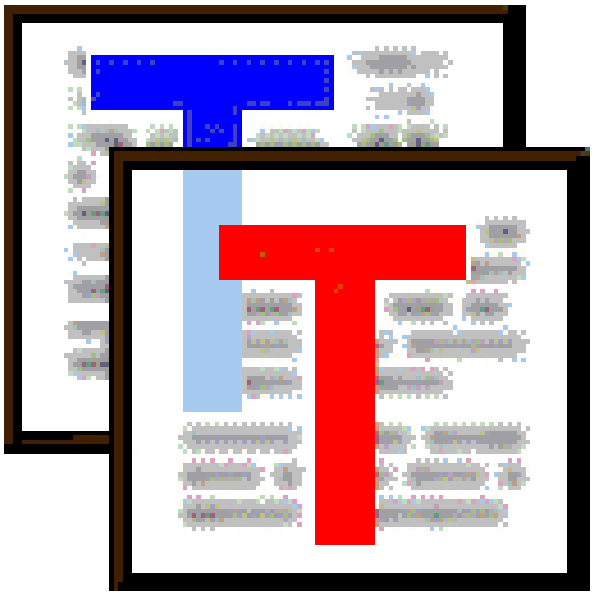
IMP Filter



1 Introduction

The IMP filter is plugin for the anti-spam software Spamihilator:

http://www.spamihilator.com

Short description

With the IMP filter (individual mail parsing filter) e-mails are analyzed to judge whether they is spam or not. The criteria of this classification aren't fixed. The user has to create his own individual TextTransformer project for this purpose.
For the construction of such projects the free version of the TextTransformer program suffices. Texts can be analyzed almost arbitrarily exactly with such projects. The effort for the making such projects is only worthwhile perhaps, though, when the statistical results of the word filters are too unsure and a certain result for special cases is needed.

2 De-/Installation

The **installation** of the IMP filter presupposes the installation of the Spamihilator and is made by the IMPInstall.exe.

The **deinstallation** should be carried out via the system control/software of the operating system. The Spamihilator must be closed before since otherwise the loaded dll's aren't removed.

The paths are automatically adjusted to paths of the Spamihilator.

After installation of the IMP filter, at first it does nothing. The user has to select a TextTransformer project before the filter starts working.

The position in the list of Spamihilator plugins should be adjusted to the project.

2.1 Paths and files

The paths are adapted to those of the Spamihilator automatically.

Program directory

The real plugin, the *impfilter.dll,* is installed into the sub-directory plugins of the program folder of the Spamihilator. Normally:

```
C:\Programme\Spamihilator\plugins
```

Application data directory

The installer writes the project examples, this help and a language file into a subdirectory of the program directory at first
Depending on operating system and a way of the Spamihilator installation they are copied from there at the first call of the plugin into an application data directory of the respective user. The name of such application data directory is formed by appending a version name to the name "impfilter". E.g. for the version 0.7.0 the resulting name is:

```
...\impfilter070
```

These folders have to be deleted by hand at the deinstallation of the IMP filter.

You can find this directory at one of the following places:

1. If Spamihilator has been installed with the option “Separate settings per user account” (recommended, default method):

   I. Windows 2000/XP:

   ```
   C:\Documents and Settings\Username\Application Data\Spamihilator\plugins
   \impfilterXXX
   ```

   II. Windows Vista:

   ```
   C:\Users\Username\AppData\Roaming\Spamihilator\plugins\impfilterXXX
   ```

2. If Spamihilator has been installed with the option “Shared settings for all users” (old method) the data aren't copied into an extra folder:

```
C:\Program Files\Spamihilator\plugins\impfilter
```

You have to save changed projects of this folder by hand, before you install an update of the IMP filter.

System directory

The parser interpreter *imp_engine.dl* and some other dll's, which are commonly used by it and the TextTransformer are installed in the system folder

```
C:\WINNT\system32
```

Experts might want to know the complete list:

```
imp_engine.dll
imp_cppitp.dll
TTXercesLib.dll
boost_regex-bcb-mt-1_34_1.dll
stlpmt45.dll
cc3260mt.dll
rtl60.bpl
vcl60.bpl
```

2.2 Priority

It is recommended to put the IMP filter in second place just after the newsletter in the list of filters if you have designed your individual mail parser in the way described below.

Here the Spamihilator on-line help decribes how to make this setting:

http://wiki.spamihilator.com/doku.php?id=en:configpriorities

Of course it depends on the kind of the project where the IMP filter should be placed in the list of the Spamihilator filters. However, it is the special capability of the IMP filter that you can make your individual mail parser in such manner, that it can decide surely for certain mails whether they are spam or not spam. It therefore makes sense to put the IMP filter in the list relatively far above. Desired mails should not be eliminated by other filters and unwanted mails not be excluded from the further analysis as not spam, before the IMP filter can test them at all. In cases where the IMP filter cannot decide surely whether it is spam or not spam, the mail should be passed to the other filters however.

3 Known problems

The IMP filter isn't consistent with the **Blacklist filter** well. The Spamihilator crashes if you reload the plugings in the spamihilator settings dialog and then click on OK. Otherwise there aren't any known problems.
The Blacklist filter is anyway regarded as obsolete now.

