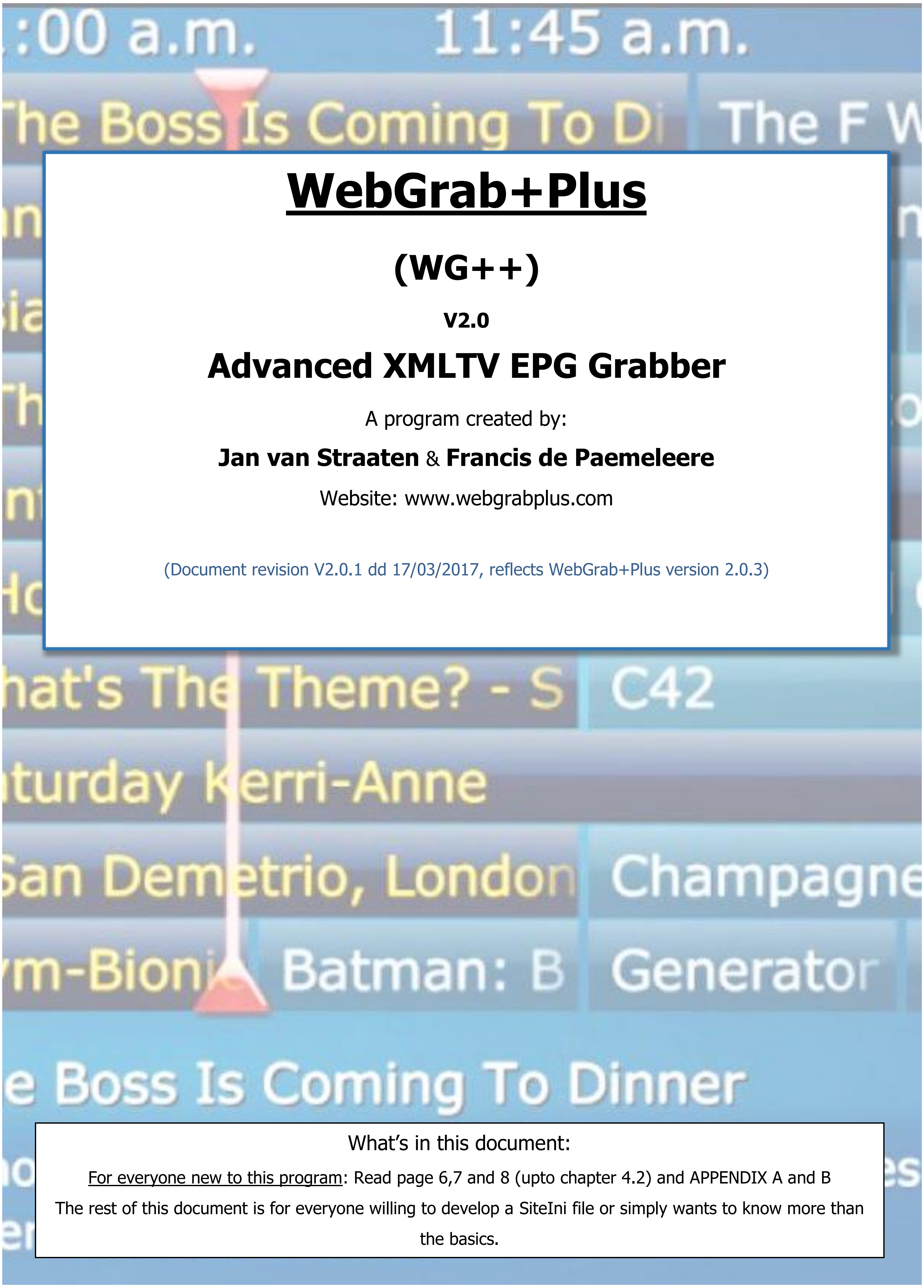

WebGrab+Plus

(WG++)

V2.0

Advanced XMLTV EPG Grabber

A program created by:

Jan van Straaten & **Francis de Paemeleere**

Website: www.webgrabplus.com

(Document revision V2.0.1 dd 17/03/2017, reflects WebGrab+Plus version 2.0.3)

What's in this document:

For everyone new to this program: Read page 6,7 and 8 (upto chapter 4.2) and APPENDIX A and B

The rest of this document is for everyone willing to develop a SiteIni file or simply wants to know more than the basics.

New in this issue:

w.r.t docoment V2.0:

- Text of chapter *1.2 How to run, files and folders* adapted to reflect the V2.0 installer
- Corrected the incorrect link to the configuration files on our website in *3. Configuration files*
- Changed chapter *4.4.1.2 argument preload* added the latest functionality of preload
- New chapter *4.4.3 url_preload* to introduce the latest functionality of preload
- A few small changes to correct text errors

w.r.t. document V1.1.6:

- Revised chapter *6.4 Series episode details*. Explains step by step how to develop a MDBIni for series data extraction using the new option to match the xmltv episode number besides the already existing episode title.
- New chapter *6.4.1 MdbIni for series example.*
- New chapter *6.3.3 MDBini scopes.* Describes the use and effect of the recently introduced MdbIni scopes.
- New elements mdb_episodenumlist and mdb_episodetitlelist in *6.2 MDB Elements.*
- Removed APPENDIX B , example config files. These can now be found on the WebGrabPlus website
- New *APPENDIX B Times, time-zones and DST corrections*. Explains how xmltv time must be interpreted.
- New additionds in *4.6.4.5.4 Bitwise Calculations*, the bitwise shift commands
- New style *argument* style=htmlencodespecialchar and style=htmldecodespecialchar
- New style *argument* style=base64encode and style=base64decode
- New chapter *4.6.4.6.3 Cleanup as string converter.* Examples of all the possible string conversions that can be done with the command cleanup
- New format *argument* format=chartodec to be used in *4.6.4.5 Calculate*
- New type *arguments* type=run
- New chapter *4.6.4.10.1 Set with argument type=run* Explains how to run an external program to get an element value from the output of that external program.
- New chapter *4.6.5 Operations (and scrubs) with argument 'pattern'.* Dealing with time and episode patterns in scrubs and operations.
- Element episode added to elements that are processed in a special way *4.5.2.2 The others* detailing how argument pattern can be used to convert episode numbers.
- New *argument* timezone to be used in *4.6.4.5.3 Date and time calculations* to covert a datetime into another time-zone
- All (at least most) references to other places in and outside this document are changed to real 'hyperlinks' for easier navigation.
- Plus numerous small changes and corrections not mentioned here.

Please report any omissions, errors or suggestions to jan_van_straaten@outlook.com

Table of Content:

WebGrab+Plus , an advanced XMLTV EPG Grabber

1.Introduction

Beside this manual, www.webgrabplus.com/documentation provides additional documentation of various topics not listed here.

1.1 What it does, features

The program grabs EPG data from TV Guide internet sites and

- runs in WINDOWS, LINUX and OSX and
- can grab from multiple sites in one run, programmable by user trough a SiteIni file
- very fast through incremental grabbing (only changed and new shows grabbed)
- programmable through editing commands that enable changing, filtering, adding, moving, removing (parts) and calculating of the xmltv elements.
- regular updates, support, documentation, user guides and a vast collection of SiteIni files available on *http://www.webgrabplus.com*

For a full list of features *(see APPENDIX A WebGrab+Plus Features)*

1.2 How to run, files and folders

For installation use either the *Windows Installation Package* or the *Linux installation Package* . For windows the program will install on the standard Program Files folder e.g. C:\Program Files (x86)\WebGrab+Plus for a 64 bit computer) (or another folder as option) and will also create a WebgrabPlus home folder *C:|Users|username|AppData|Local|WebGrab+Plus* and fills it with all the necessary files and sub-folders. The program can be run by a double click on the WG++ icon in this homefolder. This folder also contains a QuickStartGuide. For more details follow the instruction on *documentation/installation/windows*
For Linux, follow the instructions in *documentation/installation/linux*. As with Windows it will create a WebgrabPlus homefolder with all the necessary files and folders.
The *Linux installation Package* can also be used for OS-X (*documentation/installation/os-x*), Raspberry Pi (*documentation/installation/raspberry-pi*) and Synology NAS (*documentation/installation/synology-nas*)

1.3 Xmltv, *Single* - versus *multiple* - value xmltv elements

For an overview of the xmltv elements supported see *(APPENDIX E Element names)*, column: xmltv name.
According to the xmltv specification, some elements can have more than one value in the xmltv file. We distinguish *single value* xmltv elements (e.g. description) and *multiple value xmltv* elements (e.g. category, actor).
WebGrab+Plus treats them differently. For examples see *(4.2.4 Types)*
Note that the element 'title' is a *single value element* but the program supports a second version of the same title (titleoriginal) with different 'lang=' attributes. See *(4.2.5.6 Dedicated Arguments)* argument *lang*

1.4 Robots exclusion standard check

A quote from *http://www.robotstxt.org/orig.html* to explain:
quote/
WWW Robots (also called wanderers or spiders) are programs that traverse many pages in the World Wide Web by recursively retrieving linked pages. The 'Robots exclusion standard' is a common facility the majority of robot authors offer the WWW community to protect WWW server against unwanted accesses by their robots.
/end quote
Following this definition of WWW Robots, WebGrab+Plus is such a program. Therefore, it obeys the methods and rules of this standard in that it displays a warning to the user if a site disallows access to pages that the program wants to grab from.

2.The grabbing, show update process and update modes

2.1 The show update process

Assuming a previous xmltv listing exist (e.g. of yesterday), the program reads this and stores it as a target for update and as reference of what shows have to be changed or added. If no xmltv listing exists, the program creates a new one. Before grabbing show details, the program determines if the existing show in the xmltv listing is still valid or needs an update. For that it connects to the TV Guide website and grabs the so called index pages (the html pages that contain an overview the scheduled shows per timespan (e.g. day or several days)). It then compares the shows listed there (channel, start and stop times and title) with shows in the existing xmltv listing. As a result of this comparison the following situations occur:

- same (.), no update. The show in the index page is considered the same as the one in the existing xmltv listing.
- changed (c), update. The index show is different from the xmltv show but they have overlapping or equal time span.
- gap (g), insert. The index show fits in a time gap of the xmltv listing.
- new (n), add. The index show is new; it will be added to the end (or to the beginning if that is the case) of the xmltv listing.
- repair (r), update. This is a special situation that occurs if errors or overlapping shows are detected in the xmltv listing. The program will try to solve this by remove and update.

When the program runs, these resulting situations for each show are printed in the command window like this (the iiii indicates 4 days of index pages downloaded):

```
iiii..............g...........ccc........c.c.......g....r.....nnnnnnnnnnnnnnnnnnnnnnn
```

The comparison of the show title in the index page (index_title) and the one in the xmltv file is rather complicated and tricky. This is due to the fact that the index_title frequently differs from the one in the show detail page to a certain extend. Differences can be due to abbreviation of long titles, different use of punctuation characters and combination of title with other elements in the index_title (like category and subtitle). The program deals with all those differences through a weighted comparison. The result of this comparison is a 'title match factor', which , roughly, is the biggest percentage of 'matching' words between the two titles in any of the elements of the index_title. If this titlematchfactor is less than the value for it in the SiteIni file (see *4.3 General Site dependent data*) the show is considered - not same - and a show update is started.
For that it will grab the show details from the show detail html page(s) (see *2.3 The Grabbed Site pages*) of the TV Guide website if provided by it.

2.2 The update modes

The program supports a variety of update modes. The preferred and most efficient is :

- 'incremental' (i) Works as described above for all shows in the index page. In this mode the download time is minimized to the minimum.
- Other update modes are:
- 'light' (l) which is incremental but forces a re-grab of all shows for 'today',
- 'smart' (s) is the same with a forced re-grab for today and tomorrow,
- 'full' (f) not incremental, forces a full re-grab of all days requested.

Index-only mode:

Besides, and independent from, the modes mentioned above is a special grabbing mode :

- 'index-only' that is automatically selected by the program if no elements need to be scrubbed from the show detail page. (see also *4.5 Elements*) This mode is 'superfast' but seldom useful because most sites provide very little show data on the index page. But if you are satisfied with just start and stop times and a title it's there. Occasionally there is a site with richer data on the index page (like tvguide.co.uk). Some sites list only details on the index page or provide only more detailed information for some shows on detail pages. The program automatically recognizes these cases.

