tag-name, use the following example expression:

```
… RETURN { $X/author/name() + ' ' + $X/author/text() + ' ' + $X/author/name() }
```

which returns a result like: "author Stanislav Lem author"

Important Note! There is a small but important difference between returning text-type and value-type properties. For example, if you query a bibliographic xml document for the book titles contained,

```
RETURN $X/title/text()
```

A text-type property returns **all** titles found as a concatenated stream of words:

```
TCP/IP Illustrated Data on the Web Advanced Programming in the …
```

Whereas a value type property return the **first** occurrence of all titles found:

```
TCP/IP Illustrated
```

This has to do with, how the Indexing Service works in the case of value type properties (see the Indexing Service documentation for more details).

Examples

```
... RETURN $X/last/text()

... RETURN $X/price/number()

... RETURN $X/book[@isbn = '12-333-456']/price/number()

... RETURN { $X/last/text() + ' ' + $X/first/text() }

... RETURN { $X/price/real() + 12.99 }

... RETURN { SUM($X/price/real() }

... RETURN { COUNT($X/book) }

... RETURN $X
```

Availability

XML IFilter is available as:

- 1 Machine license.
- 4 Machine license.
- Enterprise edition for an arbitrary number of machines.

The Enterprise edition contains the source code of the IFilter implementation either as Visual Studio 6.0 c++ or as VC8.0 c++ project. This enables any developer in the Enterprise to make custom modifications for the IFilter dll to satisfy special requirements not available in the original product.

To order please visit www.quilogic.cc

QuiLogic, Inc. is an IT company headquartered in central EU, Austria. Founded in 1995, today QuiLogic creates innovative products and offers exceptional expertise in all sort of data management projects, high performance demanding applications and xml based solutions.

References

XQuery:
www.w3c.org/xml/query.html

XPath:
www.w3c.org/TR/xpath

SQL/XML-IMDB:
www.quilogic.cc/whitep.pdf