If you try with a 0.7 version of the IMP filter to load a project which was made with a newer version of the TextTransformer than 1.6.1, the Spamihilator crashs

4 TextTransformer

The IMP filter needs a TextTransformer project adapted to your individual needs. There are some examples in the application data folder which can be used as a base for developments of one's own project. The installation of the TextTransformer program is required to modify projects. It can be downloaded from the following link:

http://www.texttransformer.com/Downloads_en.html

The free version of the TextTransformer usually suffices for Spamihilator projects..

5 Parsing texts

With the IMP filter the texts of e-mails are analyzed, that means, the texts are parsed. This shall be illustrated at the example of a typical mail structure. Depending on their structure the texts are classified into spam or not spam. If the analysis fails, the parser error must be treated.

5.1 Word explanations

Mail text

There are different ways how e-mails can be constructed. Common for all ways is, that the complete mail starts with a **header**, which includes the subject of the mail the sender and other information. The real data of the e-mail following the header can be of different type:

- plain text
- HTML formatted text
- Images, audio data, video data etc. in binary form

The header also can include some information, which describe the further construction of the current mail and the kind of the data.

- This information is missing in the simplest case and the header is followed by a blank line and finally the text with the real message.
- Complex e-mails are **MIME**-encoded (Multipurpose Internet Mail Extensions). Such mails can consist of several areas which consist of header lines and other data again.

Parsing

With the IMP filter the texts of e-mails are analyzed, i.e. the texts are taken to pieces in accordance with their structure or syntax. **Parsing** is the technical term for this: the texts are parsed. A program which parses texts is a **parser**. The parsers can be made by the user with the TextTransformer. This program is a **parser generator IDE** (integrated Development environment). TextTransformer projects contain the specifications for parsers.

5.2 A typical structure of a mail text

With the IMP filter text is parsed. This means that texts are analyzed according to their structure and their components. At a simple example the principle shall be demonstrated briefly. The header of the e-mail and possible sub-structures with binary data shall be ignored in this introducing page.. Here the pure text shall be analyzed as it is shown on an e-mail program.

This text can be analyzed in different ways. In the simplest case as a simple word list. (A list is a structure too.) A more complex structure, however, is suggested at the following typical mail.

```
---------

    Dear  Heinz,

    blah  blah  blah

    Cordially  yours

    Fatty

-----------
```

A text follows on the salutation and on this a greeting follows. And the salutation in turn is structured in itself and has a named component among others. If this name could be found, then it can be used well as a criterion for a not spam mail. If the addressee of the above mail is actually Heinz, then it in all probability the mail isn't spam, but it would be spam, if e.g. the salutation would be "Dear Fatty". This would be an advertisement for a slimming product, presumably.
In this case a word filter doesn't suffice for the classification as spam not. Heinz could have a friend whose nickname is Fatty. For the distinction it is therefore necessary to know in which positions the name appears: Fatty in the salutation wouldn't be spam, but not Fatty as signatory.

With the TextTransformer you can analyze such a mail text. At first the structure of the mail above is described a little more abstractly:

```
mail ::= salutation text greeting
```

Salutation, text and greeting are text components which are structured themselves in a way of their own. A definition for the salutation could be:

```
salutation ::= "Dear" "Heinz"
```

I.e. the words "Dear" and "Heinz" succeed one another in a salutation.
However, this doesn't characterize a salutation sufficiently. "Hello Heinz" would be a correct salutation too. Therefore the above rule (technical term: production) could be generalized to:

```
salutation ::= ("Dear" | "Hello") "Heinz"
```

I.e. the word "Heinz" follows in a salutation on one of the words "Dear" or "Hello." This certainly still doesn't suffice, however, it illustrates the principle. It works very similarly as the regular expressions which might be known to many users of the Spamihilator. In a TextTransformer project rules can describe the construction of a text quite similarly like regular expressions describe the construction of words. E.g. the repetition operators well-known from the regular expressions can be used for the text rule:

```
text ::= (WORD+ PUNCTUATION)*
```

The tokens WORD and PUNCTUATION can be described as "genuine" regular expressions.

5.2.1 Remarks to the syntax

The regular expressions in the TextTransformer are a little bit different from those in the Spamihilator. The TextTransformer expressions are looking for the longest possible match in the text and the

Spamihilator expressions are looking for the first possible match. Also the analogy of the syntax of the TextTransformer productions with the syntax of the regular expressions is limited.For the details you have to study the help of the TextTransformer.. Here only the hint: the productions of the TextTransformer must obey to the so-called LL(1) principle at first, though this principle is extended by the possibility of look-aheads.

5.3 Spam or not spam

It would be ideal if one could describe the structure of the e-mails so completely that the amount of the mails which cannot be parsed would be identical with the amount of the spam mails. This ideal case might be rare, but it is not impossible. E.g. company mail could be organized such, that it must obey an exactly defined structure. The IMP filter will usually work with approximations, however and only a certain subset of the mails is recognized for certain as spam or not spam. So the other filters of the Spamihilator are needed furthermore.