2.3 The Grabbed Site pages: index-page, detail-page and sub-detail-page

As explained in *2.1 The show update process* , the program starts with grabbing the index-page to get an overview of the shows for the time period for which epg data is requested. Depending on the update decision outcome and of the availability of them, the program grabs detailed show epg data from the show detail html page. Some sites split the epg data into sub-detail pages. The program supports additional grabbing from one or more of such sub-detail pages. (see *4.5 Elements* and *APPENDIX E Element names*)

3. Configuration files

3.1 WebGrab++.config.xml

This file supplies all TV Guide website independent settings for WebGrab+Plus. Among them are :

- *filename* The path and name of the xmltv output file
- *update* mode , as discussed in *2.2 The update modes*
- *timespan* ,the number of days to grab, (zero based)

and, most important, a

- *list of channels* to grab. Each channel for which epg data in the xmltv listing is requested needs to be added to this channel list. The channel data in this list consists of the *update* mode (see *2.2 The update modes*) , the *site* to get the data from (see *4. SiteIni file*) the *site-id* (the channel id of the site, see *4.4.2 url_index*), the *xmltv_id* (the id by which xmltv recognises the channel) and the channel *display name*.

Besides these, several other settings, like *mode* , *postprocess* , *proxy* , *user-agent* , *logging* , *credentials* , *retry* , *skip* and *channel-grouping.*

A typical concept WebGrab++.config.xml file is available at*http://webgrabplus.com/download/utility/documented-configuration-files*. It also provides the explanation of all the settings. The file is self-explanatory. For detailed configuration instructions see *http://www.webgrabplus.com/documentation/configuration*

3.2 MDB.config.xml

The MDB postprocessor of WebGrab+Plus, automatically adds movie and serie details from online 'MDB' sites (e.g. *IMDb.com* and *thetvdb.com*) to the xmltv file created by the basic WebGrab+Plus EPG frontend grabber. It has its own configuration file which resides in the subfolder \mdb of program’s home-folder. This mdb.config.xml file also serves as the mdb configuration user guide. An example of it is also available at *http://webgrabplus.com/download/utility/documented-configuration-files.* For detailed configuration instructions see *http://www.webgrabplus.com/documentation/configuration-mdb*

3.3 REX.config.xml

The purpose of this postprocessor is to re-arrange and edit the xmltv file created by the grabber section of WebGrab+Plus. This can be useful or necessary if the EPG viewer of the PVR/Media-Centre used, or the xmltv importer it uses, does not support all the xmltv elements in the xmltv file created by WG++.
It can:

- Move the content of xmltv elements to other xmltv elements
- Merge the content of several xmltv elements
- Add comments/prefix/postfix text
- Remove or create xmltv elements

E.g.: If the PVR doesn't support import of credit elements (actors, directors etc.) it can add the content of them to the description and remove the original credit elements which are useless. Or, it can move the episode data to the beginning or end of the subtitle element- Etc. ..
It has its own configuration file which resides in the subfolder \rex of program's home-folder. This rex.config.xml file also serves as the rex configuration user guide. An example of it is also available at *http://webgrabplus.com/download/utility/documented-configuration-files* , mdb.config.xml

4. SiteIni file

For each TV Guide website that is entered in the channel list of the config file (see above) a SiteIni file is required to supply WebGrab+Plus with site dependent settings. The name of this file is directly related to the value of the site attribute in the channel list through the addition of .ini to this value. (e.g. channel list site attribute : tvgids.nl .. SiteIni file name : tvgids.nl.ini)

4.1 SiteIni file parts

The data in this file consists of the following parts:

- A top header section that contains meta data like the site, the required WG++ version, revision number, date and author and eventual remarks.
- General Site dependent data (see *4.3*)
- Data that WebGrab+Plus needs to compose the url's to download pages (see *4.4 Url builder*)
- Data that WebGrab+Plus needs to scrub xmltv elements from the downloaded pages (see *4.5 Elements*)
- Optional data that allows post modification of the scrubbed xmltv elements (see *4.6 Operations*)
- A channel file creation part (see 4.5.3 *channellist*)

4.2 The SiteIni file basics

4.2.1 scrubstrings

A scrubstring is just one line in the SiteIni that specifies an action for a SiteIni element. The general format is

```
Elementname.action {type(arguments)|datastrings required for the action}
```

Most of the settings in this file relate to how WebGrab+Plus extracts, "scrubs", xmltv elements from the TV Guide website html pages. The program supports two methods for that: The 'separator strings method' (described in *4.2.1.1*), by means of element separator strings pointing to the start and end of the element to be scrubbed and the 'regular expression method' (described in *4.2.1.2*), by which the element to be scrubbed is extracted by means of a 'regular expression'.
Both methods can be used together mixed in one SiteIni file and both cover more or less the same functionality. The 'separator strings method' is the easiest to understand and is recommended if not familiar with 'regular expressions'. The 'regular expression method' can be considered as the 'expert' method and is extremely powerful and compact.

4.2.1.1 The 'separator strings' method

For that it uses (up to) 4 strings that should point to the beginning and the end of the element to scrub:

- the *element start* es and the *element end* ee string.
  They represent the unique strings (e.g. html tags or parts of it) between which required the element is always located on the html page. In most cases such unique es and ee are unavailable because somewhere else in the html page the same strings exist enclosing other data. In that case we need to separate the right es and ee pairs from the unwanted pairs.
  For that we use the block separators:
- *block start* bs and *block end* be *.*
  These should enclose a html region (block) in which es and ee enclose our wanted element and nothing else.

Consider the following sample html:

```
 <div id = "detail-page">
   <div id = "program-content">
     <div id = "program-info">
       <img alt="RTL 7" id="channel-logo" src="/media/pc/epg_upc/nl/channel_logos/rtl7.gif" />
       <h3>Basilisk: Serpent King</h3>
       <a class="channel-title" href="/TV/Guide/Channel/RTL+7/Today/">RTL 7</a>
       <div id = "program-desc-text">
         Amerikaanse actiefilm. Een team van archeologen ontwaakt een mythische slang die vernieling zaait. De enige
manier om het wezen te stoppen is door een magische scepter te vinden.
       </div>
       <!-- Genre Subgenre Data -->
       <dl>
       <dt>Genre:</dt><dd>speelfilm</dd>
      <dt>Genre:</dt><dd>sequel</dd>
      <dt>Subgenre:</dt><dd>avontuur</dd>
<dt>Duur:</dt><dd>90 min</dd>
<dt>Regie:</dt><dd>Louie Myman</dd>
<dt>Met:</dt><dd>Jeremy London, Wendy Carter, Griff Furst, Cleavant Derricks, Daniel Ponsky, Bashar
Rahal</dd>
       </dl>
```

To scrub the title - *Basilisk: Serpent King*- we need es= *<h3>* and ee= *</h3>*. In fact if it is sure that <h3> tag is uniquely used to enclose the title we wouldn't need more than that. However even if that is the case on this (part of) html page, simple html tags like <h3> are seldom unique and thus it is more secure to use the block separators bs= *<div id = "program-info">* and be = *<a class*
It is a little different with the description, here es= <div id = "program-desc-text"> and ee= </div> . Very likely this es is unique for the description, so we wouldn't need block separators.
Strings like bs, es, ee and be will be called separatorstrings in the remainder of this document.
The syntax in which the SiteIni file expects them is :

{type(optional arguments)|bs|optional es|ee|optional be}
or:
{type(optional arguments)|bs|optional es|optional ee|be}

To complete a SiteIni scrubstring we need to add the xmltv element name and an action specifier :

ElementName.ActionSpecifier {type(optional arguments)|separatorstrings}

The scrubstrings for two scrubstrings from above for description and title respectively:

```
description.scrub {single|<div id = "program-desc-text">||</div}
title.scrub {single|<div id = "program-info">|<h3>|</h3>|<a class}
```

4.2.1.2 The 'regular expression' method

Most of the functionality to extract selected EPG data from the pages received from the TV Guide sites is covered by the 'separator string' method described in the previous chapter. Besides that the program offers the 'regex' method. This method is very powerful and flexible but requires detailed understanding of the Regular Expression syntax. Therefore it is only recommended for experienced SiteIni designers with this knowledge or the determination to delve into it. As introduction a quote from Wikipedia, *http://en.wikipedia.org/wiki/Regular_expression* :
quote/
*...... a **regular expression** (abbreviated **regex** or **regexp**) is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. ....*
/end quote
The program supports the use of these 'regex' as an alternative for the *'separator strings method '* described in *4.2.1.1*
More information about regular expressions in general, besides the Wikipedia article, can be found in *http://www.regular-expressions.info/tutorial.html* and numerous other articles in the www domain.
The program uses the .NET regular expressions library as documented in *http://msdn.microsoft.com/en-us/library/hs600312.aspx*
The detailed description of the syntax and use of this extraction method is found in chapter *4.2.4.3 type regex*.

4.2.2 Element names

In this document an *element* is defined as a named string object or an array of them in which the result of an action (scrub, modify etc. see *4.2.3 Action specifiers*) is stored. Their name consists of a fixed part (see *APPENDIX E Element names*, first column) that describes the type of data it contains — and, in most cases, a prefix that indicates from which of the html pages (see *2.3 The Grabbed Site pages*) its data is obtained.
A complete list of supported elements can be found in *APPENDIX E*, first column, SiteIni name.
Elements with a prefix index_ are scrubbed from the index page, the ones with prefix detail_ or without a prefix from the show detail page and the ones with prefix subdetail_ from the sub-detail page. Notice that most elements can be scrubbed from either of the three possible html pages. This depends on the actual content of these pages. It is allowed to have an element scrubbed from more than one page, in that case the scrubbed values will be added in a way which depends on if the element is a *multiple value* xmltv element or not. (see *1.3* and *4.2.4*). In the case of a *multiple value element* they will be listed as separate elements, while in the case of a *single value element* the values are merged.
The obligatory elements (un-checked *optional* column in *APPENDIX E*) are either required for proper functioning of the program (url_index and urldate to connect to the site, index_showsplit to separate the show index parts, index_start and index_title for update decision making) or as a minimum for a meaningful xmltv output (index_start and title or index_title).

4.2.3 Action specifiers

Action specifiers are either *url* (optional)*, headers, format, scrub* or *modify.* They specify what kind of action the program has to perform. See *APPENDIX E* , colunn 'action' for an overview.

4.2.4 Types

Type is url, single, multi or regex

4.2.4.1 type *url*

Scrubstring specifications for this type have varying formats to build the url's to connect to the various site pages. (see *4.4 Url builder*)

4.2.4.2 types *single* and *multi*

Very often elements in html pages are divided in several paragraphs or otherwise split into parts such that no *single* pair of element separators (es and ee) enclose the element.
Suppose the description in the html page looks like this:

```
        <div id = "program-desc-text">
          <p>Amerikaanse actiefilm.</p>
          <p>Een team van archeologen ontwaakt een mythische slang die vernieling zaait. De enige manier om het
wezen te stoppen is door een magische scepter te vinden.</p>
          <p>Geproduceerd in 1998</p>
        </div>
```

In such a case we use type *multi* to instruct WebGrab+Plus to scrub all the elements within the block with the specified separatorstrings, like here es = <p> and ee = </p>

To illustrate the scrub results from this html with type *single*:
description.scrub {single|<div id = "program-desc-text">|<p>|</p>|</div>} will result in :

<desc lang="xx">Amerikaanse actiefilm.</desc>

While the same with type *multi* :
description.scrub {multi|<div id = "program-desc-text">|<p>|</p>|</div>} will result in :
<desc lang="xx">Amerikaanse actiefilm. Een team van archeologen ontwaakt een mythische slang die vernieling zaait. De enige manier om het wezen te stoppen is door een magische scepter te vinden. Geproduceerd in 1998</desc>
Notice that WebGrab+Plus adds the three description paragraphs together. This is due to the fact that the element description is a *single value* xmltv element. (see *1.3* and *4.2.2*)
To illustrate what happens with a *multiple value* xmltv elements, consider the category.
In the html the genre and subgenre are the obvious choice for that. Xmltv doesn't specify a subgenre element, so we take them all together as category

```
<!-- Genre Subgenre Data -->
<dl>
<dt>Genre:</dt><dd>speelfilm</dd>
<dt>Genre:</dt><dd>sequel</dd>
<dt>Subgenre:</dt><dd>avontuur</dd>
<dt>Duur:</dt><dd>90 min</dd> </dl>
```

There are two genre entries in the html, with the same element separators, so we use type *multi* to grab them both.
category.scrub {multi|<!-- Genre Subgenre Data -->|<dt>Genre:</dt><dd>|</dd>|</dl>}
The result will be the following xmltv listing for category:

```
<category lang="xx">speelfilm</category>
<category lang="xx">sequel</category>
```

Because category is a *multiple value xmltv* element the two are not joined to one xmltv element but listed as separate category elements.
To add the third category element , the Subgenre in the html, we use another feature of the SiteIni specification : For most SiteIni elements it is allowed to use more than just one scrubstring for the same xmltv element! (see *APPENDIX E*, column -multiple scrub- for which this is allowed)
So we add:
category.scrub {single|<!-- Genre Subgenre Data -->|<dt>Subgenre:</dt><dd>|</dd>|</dl>}
The final result:

```
<category lang="xx">speelfilm</category>
<category lang="xx">sequel</category>
<category lang="xx">avontuur</category>
```

4.2.4.3 type *regex*

Used to specify the 'regex' method of data extraction. See also *4.2.1.2* for some background information. This method doesn't need the type *single* and *multi* distinction as is explained below.
The syntax:

Element.scrub {regex(optional argument)||regular expression||}

- *regex* : the action specifier for this method
- *argument* : for this method the only arguments supported are *debug* *4.2.5.5* and *pattern* *4.2.5.6*
- *regular expression*: The regular expression that matches the desired element content.
- The place of the regular expression in the scrubstring is the same as the 'element start - es' in the separator string method (see *4.2.1.1*, syntax) or, simply put: two || in front and two || after it.

The easiest way to get started with this method (after mastering the 'separator string' method) is to use a direct substitution of the separator strings be, es, ee and be used there.
Remember the syntax for the separator string method (see *4.2.4.2* type *single* and *multi* explanation)
For type *single*:

Element.scrub {**single** (arguments)|bs|es|ee|be}

A direct 'regex' substitute for it will be:

Element.scrub {**regex** (arguments)||bs(?:.*)es(.*?)ee(?:.*?)be||}

Examples: The 'separator string method' title solution of the previous chapter:

title.scrub {**single**|<div id = "program-info">|<h3>|</h3>|<a class}
can also be achieved with the following regex scrubstring:

title.scrub {**regex**||<div id = "program-info">(?:.*)<h3>(.*?)</h3>(?:.*?)<a class||}

And for the description :

description.scrub {**single**|<div id = "program-desc-text">||</div>}
description.scrub {**regex**||<div id = "program-desc-text">(.*?)</div>||}

For type *multi* the substitution is as follows:

Element.scrub {**multi**(arguments)|bs|es|ee|be}

Will look like this in regex:

Element.scrub {**regex**(arguments)||bs(?:.*)(?:es(.*?)ee(?:.*?))*be||}

Example the category of chapter *4.2.4.2*
category.scrub {**multi**|<!-- Genre Subgenre Data -->|<dt>Genre:</dt><dd>|</dd>|</dl>}
category.scrub {**regex**||<!-- Genre Subgenre Data -->(?:.*?)(?:<dt>Genre:</dt><dd>(.*?)</dd>(?:.*?))*</dl>||}

4.2.5 Arguments

Arguments can be either/and includeblock, excludeblock, separator, max, include, exclude, debug and dedicated arguments lang, force, pattern, sort, timespan, preload, alloc, target.
!! All these arguments are irrelevant for type regex, with the exception of debug and pattern !

4.2.5.1 Argument includeblock and excludeblock

If it is only possible to find blocks that, apart from the required information, contain unwanted information with the same separatorstrings *es* and ee , these arguments can be used to select the correct blocks. The syntax:

includeblock=bn1,bn2, .. ,bnn/tn -or- "string-1""string-2" .. "string-n"
excludeblock=bn1,bn2, .. ,bnn/tn -or- "string-1""string-2" .. "string-n"

- *bn* , the block number to include or exclude, starting with 1
- *tn* , the number of blocks for which the block numbers bn repeat
- "*string*" , include or exclude only the blocks that contain the "string". When more than one "string" is entered, the block selection is done by an 'or' function of the strings. The use of wildcards [x] and [?] is allowed (see *4.2.6 String matching / wildcards*)
  Example : includeblock="abc""def" , the blocks included contain the string "abc" or "def" .
  When more than one "string" is entered separated by the char & , the block selection is done by an 'and' function.
  Example : includeblock="abc"&"def" , the blocks included contain the string "abc" and "def".
- All characters are allowed.
- The characters " ' { and) need to be preceded by \ . So the string ("O'Neil {superhero}") must be entered as "\(\"O\'Neil \{superhero}\"\)"

4.2.5.2 Argument *separator*

As example take a look at the actors :
<dt>Regie:</dt><dd>Louie Myman</dd>
<dt>Met:</dt><dd>Jeremy London, Wendy Carter, Griff Furst, Cleavant Derricks, Daniel Ponsky, Bashar Rahal</dd>
</dl>

If we use : actor.scrub {single|<dt>Met:</dt>|<dd>|</dd>|</dl>} the xmltv listing of actor will be
<actor>Jeremy London, Wendy Carter, Griff Furst, Cleavant Derricks, Daniel Ponsky, Bashar Rahal</actor>
That is clearly not what we want. To separate them we use the *separator* argument. It specifies which string or strings separates the elements. Its syntax is:

separator="string-1" "string-2" .. "string-n"

- Between the separator strings a space is allowed but not required.
- All characters are allowed with the exception of | (vertical line). This is no limitation of this function because the program will automatically replace all | characters in the html page into the character combination !?!?!, this to avoid problems with the special function of this character.
- The characters " ' { and) need to be preceeded by \ So the string ("O'Neil") must be entered as separator="\(\"O\'Neil\"\)"

The scrubstring for actor then becomes:

actor.scrub {single(separator=", ")|<dt>Met:</dt>|<dd>|</dd>|</dl>}

and the resulting xmltv listing:
<actor>Jeremy London</actor>

```
                    <actor>Wendy Carter</actor>
                    <actor>Griff Furst</actor>
                    <actor>Cleavant Derricks</actor>
                    <actor>Daniel Ponsky</actor>
                    <actor>Bashar Rahal</actor>
```

Suppose the html line with the actors looked like this:
<dt>Met:</dt><dd>Jeremy London, Wendy Carter, Griff Furst, Cleavant Derricks, Daniel Ponsky and Bashar Rahal</dd>
(The last two actors separated by the word - and -) We then can use separator=", " " and " for the same result.

4.2.5.3 Argument *max*

To limit the number of elements (either added together in the case of *single value* xmltv elements or listed separately in the case of *multiple value* xmltv elements) we can use the argument max. Its syntax:

```
max=n
```

in which n=positive integer
actor.scrub {single(separator=", " max=3)|<dt>Met:</dt>|<dd>|</dd>|</dl>} will result in:

```
          <actor>Jeremy London</actor>
          <actor>Wendy Carter</actor>
          <actor>Griff Furst</actor>
```

4.2.5.4 Arguments *include* and *exclude* :

These allow further control over which of the scrubbed elements will be passed to the final result. It is important to realise that both *include* and *exclude* can be used together in one scrubstring. The program will execute these in the order in which they occur in this specification. See for an example of the effect of this in *5.2 Tricks*
Its syntax:

include=n -or- first -or- firstn -or- last -or- lastn -or- "string"

exclude=n -or- first -or- firstn -or- last -or- lastn -or- "string"

- *n* the element number to include or exclude, starting with 1
- *first* or *firstn* (like first2) , the first or the first n elements to include or exclude
- *last* or *lastn* (like last2) , the last or the last n elements to include or exclude
- "*string*" , like "met o.m.", include or exclude only elements containing the "string". The use of wildcards [x] and [?] (see *4.2.6*) is supported.
- All characters are allowed with the exception of | (vertical line). This is no limitation of this function because the program will automatically replace all | characters in the html page into the character combination !?!?! , this to avoid problems with the special function of this character.
- The characters " ' { and) need to be preceded by \ So the string ("O'Neil") must be entered as "\(\"O\'Neil\")"
- As with the argument *separator* (see *4.2.5.2*) a list of strings is allowed like:

include="string-1" "string-2" .. "string-n"

The effect of these arguments differs depending on whether it is entered in - combination and after the argument *separator* — (case A) or not (case B).

- Case A (after the argument separator):
  In this case it allows to make a selection of the elements we want after they are separated.
  As example we use the following html for a title and sub-title combination that occurs frequently:

  ```
          <div class="intro-datasheet">
                  <div class="img">
                          <img src="/img/programas/fotos/Motociclismo255.jpg"  alt="" />
                          <p>Motociclismo: Cto. del Mundo</p>
                  </div>
  ```

  Here, the title Motociclismo, is separated from the sub-title Cto. del Mundo with a : character.
  So we can use the arguments separator=": " to separate them , we then use include=first for the title and exclude=first for the sub-title, like this:
  title.scrub {single(separator=": " include=first)|<div class="intro-datasheet">|<p>|</p>|</div>}
  subtitle.scrub {single(separator=": " exclude=first)|<div class="intro-datasheet">|<p>|</p>|</div>}
  The xmltv result :

  ```
    <title lang="es">Motociclismo</title>
    <sub-title lang="es">Cto. del Mundo</sub-title>
  ```

- Case B (not after the argument *separator*):
  The program will evaluate all the scrubbed elements (single or multi) on the conditions specified by the include and/or exclude values.
  As example we use the description again:

```
    <div id = "program-desc-text">
      <p>Amerikaanse actiefilm.</p>
      <p>Een team van archeologen ontwaakt een mythische slang die vernieling zaait. De enige manier om het
wezen te stoppen is door een magische scepter te vinden.</p>
      <p>Geproduceerd in 1998</p>
    </div>
```

  Remember the original scrubstring:
  description.scrub {multi|<div id = "program-desc-text">|<p>|</p>|</div} resulted in :

```
<desc lang="xx">Amerikaanse actiefilm. Een team van archeologen ontwaakt een mythische slang die
vernieling zaait. De enige manier om het wezen te stoppen is door een magische scepter te vinden.
Geproduceerd in 1998</desc>
```

  But, the last element - Geproduceerd in 1998 - actually belongs to another xmltv element - date - which is meant to contain the date of production. So in fact it shouldn't be part of the description. We can use the following to exclude it from the description:
  description.scrub {multi(exclude="Geproduceerd")|<div id = "program-desc-text">|<p>|</p>|</div}
  or if we are sure that it is always the last element that contains the production date:
  description.scrub {multi(exclude=last)|<div id = "program-desc-text">|<p>|</p>|</div}
  or if it is always the third:
  description.scrub {multi(exclude=3)|<div id = "program-desc-text">|<p>|</p>|</div}
  or
  description.scrub {multi(include=first2)|<div id = "program-desc-text">|<p>|</p>|</div}
  Even this works!
  productiondate.scrub {multi(include="Geproduceerd")|<div id = "program-desc-text">|<p>|</p>|</div} or
  productiondate.scrub {single|<div id = "program-desc-text">|Geproduceerd|</p>|</div} both will result in:
  <date>1998</date>
  (this works because WebGrab+Plus finds any year value inside an element for the date xmltv element, see *4.5.2.2*)

4.2.5.5 Argument *debug*

Adding the word *debug* as argument will start logging of the scrubbing process for the element in the *WebGrab++.log.txt* file. The html page from which the scrubbing is attempted is written to a separate file *html.source.htm.* One should use this argument (preferably) for one element and one show at the time, otherwise the results could be confusing. The config file allows to Grab only one show with the -timespan- setting. Another way to reduce the debug logging to one show is to add an index number *n*, like *debug.4* , in this way only the 4th show from the index page will be logged.