TextTransformer projects usually are made to produce target texts from source texts. The target text for the Spamihilator is simply "1" or "0"or" -1" for non-spam, an indifferent text and spam.

Non-spam	"1"
Spam	"-1"
indifferent	"0"

The returned text is produced in the so-called semantic actions. The definition of the salutation above is therefore supplemented to:

```
Anrede  ::=
("Lieber"  |  "Hallo")
(
     "Heinz" {{iResult = 1; }}
   | WORT     {{iResult = -1; }}
)
```

"iResult" is a variable which has to be declared before. This is demonstrated in the examples.

5.4 Parser errors

Unfortunately, it isn't so easy to design a "general mail parser". If you try it, you will encounter lots of surprises. E.g. a mail of the following kind:

```
---------

    (English version in the 2nd part of this mail.)
    ====================================================


    Hallo  My_Name,

    blah  blah  blah
```

```
    Herzliche  Grüße

    Dein  Fettie


    ...

----------
```

With the production above:

```
    mail ::= salutation text greeting
```

This mail cannot be parsed, because there is another text in front of the salutation. You can adjust the filter for such cases.

5.5 Parameter

The IMP filter passes two parameters to the loaded TextTransformer project.

1. the log file can be accessed in the project with the function *ConfigParam*.
2. a name for the current mail can be read in the project with the function *ExtraParam*. The name is formed from the word "Message" on which the start date of the Spamihilator and a number are for the mail are appended. E.g.: Message_09-01-15-14-51-05_6

Warning: The parameter might become more complex in future updates of the IMP filter. This can mean that you then have to change your project, if it uses them.

By means of these parameters, e.g. it is possible to write additional information into the logging file:

```
{{
if(!ConfigParam().empty())
{
   RedirectOutput(ConfigParam(), true);   // append to log file
  m_bLog = true;
  out << ExtraParam() << endl;  // Mail name
}
else
  m_bLog = false;
}}

... // Text parsen

{{
if(m_bLog)
{
  if(m_iResult == 0)
      out << indent << " indifferent" << endl;
   ResetOutput();
}
}}
```

A further example of the use of *ExtraParam* is the storage of the mail texts.

6 Settings

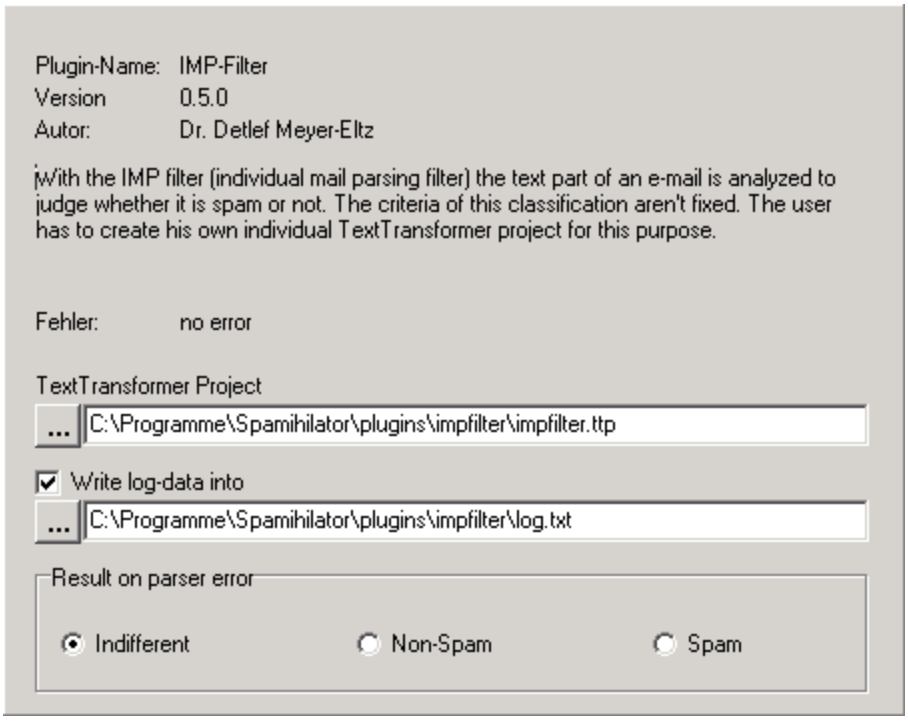
To open the Settings dialog, right-click on the icon in the tray area and choose "Settings…"

http://wiki.spamihilator.com/doku.php?id=en:mainmenu

There you can navigate to the Plugins

http://wiki.spamihilator.com/doku.php?id=en:configplugins

There you can select IPM Filter.
By the "Configure..."-button a settings dialog for the IMP-Filter is opened.

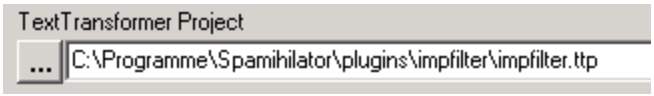


There you can set

the project-file
the result on parser error
the writing of a log-file

6.1 Project file

After installation of the IMP filter, at first it does nothing. The user has to select a TextTransformer project before the filter starts working.There are example projects in the application data folder.