4.2.5.6 Dedicated Arguments

The following arguments are dedicated to the use with a certain element

- *lang :* This argument only works for the element *titleoriginal*. See *4.5.2.2*, titleoriginal.
- *force :* This is a special argument to change the effect of scrubbing the element *index_date* (see *4.5.2.1*)
- *sort* and *timespan* : Arguments to be used together with *index_showsplit* in case of fragmented multiday index_pages (see *4.5.1 Non optional elements*)
- *preload* : Used together with any of the 3 url elements (see *APPENDIX E* and *4.4 Url builder*) Can be used to specify an url that is preloaded before the actual url that calls the requested html page.
- *alloc* : Can be used in the special elements *index_site_id* and *index_site_channel* to specify the target xmltv elements. See *4.5.3 Special elements*
- *target* : Must be used together with the special element *sort_by* which holds the data by which a *target* multi value element will be sorted with the command sort (see *4.5.3 Special elements* and *4.6.4.9 Sort*)
- *pattern* : Can be used to specify a non standard date/time pattern for the elements *start* and *stop.* (See *4.5.2.1 Time elements*)

4.2.6 String matching / wildcards

The arguments *includeblock, excludeblock, include* and *exclude* , as described in *4.2.5 Arguments* one can use strings to match with data in the elements. Normally the program uses a one-to-one-case-sensitive match. It is possible however to use 'wildcards' in the strings to match. Wildcard syntax :

```
[x] represents a multiple character wildcard
[?] represents a  single   character wildcard
```

Examples:

- "a[*]c" matches with "abc" and "avdgec" etc.
- "a[?]c" matches with "abc" and "ahc" etc.

Wildcards can also be used in conditional arguments (see *4.6.2*)

4.2.7 TimeZones

Read (*APPENDIX B Times, time-zones and DST corrections*) first!
For the calculation of the xmltv start and stop times, the program needs the time zone and the daylight-saving-time (dst) rules applicable for that time zone. This must be entered by means of a timezone_id in the *time zone* parameter of the General Site dependent data (see *timezone*). The syntax :

```
timezone=timezone_id
```

- timezone_id, e.g *US/Eastern* or *Europe/Brussels* or *Asia/Singapore* or *UTC-05:00* or *UTC*
  The program contains an integrated time zones database of more than 400 of such timezone_id's together with their dst rules which is based on *tzdata* as distributed by *http://www.iana.org/time-zones*.
  Entering *timezone=?* will list all the available timezone_id's together with their basic utc offset in the logfile. This to facilitate the choice. As the examples illustrate, it is also possible to enter just the UTC offset, like UTC-05:00, the program will map this offset to the most likely timezone_id. Just entering UTC without the offset is a special case to be used if the guide times are listed in UTC , without dst rules. (this is not the same as UTC+00:00 !!)
- For the processing of the time zone database the program uses a customized version of the public domain ZoneInfo Api developed by Mark Rodrigues as published @
  http://www.codeproject.com/Articles/25001/ZoneInfo-tz-Database-Olson-Database-NET-API
- Timezones database updates : As this database is integrated into the program, updates, if necessary, will be provided by a program update published on *www.webgrabplus.com*. It is also possible to place an updated version of this database in the same folder as the executable *WebGrab+Plus.exe.*
  The program accepts two variants of this database, as *TimezonesData.txt,* a single file variant provided by the *www.webgrabplus.com* in the download section —or– as a folder *tzdata* as provided by *http://www.iana.org/time-zones*
  In both cases the program will use the external database instead of the integrated one.

4.3 General Site dependent data

One or more lines with the following syntax:

```
site {url=x.x|timezone=tz-id|maxdays=n.p|cultureinfo=xx-XX|
charset=xxx,yyy|titlematchfactor=nn}
```

and the following (and more) are optional:

```
site {ratingsystem=xxx|episodesystem=xxx|grabengine=wget|
firstshow=n|firstday=nnnnnnn|subtitlestype=xxx|retry=xxx}
```

Site dependent data can be entered on one or more lines starting with the word 'site'

- *url*, e.g. url=tvgids.nl , the url of the site e.g. url=tvgids.nl
- *timezone*, e.g. US/Eastern or UTC+01:00, the timezone for which the TV guide data is given. See *4.2.7 TimeZones* for details.
- *maxdays*, specifies the number of days n for which TV guide data is provided by the site, followed by how many index pages p are used for it. If n and p are equal, e.g. 7 days on 7 pages, you can either specify 7.7 or just 7. However if the site has a multiday e.g. a weekly index_page 7.1 must be specified. (See also *4.5.1*, index_showsplit)
- *cultureinfo*, e.g. cultureinfo=nl-NL , gives data about standards for time and language formats used by the site. For more info : *http://msdn.microsoft.com/en-us/library/system.globalization.cultureinfo(v=VS.95).aspx*
  It is allowed to only specify the language part of it, like cultereinfo=en , but the results might be different, especially in country specific items like time formats.

- *charset*, e.g. charset=ISO-8859-1 or UTF-8. Charset is normally found somewhere at the beginning of the grabbed site page. Sets proper decoding of these pages. This charset is applied to all grabbed html pages (index, show-detail and sub-detail). Sometimes the charset for these pages is different. In that case specify them separately, separated by a comma. The first will be used for the index page the second for the show-detail and the sub-detail page.
- *titlematchfactor*, e.g. titlematchfactor=50 , this is a number from 0 to 100 that specifies how strict the title comparison is done by WebGrab+Plus (as discussed in *2.1 The show update process*). Some sites use different show titles for the index pages and show detail page. Start with a high value e.g. 90 and adjust to lower if too many unnecessary show updates occur. (see also *4.5.1*, element index_title). A value of 0 disables title comparison.
- *ratingsystem* (optional) Specifies the system attribute of the xmltv element rating. Some countries have a uniform system to classify shows (e.g. the MPAA in the US and KIJKWIJZER in the Netherlands). If the site's country has no such system it is best to use a two letter country spec like ES for Spain.
- *episodesystem* (optional), specifies the xml attribute system of the episode-num xmltv element. See xmltv specification for details *XMLTV.DTD* . The most common values are *xmltv_ns* and *onscreen.*
- *grabengine* (no longer supported!) Only the internal grabengine is used. It was used to specify which of the two available grabengines (the part of the program that connects to the site and grabs the html pages) will be used for this site.
- *firstshow* (optional). Specifies which is the first show on the index_page that will be processed (scrubbed). When not specified, or if firstshow=0, it starts with the first show found on the index_page. This value is important for sites that lists shows on the index_page from the previous day 'yesterday', because the program assumes that the first show is of 'today' if there is no date component in the times listed. A mix-up of the date value will be the result. The firstshow value allows to skip these 'yesterdays' shows. Instead of a number, the string *now* can be used. This will skip all shows until a day change (passing midnight) is detected.
- *firstday* (optional), e.g. firstday=0123456 This is to be used if the site has a multiday index page (an overview of shows for several days). When in such a case, this index_page doesn't change for several days (remains starting on the same day), the program needs info where to start. The firstday value tells the program how many days to skip to find the shows of 'today'. It needs to be entered as 7 numbers, from the first : days to skip on Monday .. To the last : days to skip on Sunday. Example:
  Suppose a multiday index_page which lists the shows for a week starting Sunday. Then, if we grab on Sunday, there is no need to skip a day, but on Monday we must skip 1 day (the Sunday), on Tuesday we must skip 2 days ... Etc. We specify firstday=1234560
- *subtitlestype* (optional). Specifies the xmltv attribute type of the element subtitles. Possible standard values are *teletext, onscreen* and *deaf-signed.*
- *retry* (optional). This is the same retry setting as the general retry setting in the config file as mentioned in *3.1*. If a site is markedly slower than others used in the same run, it is possible to set different retry, timeout and delay values for that site here. The syntax is the same as in the config file. E.g. retry=<retry>12</retry> or retry=<retry time-out="10" channel-delay="5" index-delay="1" show-delay="1">4</retry>
- *keeptabs* (optional) This will disable the default replacement of tab \t characters in spaces in html pages. In some cases tabs can be useful in scrubstrings.
- *keepindex_page* (optional) Saves the index-page for use with other channels of the same site. Useful when a site list all or a group of channels on one index-page. It saves grabbing the same index-page again and again.
- *loadcookie* (optional) If a site requires a login with username and stores your personal settings in a cookie, it is necessary to load this cookie into WG++ for it to send to this site as part of the WebRequest. Specified as *loadcookie=cookie-file-name.* The cookie-file-name is the name of the cookie file which must be present in the WebGrab home folder. (see loadcookie.txt on the download page for how to create such a file , *http://www.webgrabplus.com/sites/default/files/download/documentation/Set%20of%20help%20files/help-files.zip*). The program filters the cookies in this cookie-file for the cookies relevant for the site, using the url (see above) as domain. Optionally the *cookie-file-name* can be followed by additional domain strings that specify which of the cookies for other domains will be kept. Example : site{loadcookie=yourtv.com.au.cookie.txt} or site{loadcookie=yourtv.com.au.cookie.txt,yahoo.com}
- *skip* (optional) Can be added if a site needs a different (than the one in the config file) setting for any of the values of skip. It overrules the config setting. It must be specified using the same syntax as required for the config file.
  E.g. skip=<skip>16,1</skip>

- *compression* (optional) The program is able to decompress compressed site responses. Specifying a value for compression like *gzip* or *deflate* will invite the site (by means of the httpwebrequest header *Accept-Encoding*) to use compression. If a compressed response is the result, the program will automatically enable decompression. If no value is entered for compression the header Accept-Encoding will not be issued.
- *nopageoverlaps* (optional) If subsequent index pages have no overlapping index shows, this setting can be used to disable the automatic removal of these duplicates. This automatic removal can cause problems when channels have few shows per day and/or long gabs in the programming.
- *allowlastpageoverflow* (optional) When the last shows on the index page are in fact shows from the following day, specifying this will grab these shows even when outside the range of the <timespan> setting in the config file.

Examples:

```
site {url=tvgids.nl|timezone=Europe/London|maxdays=6|cultureinfo=en-US|charset=ISO-8859-1,UTF-8}
site {titlematchfactor=90|firstshow=5}
site {ratingsystem=KIJKWIJZER|episodesystem=xmltv-ns|retry=<retry time-out="15">10</retry>|keeptabs}
site {keepindex_page|loadcookie=yelo.be.cookies,yahoo.com|skip=<skip>noskip</skip>}
site {compression=gzip|nopageoverlaps|allowlastpageoverflow}
```

4.4 Url builder

Data that WebGrab+Plus needs to compose the url's to download pages

4.4.1 General URL settings

The majority of the TV Guide websites use the HTTP or HTTPS protocol. The following chapters describe the setup for this protocol. Beside that the program supports the FTP and the File protocol. These are described briefly in *4.4.4 the FTP and File protocol*

4.4.1.1 HTTP Headers, method GET, POST, POST-BACK and SOAP

The program supports the following HttpWebRequest methods:

- Method GET: The default way (method) to get a response from a site for a specific html page is to do a 'GET' HttpWebRequest for the url of that page. The URL contains all the necessary details to specify the requested content, usually in the form of a *channel* and a *date* variable.
- Method POST (see *5.1.1* for more details): A common alternative way (method) is to do a 'POST' HttpWebRequest to a specific url which is accompanied by a header 'postdata' with a string which further specifies the request.
- Method POST-BACK (see *5.1.2* for more details): This is a rare variant of the POST method. It starts with a GET request on which the site responds with a html page containing the 'postdata' , most of it, but not all, in a variable named *VIEWSTATE*. All the following requests are to be done with method POST using the 'postdata' as received from the site.
- Method SOAP (Simple Object Access Protocol) (see*5.1.3* for more details): Another rare variant of the POST method. In this variant the details of the request are not send to the site by means of the 'postdata' header but by means of an xml file containing the *soapEnvelope.*

To specify which of these methods is to be used the action specifier *headers* must be used. Besides *method* a number of other headers can be set in this manner.

The syntax:

```
urlname.headers {headername=string|…|headername=string}
```

- urlname : either url_index, index_urlshow or index_urlsubdetail
- headername : The following headernames are recognized by the program:
  - method=GET (default), method=POST, method=POST_BACK and method=SOAP
  - contenttype=application/x-www-form-urlencoded (default) or another contenttype string
  - referer=string (optional)
  - accept=string (optional)
  - credentials=name,password or name,password,domain
    (optional, for sites that require the approved identity of the user)
  - allowautoredirect=boolean (optional) Enter as string values : true (default) or false
  - expect=string (optional)
    Values : 100-continue=false or 100-continue=true or 100-continue (same as =true)
  - host=string (optional), to overrule the default host header that equals the url of the request
- Most other headers that might be required can be constructed with a 'customheader':
  - customheader=headername=headervalue
    e.g. customheader=Accept-Encoding=gzip,deflate

- And, only of significance for *method=POST, POST_BACK* or *SOAP :*
  - postdata=post-data-string
    This header which is, by its definition, only meant to contain the postdata string for a regular POST method, like POST and POST_BACK.
    The program also uses its value for the SOAP method to fill the soapEnvelope xml file (see *5.1.3*)
  - *post-data-string* : this string may contain the following variable components:
    in a postdata header for url_index:
    - 'urldate' to specify a date/time component (see *4.4.2.1 urldate format*)
    - 'channel' to pass the site_id of the channel
    - 'index_variable_element' or any other global element (see *4.5.3 Special elements*)
    - 'subpage' to pass the subpage data (see *4.4.2.2 subpage format*)

    in a postdata header for index_urlshow or index_urlsubdetail:
    - 'index_variable_element' or any other global element (see *4.5.3 Special elements*)
- All headers may also contain the element values: 'index_temp_1 to _9' and any of the 'global' elements.They will be expanded to their content value similar to other variable components.

Example:

```
url_index.headers {method=POST|contenttype=application/x-www-form-urlencoded}
url_index.headers {postdata=getEPG&StartTime='urldate'&ChannelIDs='channel'}
```

4.4.1.2 argument preload

Some sites require a call to a specific url prior to the one with the required data. The program saves the cookies from the response of this preload and re-issues it in the following httpwebrequest. Such a preload can be done by adding the argument *preload.* Example:

```
url_index {url(preload="http://www.mobistar.tv/tv-guide.aspx")|http://www.mobistar.tv/epg.aspx?f_format=pgn&medium=0&lng=nl&f=|urldate|&t=xxxxx&s=|channel|,0,2,&_=|urldate|}
```

An alternative way to specify this preload url is described in section It is possible to specify a set of HTTP headers for this url.

4.4.2 url_index

This is the url WebGrab+Plus uses to download the index pages (see *2.3 The Grabbed Site pages*). Every site uses its own way to compose these url's, but, in most cases, it contains references to the channel and to the timespan for which it is valid. WebGrab+Plus includes an url_index builder that composes this url based on an entry in the SiteIni file with the following syntax:

```
url_index{url|stringfragment-1|stringfragment-2| … |stringfragment-n}
```

- url : just an indication of the type of data that follows, (argument debug supported)
- stringfragment: a fragment of the urlstring for the position n. It can be either a fixed string fragment (independent from channel, date or subpage) like http://www.tvgids or one of the 3 types of variable string fragments: channel or urldate or subpage
- channel : The reference to the channel for which the url is meant. WebGrab+Plus uses the value of the site_id attribute of the channel table in the WebGrab++.config.xml file. Most sites use a simple channel number as site_id but some use rather complicated constructions. (e.g. TvGids.nl uses a number 1 for Nederland1, while Skynet.be uses nederland-1?channelid=216 for the same). For most sites a channel list file is provided together with the SiteIni file.
- urldate : The reference for the timespan or start date. Most websites have one index page per day. WebGrab+Plus supports this per day timespan style.
  The program also supports multiday e.g. weekly index pages (see also *4.3 General Site dependent data* , maxdays). In that case the urldate can specify a start day.
- Some other sites however have (occasional) index subpages e.g. when the number of shows of that day exceeds the space of the displayed webpage. In that case the subpage reference has be specified:
  subpage : Specifies eventual subpages part of the Url. See *4.4.2.2 subpage format*

4.4.2.1 *urldate* format

The day string that appears at the position of –urldate– depends on the value of a separate SiteIni specification urldate, its syntax:

```
urldate.format {daycounter|todaynumber} or
urldate.format {weekdaynumber|Sunday-number} or
urldate.format {weekdayname|Monday-name|Tuesday-name|…|Sunday-name} or
```

urldate.format {datestring|string|optional cultureinfo} or
urldate.format {datenumber|standard|offset} or
urldate.format {list|day1string|day2string|..|daynstring|{urldate format for following days}}

- *datestrings* follow the .Net standard for datestrings as found in : *http://msdn.microsoft.com/en-us/library/az4se3k1.aspx* and *http://msdn.microsoft.com/en-us/library/8kb3ddd4.aspx*
- When the *cultureinfo* used for the datestring is different from the one given in the site specification (*4.3 General Site dependent data*, cultureinfo) it can be added as option. Assume cultureinfo=nl-NL for the following examples.
- *list* method is to be used when the Site uses a value like –today– for today rather than a date value. It fills the url with these day strings, eventually followed by whatever urldate format specified for the remaining days.
- *datenumber* method returns a number that represents a date-time value. It supports the following standards : VBA , the daynumber as used in MS Office ; UNIX , the number of seconds from 1970/1/1 00:00 UTC ; JAVA , the number of milliseconds from 1970/1/1 00:00 UTC ; TICKS , the number of 100 nanoseconds units from 00/00/00 00:00UTC . The offset specifies a time offset w.r.t. to datenumber value. It must be entered as hours like 5:30 or 5.5
- *weekdayname* method is to be used when the daystring required is a non standard weekday name that cannot be generated by the datestring method.

Some examples to illustrate:
urldate.format {daycounter|0} * output like: 0 1 2 ....
urldate.format {weekdaynumber|0} * suppose today is Tuesday, output like: 2 3 4 ..
urldate.format {weekdayname|lu|ma|mi|ju|vi|sa|do} * suppose today is Wednesday, output like: mi ju vi ..
urldate.format {datestring|yyyy/MM/dd} * output like: 2010/05/10 2010/05/11 ...
urldate.format {datestring|dddd} * output like: dinsdag woensdag ...
urldate.format {datestring|d} * output like: 10-5-2010 11-5-2010 ...
urldate.format {datestring|d|en-GB} * output like: 10/5/2010 11/5/2010 ... (other culture, other standard)
urldate.format {datestring|ddd/dd/MMM/yyyy} * output like: ma/10/mei/2010 di/11/mei/2010 ...
urldate.format {datestring|ddd/dd/MMM/yyyy|en-GB} * output like: Mon/10/May/2010 Tue/11/May/2010 ... (other culture, other standard)
urldate.format{datestring|dddd-dd-MM-yyyy} * output like: maandag-10-05-2010 dinsdag-11-05-2010 ...
urldate.format{list|vandaag|morgen|{datestring|dddd|nl-NL}} * output like (if today is Monday): vandaag morgen woensdag donderdag ...
urldate.format{list|Today|{datestring|d|en-GB}} * suppose today is 9/5/2010 , output like: Today 10/5/2010 11/5/2010 ...

4.4.2.2 *subpage* format

subpage.format{number(format=xx)|leadstring|first page number|stopstring} or
subpage.format{letter|leadstring|first page letter|stopstring} or
subpage.format{list|subpage-1-string|subpage-2-string|..|subpage-n-string} or
subpage.format{list(format=xx step=stepsize count=countnumber)|startvalue}

- *leadstring*: fixed part of the subpage string.
- *stopstring*: The unique string that occurs on the subpage after the last one valid. When a subpage is specified in the index_url specification, the program will automatically step from one page to the next until the stopstring is detected. After that the same subpage stepping will start for the next day. If the stopstring is not detected the stepping will stop after 8 subpage tries with a subpage warning and try the next day.
- *startvalue* : an integer for the first subpage value.
  This startvalue may be expanded from a global variable like 'index_variable_element' (see *4.5.3 Special elements*)
- *stepsize*: an integer number by which the startvalue will be incremented
- *count* : an integer number that determines the number of subpage values in the list.
- *format* : optional, specifies the number format. Default xx=D0
- *startvalue* : step and count may contain element references from 'index_temp_1 to 9' and/or a global element (see *4.5.3 Special elements*) like 'index_variable_element'

Some examples of to illustrate:
subpage.format {number||1|<p>page not found</p>} * output for subsequent pages: 1 2 3 ..
subpage.format {number|section_|1|page not found} *output: section_1 section_2 section_3 ..
subpage.format {letter|p|a|page not found} * output: pa pb pc ...
subpage.format {list|04:00|12:00|20:00} * output: 04:00 12:00 20:00
subpage.format {list(format=D2 step=6 count=4)|0} * output: 00 06 12 18

4.4.2.3 Full examples of the url_index specification

Suppose WebGrab++.config.xml channel entry:

```
<channel update="i" site="tvgids.nl" site_id="1" xmltv_id="NED1-tvgids">NED1</channel>
```

And the url_index and urldate.format entries in the SiteIni:

```
url_index{url|http://www.tvgids.nl/zoeken/?q=&d=|urldate|&z=|channel|&t=0&g=&v=0}
urldate.format {daycounter|0}
```

for 3 days will result in:

```
http://www.tvgids.nl/zoeken/?q=&d=0&z=1&t=0&g=&v=0
http://www.tvgids.nl/zoeken/?q=&d=1&z=1&t=0&g=&v=0
http://www.tvgids.nl/zoeken/?q=&d=2&z=1&t=0&g=&v=0
```

Another example:

```
<channel update="i" site="skynet.be" site_id="nederland-1?channelid=216" xmltv_id="NED1-skynet">NED1</channel>
url_index{url|http://www.skynet.be/entertainment-nl/tv/kanalen_|channel|&new_lang=nl&date=|urldate}
urldate.format {datestring|yyyy-MM-dd|nl-BE}
```

for 3 days will result in :

```
http://www.skynet.be/entertainment-nl/tv/kanalen_nederland-1?channelid=216&new_lang=nl&date=2010-06-22
http://www.skynet.be/entertainment-nl/tv/kanalen_nederland-1?channelid=216&new_lang=nl&date=2010-06-23
http://www.skynet.be/entertainment-nl/tv/kanalen_nederland-1?channelid=216&new_lang=nl&date=2010-06-24
```

And one using the list method:

```
<channel update="i" site="tvgids.upc.nl" site_id="Nederland+1" xmltv_id="NED1-upc">NED1</channel>
url_index {url|http://tvgids.upc.nl/TV/Guide/Channel/|channel|/|urldate}
urldate.format {list|Today|Tomorrow|{datestring|dddd|en-GB}}
```

today being Tuesday, for 3 days will result in :

```
http://tvgids.upc.nl/TV/Guide/Channel/Nederland+1/Today
http://tvgids.upc.nl/TV/Guide/Channel/Nederland+1/Tomorrow
http://tvgids.upc.nl/TV/Guide/Channel/Nederland+1/Thursday
```

A subpage example:

```
<channel update="" site="plus.es" site_id="PLAYDC" xmltv_id="PlayDisney">PlayDisney</channel>
url_index{url()|http://www.plus.es/guiatv/resultados.html?tipo=dh5&frm=B&dia=|urldate|&c%5B%5D=|channel|&f=TO&pr=L|subpage}
urldate.format {datestring|yyyy-MM-dd}
subpage.format {number|&pag=|1|No hay ningún título que cumpla con las condiciones de la búsqueda}
```

Supposing this channel has 2 subpages this will result in :

```
http://www.plus.es/guiatv/resultados.html?tipo=dh5&frm=B&dia=2010-09-30&c=PLAYDC&f=TO&pr=L&pag=1
http://www.plus.es/guiatv/resultados.html?tipo=dh5&frm=B&dia=2010-09-30&c=PLAYDC&f=TO&pr=L&pag=2
http://www.plus.es/guiatv/resultados.html?tipo=dh5&frm=B&dia=2010-09-30&c=PLAYDC&f=TO&pr=L&pag=3
```

(detects the stopstring —> step to next day)

```
http://www.plus.es/guiatv/resultados.html?tipo=dh5&frm=B&dia=2010-10-01&c=PLAYDC&f=TO&pr=L&pag=1
http://www.plus.es/guiatv/resultados.html?tipo=dh5&frm=B&dia=2010-10-01&c=PLAYDC&f=TO&pr=L&pag=2
http://www.plus.es/guiatv/resultados.html?tipo=dh5&frm=B&dia=2010-10-01&c=PLAYDC&f=TO&pr=L&pag=3
```

(detects th e stopstring —> stop, last page of last day)

4.4.3 url_preload

See also *4.4.1.2 argument preload*

An alternative way to specify this url is as a separate entry. Its syntax follows that of url_index :

```
url_preload{url|stringfragment-1|stringfragment-2| … |stringfragment-n}
```

See for the detail explanation of the parts *4.4.2 url_index* . The url uses the same Url builder , headers and the same variables (like urldate, channel etc) are permitted. An example:

```
url_preload {url()|http://www.csfd.cz/televize/?day=|urldate|&do=stationSelectForm-submit}
url_preload.headers {method=POST|allowautoredirect=false}
url_preload.headers {customheader=Accept-Encoding=gzip,deflate}
url_preload.headers {postdata=station0='channel'&station1=&station2=&station3=&station4=&stationNew=}
```

4.4.4 other url elements

The next important url is the one that points to the show detail page and possibly the show subdetail page. Normally there is some kind of hyperlink with a *<a href=* tag that points to it but not always the complete url is found there.
The SiteIni specification for url's other than the url_index (above) allows to add the missing components if necessary:

```
url-elementname {url(optional arguments)|leadstring|separator strings}
```

- *url-elementname* : Can be either index_urlshow, (index_)urlsubdetail and index_urlchannellogo (optional)
- *url* : just an indication of the type of data that follows, (argument debug supported)
- *leadstring* : The invariable part of the url that sometimes misses from the html link
- *separator strings* . As explained in *4.2.1.1 The 'separator strings' method*. The regular expression method is not supported here.