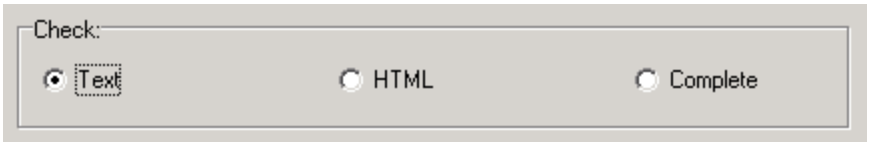


If you have changed the loaded project, you have to restart of the Spamihilator or to reload the impfilter.dll, before the changes has an effect.

6.2 Check

The Spamihilator offers not only the possibility of analyzing the complete text of an e-mail, inclusive of the header data, but Spamihilator also offers the possibility of restricting the analysis to the already extracted readable texts. These are provided by the Spamihilator in double form: either text with HTML formattings, if there are some, or the plain text with all HTML tags removed.

The IMP filter offers its users the same possibilities:



1. Text

This option is activated per default and means that the plain readable message text of an e-mail is analyzed with the project. This is the text, that the e-mail program is showing to you. The links to external internet pages, which frequently exist in spam mails, often are not visible in these pure texts.

2. HTML

If this option is activated, the used project must be able to manage HTML tags. Unfortunately, it isn't guaranteed - just at spam - that the HTML text is well-formed. In addition, the text provided by the Spamihilator often consists of a mixture of plain text and HTML formatted text.

3. Complete

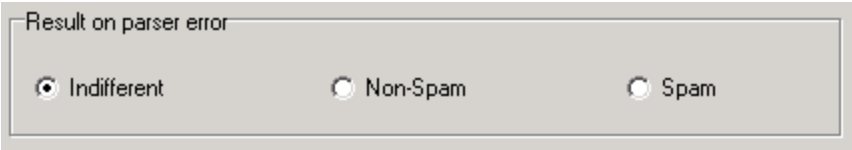
All information of the e-mail are at the disposal to the parser with this option, because it has to process the complete mail with headers sub-structures and binary data.

After a call of the settings dialog a value is written into the Spamihilator.ini:

[impfilter.dll]
…
Checkr=Text

6.3 Result on parser error

Three options can be chosen how the filter shall proceed if a mail cannot be parsed.



1. Indifferent

Per default mails which cannot be parsed are treated indifferently and therefore passed on to the other filters. This is the recommended option for the IMP filter with a high priority.

2. Non-Spam

The classification of the mails as non spam makes sure that they aren't lost. This can make sense particularly if you are experimenting with a new TextTransformer project and spam mails are needed as examples for testing the project.

3. Spam

The mail is classified as spam. This is risky because mails get lost easily if classified as spam. Even experienced users of the TextTransformer frequently miss alternatives which lead to parser errors. This option isn't recommended therefore.

After a call of the settings dialog a value is written into the Spamihilator.ini:

Non-Spam	1
Spam	-1
indifferent	0

[impfilter.dll]
…
OnParseError=0

6.4 Writing a log-file

You can choose with a check box whether a log file shall be written or not. When the box is checked, new logging information is appended to the data which might exist already.
The log information which per default is written by the IMP filter can be enriched with information from the TextTransformer project.

(This code snippet is from *NonFree.ttp*)
After a call of the dialog a value is written into the Spamihilator.ini for the log option and the log path.

```
[impfilter.dll]
Log=1
LogPath=C=3A=5CProgramme=5CSpamihilator=5Cplugins
```

6.5 Last error message

If there was an error, you can see the last error message in the settings dialog again.

7 Example projects

There are some example projects in the application data folder. It is tried to explain the examples so far that they get understandable without more exact knowledge of the TextTransformers syntax. However, this can be successful only, if the reader studies them carefully in the order as they are given.

The example parsers are grouped according to the check options.

Plain text parsers

- Emptymail
- NonSpam-Worte
- SpamAndNonSpam
- DirtyFriends
- Formmail
- Impfilter

HTML and text parsers
Complete mail parsers

The projects can partly be modified intuitively by simple word substitutions and analogous additions. For more ambitious projects you have refer to the help to the TextTransformer.

7.1 Plain text parsers

If one of the parsers for plain texts shall be used with the IMP plugin, then the corresponding option to check texts should be set. The following examples are designed in such way, though, that they should work also with the other check options.

You can adapt the parsers for plain texts most easily to your needs, because no knowledge of the construction of a complete e-mail or of HTML is required.

Emptymail
NonSpam-Worte
SpamAndNonSpam
DirtyFriends
Formmail
Impfilter

7.1.1 Emptymail

The example project *Emptymail.ttp* in the application data folder works similarly as the Spamihilator empty mail filter. The example shall not at all replace the more flexible empty mail filter. The *Emptymail* project is made for the didactic reason only that it is very simple. The complete project consists of only two lines:

```
  SKIP {{ out << 0; }}
| {{ out << -1; }}
```

SKIP is a special token in the TextTransformer. The complete text is matched with *SKIP* up to the position, where a token matches, that follows on *SKIP*. Here nothing follows on SKIP, though, but the end of the text. Therefore the complete text is recognized with *SKIP*, if there is one. If the mail doesn't include any text the alternative behind the or '|' is executed: "{{" out <<-1 ; }}"
The brackets "{{ ... }}" enclose actions. The action "out << -1" means, that the value -1 is output and this value represents spam. The value 0, however, means that the filter cannot decide, whether the text is spam or not.