Example for show Max Geheugentrainer on site tvgids.upc.nl
index_urlshow {url|http://tvgids.upc.nl|<a href="||">} results in:
http://tvgids.upc.nl/TV/Guide/Programme/9184772/MAX_Geheugentrainer/Nederland+1/
And an example without a leadstring of the same show on site skynet.be:
index_urlshow {url||<a href="||">} results in:
http://www.skynet.be/entertainment-nl/tv/tv-gids/detail_max-geheugentrainer?programkey=MagnetMedia__11435188
!!Notice that the 'empty' leadstring || is required!!

4.4.4.1 multiple subdetail pages

Sometimes when a subdetail page is used by a site the data is split over more than one subdetail page. In that case it is possible to define multiple urlsubdetail elements. (For this we use operations, action specifier 'modify', which will be discussed in *4.6 Operations* in detail.) The program will grab each of these pages and add all of them in one result. The subdetail elements can be grabbed from that combined page . Example of multiple urlsubdetail element specification:

```
urlsubdetail.modify {addend|'index_urlshow'takepart.html}
urlsubdetail.modify {addend|####'index_urlshow'members.html}
urlsubdetail.modify {addend|####'index_urlshow'comments.html}
urlsubdetail.modify {replace()|####|\|}
```

The last line replaces the element separator placeholder #### by the | character that is used the program to separate multivalue element values. (see *4.6.1.3 Multiple value elements and modify*, for explanation about this element separator character |)

4.4.5 the FTP and File protocol

4.4.5.1 FTP

The program also supports the FTP protocol. The configuration is very simple, just replace *http://* in the url element specification by *ftp://.* The program will automatically configure a FtpWebrequest .
Example:
url_index{url|ftp://ftp.pop-tv.si/|channel|_|urldate|_####_EPG.XML}
If the site requires user identification by means of a name and password , issue a credentials header:
url_index.headers {credentials=name, password}

4.4.5.2 File

Mainly for testing and debugging it can be useful to get a local file into the program. Like this:

url_index {url|file:\\computername\path\filename} or

url_index {url|\\computername\path\filename}

4.5 Elements

All the supported elements are listed in *APPENDIX E*. In it, the SiteIni name, the xmltv target element name and an overview of its properties.
There are:

- the elements scrubbed from the index page, element name prefix: *index_*
- the elements scrubbed from the show-detail page, *no element name prefix* or: *detail_*
- the elements scrubbed from the sub-detail page, element name prefix: *subdetail_*

The program allows most elements with the same xmltv target element, to be scrubbed from any or all of these three html pages. If the program finds more than one scrubstring for a certain element any result will be added together in

a way that depends on the if it concerns a *multiple value xmltv* element or not (as discussed in *1.3* and *4.2.4.2 types single and multi*)
All scrubstrings of these sections follow the syntax as explained in section *4.2 The SiteIni file basics* with the action specifier *scrub* , using either/or the *separator string* method or the *regular expression* method.

4.5.1 Non optional elements - elements needed by the program

See for more details the element name table in *APPENDIX E*, unchecked column *optional*

- index_showsplit
  This is not a regular xmltv element, (it has no xmltv target name), but is required for the proper operation of the xmltv update process.
  As explained in section*2.The grabbing, show update process and update modes*, WebGrab+Plus uses the show data on the index page to determine if an update of the xmltv file is necessary for that particular show. To do that it first splits the index page into parts, one for each show. For that it needs the – index_showsplit - scrubstring that returns all those show parts as result.
  Normally, the index_pages list the shows of one day in ascending start-time order. Some sites however list shows of several days on one page (multiday index_page, see also *4.3*, *maxdays*). As a complication it occurs that the shows on these multiday index_pages are not listed in pure ascending start-time order but in day section fragments (like morning, afternoon, evening etc.), e.g. first all morning shows of all days followed by all afternoon shows of all days etc.
  The program provides two options to sort the shows on this type of multiday index pages:
- First option: With the special attribute *sort@.* This will sort the shows in ascending time order. It can be used if the division into the day section fragments occurs at fixed times of the day (e.g. the evening section always start at the first show after 20:00). In that case enter :
  index_showsplit.scrub {type(sort@time-1, time-2, ..,time-n)|. . . .}
  It can also be used if the division always occurs at — or immediately after - the next full- or half-hour following the last show of the previous day section. In that case enter for full hour :
  index_showsplit.scrub {type(sort)|. . . .} or
  index_showsplit.scrub {type(sort@fullhour)|. . . .}
  and for half-hour:
  index_showsplit.scrub {type(sort@halfhour)|. . . .}
- Second option: By scrubbing the each day section fragment on the index_pages separately. Like , if the index page is split in a day-section and an evening-section :
  index_showsplit {type(optional timespan=hours)|day-section-scrubstrings}
  index_showsplit {type(optional timespan=hours)|evening-section-scrubstrings}
  Each scrub will result in an array of shows and the resulting arrays will be merged into one in ascending start time order. The program will attempt to determine the time structures of the arrays automatically. In some cases it can help to specify a *timespan* attribute, which is the approximate duration of each day section. E.g. if the day-section is from 05:00 to 18:00, specify timespan=13:00 and if the evening/night-section is from 18:00-05:00, specify timespan=11:00
  Adding the attribute *debug* during the development phase will show the result of the sorting in the log file.
  Experience learned that the order in which the shows in the index pages occur can have nearly every imaginable form. The following 'specials' have occurred (and are handled by the program):
  - overlaps between subsequent index pages. The duplicate shows are normally removed automatically by the program, but only if the remaining of the pages are in the normal 'starttime' ascending order. If not, one can use the command cleanup(removeduplicates) , see *4.6.4.6.1 Cleanup with argument removeduplicates*
  - multi day index pages with an order of fixed timeframes (in which more that one show can start), spread over the days in a regular pattern. Here the command substring(type=element)|start length/repeat can be used to rearrange the timeframes and the shows in it. See *4.6.4.3 Substring*
  - fully random or reverse order of the shows in the index page.
    Use the command sort(ascending/descending,string/integer). See *4.6.4.9 Sort*
  - a mix of index shows of different channels. Use the command select|string operator to filter the shows to keep. See *4.6.4.8 Select*
- index_start , xmltv element *–start-*
  Scrubs the start time of the show, essential for both the xmltv output file as for the update decision process of the program. See also *4.5.2.1 Time elements index_start*, *index_stop* and *index_duration*.

- index_title *, compared with* xmltv element *–title-*
  The show title is found on two places in most sites, on the index page and on the show detail page. The latter is the most accurate and is used by WebGrab+Plus for the xmltv element *-title– .* The *index_title* is used (and essential) for the xmltv update decision process.
  Because some sites have 'varying' differences between both show titles and because WebGrab+Plus compares the index-title with the title in the existing xmltv listing (which originates from the show detail page), a one to one comparison is not possible due to these differences. WebGrab+Plus uses a smart-comparison which results in a titlematchfactor as detailed in *2.1 The show update process*
  If a site lists a combination of elements together with the *index_title* (like es *category, title, subtitle* ee or other combinations including the title) it is best to scrub them with a separator argument without include or exclude arguments! This yields all these elements in the *index_title*. The title comparison is smart enough to find the real title within this combination.
  In rare cases, a site has no comparable titles on index— and detail page. In such cases the title comparison can be disabled by specifying *titlematchfactor=0*

4.5.2 Elements that are processed in a special way

4.5.2.1 Time elements

Recommended reading: *APPENDIX B Times, time-zones and DST corrections* !

- index_date (optional), part of xmltv element -*start*- and -*stop*-
  Scrubs the date from the first index page which is the date of the first day of the timespan for which the shows will update. If no index_date scrubstring is entered in the SiteIni file the date of '*today*' is taken. By default the scrubbed date value is only used as a check if the first grabbed index page is indeed from '*today*'. If not, the scrubbing of the shows will be stopped with an error message. When the dedicated argument *force* is added to the scrubstring, the scrubbed date value will be used as the date of the first show on the index page. The date of the following days is calculated by the program.
- index_start, xmltv element *–start-*
  The value of this element (and also the next one, index_stop), may contain a full date/time or just a time alone. If it contains just a time alone, (e.g. when the index_page doesn't provide a date for the show to be scrubbed) the date is added automatically as described above in index_date.
  When a date component is present, obviously, adding a date is not necessary. Also, any value of index_date is ignored.
  The scrubbed value is processed by a piece of the program that tries to recognise a time or a date/time in it. It uses the known date/time patterns for the cultureinfo value (see *4.3 General Site dependent data*) and is in general rather forgiving w.r.t small deviations. If the recognition into a data/time fails, the special argument 'pattern' can help. The value of 'pattern' must be enclosed by " " and follows the notations of
  https://msdn.microsoft.com/en-us/library/8kb3ddd4(v=vs.110).aspx
  Some examples:
  index_start.scrub {regex(pattern="yy/M/d H:m")||regular expression||} will recognise a string *15/8/18 0:30*
  index_start.scrub {single(pattern="HH:mm dd MMMM yyyy")|bs|es|ee|be} accepts *06:30 dinsdag 18 augustus 2015*
  Note: that argument *'pattern'* can also be used in *'operations'.* See *4.6.5 Operations (and scrubs) with argument 'pattern'* and *4.6.4.5.3 Date and time calculations*
- index_stop (optional), xmltv element *–stop-*
  Scrubs the stop time of the show. Because not all Sites provide this information (relying on the start time of the next show for it), WebGrab+Plus does the same if no scrubstring for index_stop is entered in the SiteIni file. The resulting value, either from direct scrub or from substitution, is essential for the xmltv update decision process. As with index_start the value can consist of a time only or a complete data/time. It also accepts the special argument 'pattern' to help the recognition of it. See above.
- index_duration (optional)*,* xmltv element *–stop-*
  Alternative for index_stop. Some Sites specify this rather than the stop time. WebGrab+Plus calculates the stop time from stop=start + duration. The scrubbed value is processed to recognise the correct timespan. It accepts the following formats:
  - hh:mm
  - dd:hh:mm
  - minutes (integer)
  (The operations discussed in *4.6 Operations* provide ways to convert it to this format if it isn't)

4.5.2.1.1 Times from the detail page

In rare cases, the showtimes are not listed on the index_page. The program can handle the following of those cases:

- Start time from the index_page and only stop time from the detail_page
  Specify (detail_)stop in the SiteIni will automatically handle this.
- Start time from the index_page and duration from the detail page
  Specify (detail_)duration in the SiteIni
- Both start and stop time from the detail_page
  Specify both (detail_)start and (detail_)stop or (detail_)duration in the SiteIni
  In this latter case, the incremental update mode (see *2.1 The show update process*) is disabled, because the most important value (start time) is missing at the time of the update decision making.

4.5.2.2 The others

- titleoriginal (optional), xmltv element *–title-*
  See also *4.2.5.6 Dedicated Arguments*.
  It is meant to allow multiple titles for different languages. The scrubstring can include the dedicated argument *lang.* The syntax of it is :

lang or lang=xx

  - xx=two letter language spec like *en* for English
    If a *lang* argument is given without a language value or with the value "xx" or if no *lang* argument is added, the title lang attribute in the xmltv will be lang="xx", which is supposed to indicate the 'original' show title in an unspecified language. If a two letter language spec is provided it will use that in the xmltv lang= attribute.
    Example:
    title.scrub {single|….}
    titleoriginal.scrub {single (lang=en)|….}
    could result in something like:
    <title lang="es">Mujeres Desperadas</title>
    <title lang="en">Desperate Housewives</title>
- productiondate (optional), xmltv element *–date-*
  The productiondate should yield the year of the production of the show. Because it is often hidden inside another element, like the description it is rather difficult to find unique element separators. WebGrab+Plus will scrub the first 'year' value (yyyy) between the element separators automatically.
- episode (optional), xmltv element *–episode-num-*
  For this element a dedicated argument 'pattern' can be used to recognise and convert the scrubbed value. The background of that is that the majority of xmltv import utilities of PVR programs expect a certain standard of the episode number. (being used to schedule series recordings). The favorite episode number statndard is *xmltv_ns*. With the help of the 'pattern' argument most custom episode number formats can be converted into xmltv_ns standard or into simple 'onscreen' format. See *4.6.5 Operations (and scrubs) with argument 'pattern'* and *pattern argument*
- *Boolean type elements* (optional) like ,
  subtitles (xmltv element *–subtitles-*),
  premiere (xmltv element *–premiere-*) and
  previousshown (xmltv element *–previously-shown-*)
  These elements have no value in xmltv, they are either listed , like *<premiere/>,* or not listed.
  Note !! The program needs the string value *true* to add a listing to the xmltv file. If this required value cannot be scrubbed directly (because it is listed differently in the html page), use a modify operation to replace the actual value with *true.* E.g. like:
  subtitles.modify {replace(not "")|'subtitles'|true}

4.5.3 Special elements

(see *APPENDIX E*)

- temp elements :
  temp_1 up to temp_9 .
  Available for each of the three prefix versions.
  These special elements have no direct xmltv destination. They can be used to temporary scrub and store data that is later used together with the action specifier *modify* (see *4.2.3 Action specifiers* and *4.6 Operations*) to alter or create other elements.
- Global elements :

- global_temp_1 up to global_temp_9 and index_variable_element .
  These special elements have no direct xmltv destination.
  There is no functional difference between global_temp elements and index_variable_element other than the index_variable_element is limited to the index_ prefix . The global_temp elements are added later to the list of global elements, index_variable_element is maintained because of compatibilty reasons.
  They can be used if any of the other scrubstrings requires a value that varies (e.g. with each channel or day). Their value can be scrubbed (from the un-split index-page) and modified. It is the only element that :
  - can be used in a scrub string as part of separatorstrings bs es ee or be or a regular expression like in :
    global_temp_2.scrub {single|Billing\t\n||\t|\t} * scrubs the value from the index page
    and uses it to split the index pages in shows :
    index_showsplit.scrub {multi|'global_temp_2'||\n}
  - can also be used in *argument* values like: '
    index_showsplit.scrub {multi(includeblock='index_variable_element')|"column">|time">||}
  - allows to pass certain values from the config file with a modify command (see *4.6 Operations* for details about the modify command) like in :
    global_temp_5.modify {addstart|'config_site_id'}
    This line copies the site_id for the actual channel from the channel list in the config file.
    Other supported config values are :
    - the *xmltv_id* entered as 'config_xmltv_id'
    - the *display_name* entered as 'config_display_name'
    - the *site_channel* entered as 'config_site_channel'
    - the *credentials* entered as 'config_credentials_user' and 'config_credentials_password'
    - and *timespan* entered as 'config_timespan_days'
  - allows to pass date/time read-only values *urldate, now* and *showdate.* (see *APPENDIX C Read-only elements*)
  - Its value and function is kept independent of the 'scope' (see *4.6.1.1*).
- index_site_channel and index_site_id :
  When specified, a channel-list file will be created automatically. This is a file that lists the available channels of the site, in a format that can be copied directly into the config file WebGrab++.config.xml. These files are supplied together with the SiteIni file when downloaded from *http://www.webgrabplus.com/epg-channels*. As default, the allocation of the values of these elements in for channel specification in the config file WebGrab++.config.xml will be as follows:
  <channel update="i" site="site" site_id="index_site_id" xmltv_id="index_site_channel">index_site_channel</channel>
  With the dedicated argument *alloc* it is posible to change the allocation of these values. This argument accepts only three values : *site_id, xmltv_id* and *display_name* E.g. (*alloc=site_id,display_name)*
  These elements are scrubbed from the un-split index-page by default. If the channellist data is located on another page the url_index specification must be overruled by specifying the one for this page after the one for the tv-program data. When the channel data if found on more than one page subpages can be used. (see *4.4.2.2 subpage format*).
- sort_by :
  This must be specified when using the command *sort* (see *APPENDIX E* and *4.6.4.9 Sort*).
  It must have the value of that part of the multivalue element by which it is to be sorted. The argument *target* links it to the multivalue element to sort. It can be scrubbed and modified.

4.5.4 Read only elements

(See *APPENDIX C*)
Elements that pass parameter values from various sources in operations. Cannot be scrubbed or modified.

- previous value elements: In these elements the value of the previous scrub is stored. The program automatically stores the values of previous scrub of the following elements: *index_start, index_stop, index_duration, temp_1* to *–9.* These values can be recalled by adding an additional prefix *previous_* to the element names.
  A typical use is when a site displays index_pages graphically, each next show in a horizontal grid, the start and stop times hidden in pixel coordinates, then it is necessary to know the previous value of time elements to calculate the actual start and stop time. (See also *4.6.4.5.3 Date and time calculations*)
- urldate Passes the urldate value (unformatted!) of the url_index with which the actual page is grabbed. The returned value must be formatted to the required format using the format argument (see 4.6 *arguments*).

- now Passes the date and time value (unformatted!) of the actual moment. The returned value must be formatted to the required format using the format argument (see 4.6 *arguments*).
- showdate Passes the epg date value (unformatted!) of the actual show being grabbed. Warning! It is obvious that this will only return a value after the start time of the show is grabbed! The returned value must be formatted to the required format using the format argument (see 4.6 *arguments*).
- config_site_id Passes the value of the site_id of the channel as specified in the config file.
- config_site_channel Passes the value of the site_channel of the channel as specified in the config file.
- config_xmltv_id Passes the value of the xmltv_id of the channel as specified in the config file.
- config_display_name Passes the value of the display_name of the channel as specified in the config file.
- config_timespan_days Passes the value of the days component of the timespan in the config file
- config_credentials_user Passes the username of the credentials for the actual site in the config file
- config_credentials_password Passes the password of the credentials for the actual site in the config

4.5.5 XMLTV attributes

These attributes can be added to a xmltv element using the following syntax:

```
elementname.format {xmltv|"element-attribute pattern"}
```

This scrubstring splits the element value into the xmltv attribute value and the xmltv element value. Prior to issuing this, the element must have gotten a value that matches the 'element-attribute pattern' using the methods described in the previous chapters. If necessary, followed by any operation (see *4.6 Operations*).

- !!! This feature is only supported for the actor xmltv element and its role attribute.!!
- Element-attribute pattern: Must be a string that is composed out of two keywords representing the *xmltv element name* and the *xmltv attribute name* and any strings that separate them matching the pattern of these string parts in the elementname value.
- Action specifier format and type xmltv are needed to select this feature.

Example: Suppose the actors and their role are listed like this
Kenneth Branagh (Kurt Wallander) in the index page and the index_actor element has this value through prior scrub actions. The pattern here obviously is "actor (role)". To split these into the xmltv element and its attribute:
index_actor.format {xmltv|"actor (role)"}
The resulting xmltv listing will be:
<actor role="Kurt Walander">Kenneth Branah</actor>

4.6 Operations

Action specifier: *modify*
With the operations described in this section it is possible to modify already scrubbed elements and/or obtain a value by other means than scrub from a (html) page. The elements for which these modifications are supported are listed in *APPENDIX E Element names*, column *action*. The syntax to specify such a modification:

```
elementname.modify {commandname(optional arguments)
|optional expression-1|optional expression-2}
```

- *element name* : Any of the elements listed in *APPENDIX E* for which the action specifier modify is allowed.
- *modify* : the action specifier for this type of operation
- *commandname*, either :
  - replace   : replaces the value of expression-1 with that of expression-2 (see *4.6.4.1 Replace*)
  - remove   : removes the value of expression-1, no need for expression-2 (see *4.6.4.2 Remove*)
  - substring : extracts a part of expression-1, no need for expression-2 (see *4.6.4.3 Substring*)
  - addstart   : adds the value of expression-1 to the start of the element, no need for expression-2 (see *4.6.4.4*)
  - addend   : adds the value of expression-1 to the send of the element, no need for expression-2 (see *4.6.4.4*)
  - calculate : performs a set of calculations, expression-1 is an arithmetic expression, no need for expression-2 (see *4.6.4.5 Calculate*)
  - cleanup   : tidying up of elements, no expressions (see *4.6.4.6 Cleanup*)
  - clear   : to erase the content of any element (see *4.6.4.7 Clear*)
  - select   : to select certain members of multi value elements (see *4.6.4.8 Select*)
  - sort   : to sort multi value elements (see *4.6.4.9 Sort*)
  - set   : gives an element a value irespective of its current value. (see *4.6.4.10 Set*)
- *arguments* :
  - conditional arguments: There are two possible sets of conditional arguments

    - Pre-conditions that needs to be true for the operation to be performed. They are evaluated first. It allows to evaluate the value of any element or compare element values with constants or other elements. *(4.6.2.1 Pre-Conditional arguments)*.
    - Post-conditions, simple condition that only evaluate the value of the element to be modified before or after the operation. *(4.6.2.2 Post-Conditional arguments)*
- debug : Adding the word debug as argument will start logging of the modify process for the element in the WebGrab++.log.txt file.
- format : Specifies the output format for the calculate command. (see *4.6.4.5 Calculate*)
    - *Numeric formats*: Supported values are all the standard numeric format strings F and D, as described in *http://msdn.microsoft.com/en-us/library/dwhawy9k.aspx* like format=F0.
      Default is F2 (two decimal digit fixed point, like 16.35).
    - *Converting formats*: Performs a conversion.
      format=chartodec Returns the decimal presentation of a char. If the input element is longer that 1 characterit will concate all converted chars into one long string.
    - *Date and Time formats*: Also supported is format=time, format=date , this will convert the numeric value in HH:mm and yyyy/MM/dd respectively as default date-time format. For other formats, add a comma and date-time format string after the word time or date, like format=time,h:mmtt or format=date,dd/MMMM/yy
      See *http://msdn.microsoft.com/en-us/library/8kb3ddd4.aspx* for date/time format strings.
    - *Extra Date formats*: Besides these standard date formats the following numeric date formats are supported:
        - format=date,vba Returns the excel-vba day-number, as used in MS-Office
        - format=date,unix Returns the number of seconds from 1970/01/01 00:00 UTC
        - format=date,java Returns the number of milliseconds from 1970/01/01 00:00 UTC
        - format=date,ticks Returns the number of 100 nanosecond units fom 00/00/00 00:00 UTC
        - Adding utc as prefix, like format=utctime or utcdate will return the utc time or date (ignoring the timezone utc offset)
    - *Timespan format*: allows conversion and calculations of timespans. It must be entered as format=timespan. The default format is in days and will result in a string d:h:m
      When specified as format=timespan,hours a string h:m is the result.
    - Finally there is format=productiondate. When entered it will return the first 4 digit numeric value it finds in string-1 that is between 1900 and 'nextyear'.
      Example: productiondate.modify {calculate(format=productiondate)|'description'}
- type : When expression-1 is specified by means of place-*indices* and command is *remove*, *replace* or *substring*, it specifies the index-base. (see *4.6.1.4 Expression-1 with indices*).
  It is also used with command *calculate* - *index-of* (*4.6.4.5.2*) and *count* (*4.6.4.5.1*). Possible values are :
    - type=string (default, expression-1 is specified as string, no indices)
    - type=char (the indices specify character positions or length)
    - type=word (the indices specify word positions or length)
    - type=sentence (the indices specify sentence positions or length)
    - type=paragraph (the indices specify the paragraph position or length)
    - type=element (the indices specify element positions or length in case of multi value elements, see *4.6.1.3*)
    - type=regex must be used to indicate the use of a regular expression in expression-1
      (see expression-1 and –2 components further down in this chapter)
    - type=run (can be used with command *set* to run an external program, see *4.6.4.10.1*)
- separator : This specifies a separator string value that is used when converting multi value elements to a combined single value. It can be used together with the commands *remove*, *replace*, *addstart* and *addend*.
  If this argument is entered, the operation will try to convert the result in a single value string, adding the separator string between the multi value components. (see *4.6.1.3 Multiple value elements and modify*)
- style : This argument can be added to the cleanup command (see *4.6.4.6 Cleanup)* to specify the required style of the cleanup result. Possible values are :
    - style=sentence (converts 'this is a sentence.' In 'This is a sentence.'
    - style=name (converts 'john do' in 'John Do')
    - style=upper (converts to UPPERCASE)
    - style=lower (converts to lowercase)
      Beside these basic styles , the specials:
    - style=regex, formats the input as a regular expression, inserts the required escape codes