7.1.2 NonSpamWords

The project *Nonspamwords.ttp* is suitable to carry out a first test of the IMP filter after installation. The filter should have a high priority with this project.

There are a lot of Spamihilator filters which are used to filter spam from the received e-mails. The project *Nonspamwords.ttp* on the other hand has a positive strategy. E-mails, which contain certain keywords, are supposed to be not spam and shall be protected from the following negative filtration. Such keywords

can be the own name or the one's own company name or product names or also suitable expression of one's own special interest areas.

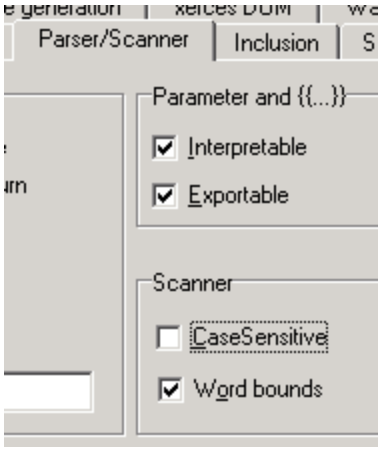
```
{{
int iResult = 0;
}}
SKIP?
(
   (
         "Spamihilator"
      |  "TextTransformer"
      | "tetra"
   )  {{ iResult = 1; }}
   SKIP?
)?
{{
out << iResult;
}}
```

The result isn't output directly, but stored intermediately into the variable "iResultat". Only at the end of the program the value is written with the instruction "out << iResultat;". The instruction "int iResultat = 0;" is a declaration (creation) of the variables and the value 0 standing for indifference is assigned to the variable at once. This value is changed only, if the outer parenthesis (...)? is passed. The question mark means - as in the case of the regular expressions - that the expression is optional. So the occurrence of one of the words which are divided by the sign '|' is optional.

```
(
      "Spamihilator"
   |  "TextTransformer"
   | "tetra"
)  {{ iResult = 1; }}
SKIP?
```

If one of the words is found in the text, the value 1 representing not spam is assigned to the variable *iResult*. The probably following text section is then skipped with the second SKIP until the end. If none of the non spam words occur in the text, the complete text is recognized with the first SKIP.

To cover more possible notations of the non spam words, the case sensitivity was switched off in the TextTransformer in the project options.



By two small modifications this project could be combined with the *Emptymail* project:

```
{{
int iResult = 0;
}}
(
    SKIP
    (
       (
             "Spamihilator"
          |  "TextTransformer"
          |  "tetra"
       )  {{ iResult = 1; }}
    )?
  | {{ iResult = -1; }}
)
{{
out << iResult;
}}
```

In the "empty" alternative "| {{ iResult = -1; }}" an action is executed only, but no text is consumed. The empty alternative makes the surrounding bracket optional without '?' .

7.1.3 SpamAndNonSpam

The previous project shall be extended to the project *SpamAndNonSpam.ttp* so, that also spam words are recognized. However, these should be only a few well-chosen words which suggest spam with a very high probability, because of the high priority of the IMP filter.

To make the project more clear, at first two additional rules (productions) are created. The first *NonSpam* covers the not spam words.

```
   "Spamihilator"
|  "TextTransformer"
|  "tetra"
```

The second additional rule may be called *Spam* and contains some spam words:

```
   "Viagra"
|  "Casino"
|  "Rolex"
|  "Watch"
|  "Watches"
```

In the start rule of *SpamAndNonSpam* a loop is executed until the complete text is processed. The star '*' behind the bracket symbolizes the loop.

```
{{
int iResult = 0;
}}
(
    SKIP
  | NonSpam  {{ iResult = 1; }}
  | Spam      {{ if(iResult == 0) iResult = -1; }}
)*
{{
out << iResult;
}}
```

The use of the loop is a little trick. The alternatives to SKIP can occur in front of it and follow on it. In the start rule SKIP either jumps to the next spam word or to the next non spam word or to the end of the text. If a word was found, a value is assigned to the result.
However, the mail is classified as spam only when it doesn't contain a not spam word. This is guaranteed by the instruction:

```
if(iResult == 0) iResult = -1;
```

I.e. the result is set to -1 only, when it is still indifferent. If a not spam word follows, is the result put to 1 and totally the mail is classified as not spam. So the IMP filter can be used furthermore with a high priority.

7.1.4 DirtyFriends

The project *DirtyFriends.ttp* permits special friends to express dirty fantasies, if they sign the mail by name.
In the result this project is quite similar to the *SpamAndNotSpam* project. Instead of the *NonSpam* production there is a *Friends* production. The main rule is built in such manner, that the parser immediately skips to the end as soon as the result is certain. It is the disadvantage, though, that the rule looks confused.

```
{{
int iResult = 0;
}}
(
```

```
  SKIP?
  (
      Friends[iResult] SKIP?
    | Spam
      (
        SKIP?
        (
            Friends[iResult] SKIP?
          | {{ iResult = -1; }}
        )
      )
  )?
)
{{
out << iResult;
}}
```

The Friends production contains a list of the friends:

```
(
    "Peter"
  | "Paul"
  | "Mary"
)
{{
xiResult = 1;
}}
```

It is new that the F*riends* production is called with *iResult* as a parameter:

```
Friends[iResult]
```

.
The parameter must be declared in the TextTransformer in the *Friends* production:

With the parameter the value is "fetched" from the Friends production: `"xiResult = 1;"`

Notice the extensive use of '?' in the start rule. That's necessary to make sure that every text can be parsed. Parser errors cannot appear then, even if e.g. the whole mail only consists of a spam word.

7.1.5 Formmail

The example project *Formmail.ttp* demonstrates how mails can be classified obviously as not spam if they have a certain structure. Since the project doesn't reject any mail as spam, again it should be put near the top in the list of the filters.

Only an imaginary company mail is accepted as non-spam. E.g. following mail.

```
Sender:  Fatty
Subject:  diet
Message:  don't  like  it!
```

The project consists of the productions:

```
Sender  ::= SKIP EOL
Subject:  "Subject:" SKIP EOL
Message  ::=  "Message:" SKIP?

Formmail  ::=  Sender  Subject  Message   {{ out << 1; }}
```

EOL is a regular expression which represents a line ending. It is defined in the TextTransformer on the token page by: \r?\n.
Although blanks and line breaks are ignored per default project options of the TextTransformer, *EOL* is recognized nevertheless as a successor of *SKIP*.

Notice, that this project always returns the value 1 (for not spam). Nevertheless there is no danger that spam is accepted, because it is sure that a mail which has not the specified structure, can't be parsed until its end at all. In this case the value for parser errors is used in the Spamihilator instead. This value necessarily should therefore be set to "indifferent" for this project.

Exercise:
Mails without senders and subject also aren't parsed. How can you change this?

7.1.6 NonFree

NonFree.ttp is the real project which I developed before the first publication of the IMP filter to adapt it to my special needs. Only my name is changed in the project. All the previous projects are usable with the free version of the TextTransformer. This doesn't apply to *NonFree.ttp*. It shall be nevertheless presented briefly here as suggestion and "quarry".

The central rule is the *text* production. It consists essentially of a loop with alternative regular expressions for words, punctuation marks and the other printable characters.

```
(
    WORD
  | PUNCTUATION
  | SPECIAL
  | NBSP      {{ SetIsSpam(" : NBSP"); }} // protected blank
)*

WORD   ::=   [[:alnum:]Æ-Ïæ-ï€ÀÁÂÃÄÒÓÔÕÙÚÛÜßàáâãäòóôõùúûü]+
PUNKTUATION  ::=  [[:punct:]]+
SPECIAL    ::=    [^[:alnum:][:punct:][:space:]Æ-Ïæ-ï€ÀÁÂÃÄÒÓÔÕÙÚÛÜßàáâãäòóôõùúûü]+
NBSP  ::=  \xA0
```

With this loop all texts can be parsed completely because the expressions contain all ANSI characters. (Blanks are skipped automatically in accordance with the project options.)

This loop is extended by additional alternatives which are candidates for beginnings of is spam phrases. E.g.:

```
(
    WORD
  | PUNCTUATION
  | SPECIAL
  | NBSP      {{ SetIsSpam("NBSP"); }} // protected blank
  | pricelist_if
)*
```

Pricelist_if :

```
IF( pricelist() )
  pricelist          {{ SetIsSpam("price list");  }}
ELSE
  price
END
```

Pricelist_if was chosen here because it uses an interesting feature of the TextTransformer, though it isn't available in the free version. With the lines:

```
IF( pricelist() )
  pricelist
```

the parser is instructed to look ahead in the text. The text is only parsed as a price list and the spam attribute is set, if a whole list of articles and prices actually follows. Otherwise the above loop is continued at the current position. The look-ahead has the advantage that not all possible alternatives of any recognized token have to be taken into consideration to prevent an early abort of the parser.

7.2 HTML and text parsers

If one of the parsers for HTML and texts shall be used with the IMP plugin, then the corresponding option to check HTML should be set.

Though the HTML option should be set, it is not guaranteed that spamihilator delivers a text, which contains HTML-tags. Parsers for this option must therefore be able to cope both with HTML code and with plain text or a mixture of them.

HTMLText
AllLinksAreSpam

7.2.1 HTMLText

The HTMLText-parser copes both with HTML code and with plain text or a mixture of them. With one exception, it doesn't presuppose that HTML code is well-formed, either..At such an assumption the

parser would frequently fail. But, if the token "<!DOCTYPE" is found, it is assumed that this is the beginning of a well-formed HTML code section.
As long as well-formed HTML cannot be assumed, unfortunately, it isn't possible with the free version of the TextTransformer to distinguish whether '<' or '>' are the beginning or end of a HTML-tag or the less or greater sign. So it is not known, whether the parser is inside or outside of a tag. (With the standard version of TextTransformer a look-ahead could be used, to make the decision.)

To make a spam filter of this project, it must be enlarged by test functions of one's own. For the call of these functions the *TextToCheck* production is made:

```
  WORD
| STRING
| SPECIAL
| Link
```

The important text components are here together: words, quotations, special characters, e-mail addresses and links.

7.2.2 AllLinksAreSpam

AllLinksAreSpam is based on the *HTMLText*-project. Syntactically they are identical. But a simple semantic action was inserted which classifies the mail as spam as soon as a link is found in it. This makes sense in the respect that almost every spam-mail is a vehicle for links. If one likes to allow links only in e-mails whose addresses are in the friend list, one has an effective spam-filter with *AllLinksAreSpam*. In addition, this project demonstrates the advantage of the HTML option over the text option: pure texts often doesn't contain all links.

The action is executed in the *Link* production:

```
  NORMAL_LINK                {{m_iResult = -1; }}
|  "http://www.mydomain.com"
| "mailto:"?
  (
      EMAIL                  {{m_iResult = -1; }}
    | "myname@mydomain.com"
  )
```

NORMAL_LINK is a regular expression which describes the pattern of most links.

```
NORMAL_LINK  ::=
(http://|ftp://)?[^\r\n\t   <>"@]+(\.[^\r\n\t   <>"@]+)+
```

EMAIL is a regular expression which describes the pattern of most e-mail-addresses:

```
EMAIL  ::=
[\w\.-]+ \// local part
@ \
([\w-]+\.)+ \ // sub domains
[a-zA-Z]{2,4} // top level domain
```

The addresses of one's own are a special case. Sometimes they are copied by spammers into the mail.

But it also can be that these addresses indicate that the mail is an answer from a sender whom you haven't included in your friend list yet.
It is a good idea to develop a regular expression which matches the own address only when it is quoted exactly in the way how you write them into your mails. E.g.: the regular expression:

```
MY_EMAIL  ::=  -+\r?\nmailto:myname@mydomain.com
```

would match the following notation:

```
----------------
mailto:myname@mydomain.com
```

The *Link* production then could be completed to:

```
  NORMAL_LINK                 {{if(m_iResult != 1) m_iResult = -1; }}
| "http://www.mydomain.com"
| "mailto:"?
  (
      EMAIL                   {{if(m_iResult != 1) m_iResult = -1; }}
    | MY_EMAIL                    {{m_iResult = 1; }}
  )
```

7.3 Complete mail parsers

If one of the complete mail parsers shall be used with the IMP plugin, then the corresponding option to check the complete e-mail should be set.

The standard specification for e-mails is called MIME (Multipurpose Internet Mail Extensions). There is a really complete MIME parser, which also parses the sub-structures of all fields in detail:

http://www.texttransformer.org/MIME_en.htm

Spam mails often are made sloppy and don't pass this parser. So it could be used as a spam filter with little effort. But this would be a sparrows-shooting with cannons perhaps.

The Simple_MIME parser is a generalized form of the MIME parser above. Simple_MIME is quite tolerant to syntax violations. Only the parts were taken from the parser above, which are necessary to break down the MIME structure and make the text sections available, separated from the headers and the binary data.

7.3.1 Simple_MIME

The name "Simple_MIME" deceives. This parser is simple only in comparison with the detailed MIME parser from which Simple_MIME was extracted. Changes on the syntactic structure of the parser. can be recommended only for experienced experts of the TextTransformer. What should, however, be possible for more users of the IMP filter is a semantic expansion of the project, so that it carries out a classification into spam or not spam in accordance with individual criteria. At first the Simple_MIME parser does nothing in this regard. It is a bare scaffolding which requires the expansion.

To program such spam tests, basic knowledge of the TextTransformer still is necessary. But then the classification criteria are unlimited, because **all information of the e-mail can be accessed** in principle.

The most important position where a spam test can be installed is the *text_element*-production:

```
    WORD
  | DIGITS
  | STRING
  | PUNCTUATION
  | SPECIAL
  | NBSP
```

WORD us the same token for the recognition of words, which already has been used by another project.

```
WORD  ::=  [[:alpha:]]+
```

E.g. the recognized content can be compared with a spam word in the following way:

```
WORD
{{
if(xState.str()  ==  "Spamword")
   m_iResult = -1;
}}
```

The *text_element*-production can easily be supplemented with other alternatives, e.g. to identify and classify links or email addresses.

The other tokens of the text-production are necessary to have a recognition alternative for all existing characters.

8 Further possibilities

Besides the pure syntax analysis of the source texts, commands of the programming language C++ can be executed in TextTransformer projects. So the data won from the analysis can further be processed and evaluated. So the projects aren't restricted to a spam analysis but the received mails can be processed automatically also in an arbitrary other way. Here only two simple suggestions:

Saving test mails
Saving sorted mails

8.1 Saving test mails

It is helpful during the developmental stage of a new project to be able to test the classification in the TextTransformer debugger step-by-step if it wasn't correct.

The project SaveTests.ttp contains code which makes, that the text of the e-mails is written into different folders depending on, whether these were indifferent mails or spam or not spam.

The project presupposes that there is a folder on the hard disk with the following structure:

```
..\impfilter\Spam
..\impfilter\NonSpam
..\impfilter\Indifferent
..\impfilter\OnError   //  for  CopyTextToDisk  in NonFree
```

Either you have to create the *impfilter* folder on disk drive C as expected from the project or you have to adapt the path in the project:

```
{{
str sTestDir = "C:\\impfilter";     <- adapt
str  sTestFile;
int iResult = 0;
}}
SpamAndNonSpam[iResult]
{{
out << 0;  // test only


switch(iResult)
{
  case 1:
   sTestDir  +=  "\\NonSpam";
  break;
  case 0:
   sTestDir  +=  "\\Indifferent";
  break;
  case -1:
   sTestDir  +=  "\\Spam";
  break;
}

sTestFile  =  append_path(sTestDir,  ExtraParam()  +  ".txt");

if(exists(sTestDir))
{
     RedirectOutput(sTestFile);
   out  << xState.lp_str();    // text covered by the last production
    //out << xState.text(0);  // not allowed in the free version of TextTransformer
    //ResetOutput();
}
}}
```

The project doesn't affect the Spamihilator filters since the value 0 for indifferent mails is returned always. But the mail is sorted into the folders as they would be classified if *iResult* would be returned instead of 0.
iResult is calculated here like in the project SpamAndNonSpam. The start rule of SpamAndNonSpam is called as a sub-rule here. This is a trick, which allows to write the complete mail text with xState.lp_str() (see help to the TextTransformer). This function may be used in the free version of the TextTransformer, in contrast to the text-function, which is used in NonFree.

As the function ***CopyTextToDisk*** similar code is used in the project *NonFree*. There mails are saved too, if a parser error occurred..

8.2 Saving sorted mails

After there already is a possibility of savng the mails sorted according to its spam character the possibility is obvious not only to use the Spamihilator for the mail filtration but also for the mail processing. E.g. the type of a company mail could be analyzed and orders, complaints, offers etc. could be saved separately in different folders.

9 Licenses

The IMP filter is subject to the same license as Spamihilator.

http://wiki.spamihilator.com

The IMP filter uses parts of the boost-libraries:

http://www.boost.org

Indirectly the IMP-Filter links xerces .

http://xerces.apache.org/xerces-c/

9.1 Spamihilator license

SPAMIHILATOR LICENSE

Version 1.0
Copyright (c) 2002-2003 Michel Krämer

Everyone is permitted to copy and distibute verbatim copies of this document,
but changing is not allowed.

1. DEFINITIONS

1.1 "License" means this document.

1.2 "You" means the licensee.

1.3 "Source code" means the preferred form of the code, that is covered by this license, including all modules it contains, any associated interface definition files and scripts used to control compilation or installation of an executable or source code.

1.4 "Compilation" means a binary or executable file or binary or executable files created by a compiler, assembler or linker or any other software that is able to process source code.

1.5 "Author" means the original copyright holder of this software.

2. COPYING AND DISTRIBUTION

This software is FREEWARE. You may distribute copies of this software but you may NOT charge for the software or the service of distrubution.
If you make copies of this program you must give the recipients all the rights you have. You must also include this license in the distribution.

Although some parts of the program's source code may be published by the Author, you are not allowed to distribute it in any way. You may read the source code to get an idea how to write new code for yourself but you may not copy any portion of the original source code to create a new compilation or verbatim compositions.

3. MODIFYING

Modifying your copy or copies of the program or any portion of it in any way is prohibited.

Although some parts of the program's source code may be published by the Author, you are not allowed to modify these parts to create a new compilation or verbatim compositions.

4. DECOMPILING

Decompiling, disassembling or any reverse engineering of the binary and executable files that are associated with this program is prohibited.

5. ACCEPTING THIS LICENSE

Distributing or copying the program is prohibited by law unless you accept this license. You are not required to do this, because you have not signed the license, but by copying or distributing the program, you declare your acceptance of this License.

6. DISCLAIMER OF WARRANTY

THIS SOFTWARE IS PROVIDED "AS IT IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THIS SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THIS PROGRAM IS WITH YOU. IF THE SOFTWARE SHOULD PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE AUTHOR OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

7. TERMINATION

This license is valid as long as you use the product. It will be terminated as soon as you violate any of its terms and conditions. In this case you have to immediately destroy, respectively delete, all copies of this product you made.

8. MISCELLANEOUS

This license represents the complete agreement between the Author and you. It overrides all other agreements, either spoken or written. It can only be changed by a written and signed contract.

If, as a consequence of a court judgment or allegation of patent violation or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this license, they do not excuse you from the conditions of this license. If you cannot distribute so as to satisfy simultaneously your obligations under this license and any other pertinent obligations, then as a consequence you may not distribute the program at all.

END OF TERMS AND CONDITIONS.

9.2 boost license

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer,

must be included in all copies of the Software, in whole or in part, and
all derivative works of the Software, unless such copies or derivative
works are solely in the form of machine-executable object code generated by
a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

9.3 xerces license

Xerces-C++ unterliegt der Apache Software License, Version 2.0.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Index

- * -

- ? -

- A -

- B -

- C -

- D -

- E -

- F -

- H -

- I -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

Endnotes 2... (after index)

Back Cover