        - style=urlencode and style =urldecode , formats the input string as an url string or the reverse
        - style=uniencode and style=unidecode , converts unicoded character sequences to the actual chars and reverse.
        - style=htmlencodespecialchar and style=htmldecodespecialchar
        - style=jsondecode, converts json encoded strings back to normal
        - style=base64encode and style=base64decode , encodes to – decodes from a base64 encoded string. base64decode accepts a format specifier, decimal, like style=base64decode,decimal With it the output is a multivalue string of decimal numbers of the decodes characters. See *4.6.4.6.3 Cleanup as string converter*
        - style=sha256encode, style=md5encode and style=base64encode provides encoding for encrypted internet communications.
    - removeduplicates : To be used with command cleanup to remove possible duplicates from multi value elements. (see *4.6.4.6.1 Cleanup with argument removeduplicates*)
    - link : Links elements when used together with removeduplicates in command cleanup
      See *(4.6.4.6.1 Cleanup with argument removeduplicates)*
    - tags : To be used with command cleanup to remove strings enclosed by specified start– and end- strings.
      See *(4.6.4.6.2 Cleanup with argument tags)*
    - keepfirst or keeplast: Specifies which of set of duplicate elements to keep in a cleanup removeduplicates operation.
    - pattern : specifies how the value of certain elements must be interpretated to faciltate recognition and conversion. See *(4.6.5 Operations with patterns)* for details.
      Comes in two forms :
        - *date time* pattern See *(4.6.5.1 Date-time patterns)* Can only be used for the time elements (index_ detail_) start and stop . See also *(4.5.2.1 Time elements) .* Can be used to recognise and convert 'non standard, custom' datatime values into the standard that belongs to the *cultureinfo* value as in *(4.3 General Site dependent data)*
        - *episode number* pattern See *(4.6.5.2 Episode-number patterns)* Only works for element (index_ detail_ subdetail_) episode. Can be used to convert 'non standard' episode data into the standard selected by the *episodesystem* value as in *(4.3 General Site dependent data)*
- *expression-1* and *-2 components*, these expressions can be composed with:
    - *text* : all characters with exception of | { and '
      If any of these are needed in the string they have to be preceded by the backslash character \, like O'Neil must be entered as O\'Neil
    - *element* : to be entered between ' ' , like 'title' or 'temp_1',
      the value of the element will be inserted in the expression result
    - *scrubstring* : to be entered between '{ }' like '{single|<a ref=|<p>|</>|<table}'
      The use of a scrubstring in this way has a small limitation: The scrub is performed from the same html page as from which the element is originated. So, in case of an element from the index page the scrub entered here is also done from the index page.
      In most cases it is easier and more flexible to use the 'temp' and 'index_temp' elements instead.
      See also *4.6.1.1* for limitations.
    - *indices* : Expression-1 only.
      Can be used to *extract* (with commands *substring* and *replace*)
      or *remove* (with command *remove)* parts of a source string.
      The use of indices must be accompanied by the argument *type* to specify the index-base (see above; arguments). Indices must be entered as (integer) numbers. Each of these numbers can also be entered as element enclosed by ' '. The first number represents the *start* position, the second (optional) number the *length* and the third (optional) the *repeat*. (see for more details *4.6.1.4 Expression-1 with indices*)
    - *regular expressions* : Expression-1 only.
      As with indices (see above) these can be used together with the commands *substring, replace* and *remove.* To initiate the use of it they must be accompanied by argument *type=regex.* For more information about the use of 'regular expressions' see *4.2.1.2 The 'regular expression' method* and *4.6.1.5 Expression-1 with 'regular expressions'*
    - *arithmetic expression* in the case of command calculate (see *4.6.4.5 Calculate*).
    - *combinations* of text, element, scrubstring, indices and arithmetic expression.
      See *4.6.4 The modify commands* for examples.

4.6.1 Notes and examples of the effects of *modify*

4.6.1.1 The order of the actions and the argument *scope*.

WebGrab+Plus executes the scrubbing and modifying of the elements in a certain order. Some of these steps have a named reference called *scope*, which purpose will be discussed later in this section. Roughly the order of actions is like this:

1. *scope=urlindex*, compose/modify the url_index and grab the index-page(s)
2. *scope=datelogo*, scrub/modify index_date, global elements and index_channellogo
3. *scope=splitinde*x, split/modify the index-page(s) in index shows
4. Step through the index shows one by one
5. *scope=indexshowdetails*, scrub/modify all other index_ elements from the index show.
6. Update decision. If no update - - back to 4, next index show
7. if: it is an index_only channel or: if no valid urlshow is scrubbed,
   back to 4, next index show
   or else:
8. grab the show-detail page
9. *scope=showdetails*, scrub/modify all show-detail elements
10. if: a valid urlsubdetail is scrubbed,
11. grab the sub-detail page
12. *scope=showsubdetails,* scrub/modify all sub-detail elements
13. compose the xmltv elements and write them to the xmltv output file
14. if more shows, back to 4, next index show. Else: next channel.

    Not in this order, but with its own scope:

0. *scope=channellist*, scrub/modify a channellist file

- The order in which the scrubbing of the elements is done (in actions 2. 5. 9. 12. and 0.) is fixed by the program, not important for the results and independent of the order of the scrubstrings in the SiteIni file.
- The order in which the modify operations are done (in actions 2. 5. 9. and 12.) is determined by the order in which the modify operations (for that group : scope) are listed in the SiteIni file. E.g. in 5. , the modification of all index_elements other than index_date, global elements and index_channellogo is done. It will modify these in the order they occur in the SiteIni file.
- The range of actions for which an operation of elements is executed is called *scope.*
- Some general supporting elements , like the index_temp and global elements (see *4.5.3 Special elements*) have a wider scope (*urlindex, datelogo, splitindex* and *indexshowdetails*), because they are used as supporting elements for others. Because of this operations specified for these supporting elements will be executed at all these actions even when the operation is meant only for one of them. This can lead to unexpected results and unnecessary consumption of processing time. To avoid this the argument *scope* can be added to the operation (see *4.6.1.2 The use and effect of argument scope*).

It is important to realise that this order will *influence the result* and also *poses restrictions* on the use of other elements in the modify operation. For the influence on the *result* consider the following:

Suppose: *description = A short story*

Case 1.

temp_1.modify {calculate(format=F0)|'description' " " #}	result temp_1 = 2
description.modify {remove|short }	result description = A story

Case 2.

description.modify {remove|short }	result description = A story
temp_1.modify {calculate(format=F0)|'description' " " #}	result temp_1 = 1

These simple cases illustrate that modify operations work on bases of the results of previous modify operations. This also explains the *restrictions*: Operations that try to use elements that have no value (as yet) will not work. Like trying to use (non index_) elements in 5. scope=indexshowdetails. That can't work because these elements are not yet scrubbed, that occurs later in 9. scope=showdetails

Consider the following:

A site has only show-detail links for a limited number of shows. We scrub both *index_description* and *description* to get a description in both cases. That could create a double description in case of a show with a show-detail link. So, we would like to erase the index_description if the description is not empty. It is only logical we try this:

index_description.modify {remove('description' not "")|'index_description'}

That, unfortunately, will not work because this operation is done in 5. and uses an element (description) that is only available after 9. A way to solve this particular case is :

index_description.modify {remove('index_urlshow' not "")|'index_description'} (It tests for the show-detail link *index_urlshow* to exist which value is available in 5.)

4.6.1.2 The use and effect of argument *scope*

As explained in *4.6.1.1* the scope of an operation is the (range of) steps for which the operation is executed. By adding the argument *scope* to the specification of the operation the scope can be narrowed to a certain (just one) step. The syntax:

```
element.modify {command(scope=string other-arguments)|..}
```

If a group of subsequent operations in the SiteIni should be set to the same *scope*, *scope.range* can be used. The syntax of that:

```
scope.range {(string)|end}  combined with end_scope
- or -
scope.range {(string)|lines}
```

- *string* : one of the following (see *4.6.1.1*) :
  - *urlindex*
  - *datelogo*
  - *splitindex*
  - *indexshowdetails*
  - *showdetails*
  - *showsubdetails*
  - *channellist*
- *end* : the word *end* indicates that the program expects end_scope after the last operation that belongs to the group for which the scope have to be set.
- *alternatively* the number of *lines* (containing operations with the same scope) can be specified.

4.6.1.3 Multiple value elements and modify

Some explanation about the internal handling of elements: Most elements can have more than one value, either through separators, through multiple scrubs or by being a multi type scrub. Internally they are not stored as array <u>but as a combined string with the | character as separator</u>. Thus, an element with the values AAA BBB ccc ddd will have the internal representation AAA|BBB|ccc|ddd It depends on another element property , multiple xmltv value, true or false, how these values will be written to the xmltv file. If true (multiple), they will get multiple xmltv elements like:

<element>AAA</element>
<element>BBB</element> etc.

If false, they will be added together, separated by a period-space, like:

<element>AAA. BBB. ccc. ddd.</element>

TABLE 1

Multi Value Elements examples:					
	element-to-modify.modify{command(argument)|expression-1|expression-2}				
element-to-modify	*element*	*command*	*argument*	*|expession-1 |*	*result*
				expession-2	
Abc def.	Ghi|Jkl|Mno	addstart		|'element'. |	Ghi|Jkl|Mno. Abc def.
Abc def.	Ghi|Jkl|Mno	addstart		|'element'\\\||	Ghi|Jkl|Mno|Abc def.
Abc def.	Ghi|Jkl|Mno	addstart	separator=" & "	|'element'. |	Ghi & Jkl & Mno. Abc def.
Abc def.	Ghi|Jkl|Mno	addend	separator=", "	|'element'. |	Abc def. Ghi, Jkl, Mno.
Abc|Def	Ghi|Jkl	addend	separator=", "	|* 'element'. |	Abc*Ghi, Jkl.def*Ghi, Jkl.
Ghi, Jkl, def, Mno.	Ghi|Jkl|Mno	remove	separator=", "	|'element'. |	def,
- empty -	Ghi|Jkl|Mno	addstart		|'element'|	Ghi|Jkl|Mno
Abc|Def|Ghi	Ghi|Jkl|Mno	remove		|'element'|	Abc|Def
Abc|Def|Ghi	Ghi|Jkl|Mno	replace		|'element'|Xyz	Abc|Def|Xyz
Abc|Def|Ghi	Ghi|Jkl|Mno	replace	separator=" & "	|'element'|Xyz	Abc & Def & Xyz

How does this effect result of *operations?*
For this, the argument *separator* plays a determining role.
In operations, all elements, in any of the expressions-1 or 2, are considered multi value elements (a single value element as a multi value element with just one value). <u>Each value is evaluated for the requested operation individually, one at the time</u>. At the end of this process, when the expression is assembled, the resulting components are 'added' together, separated by the string specified by the *separator* argument. (see *arguments*). This results in two effects:

- If no *separator* argument is entered, or if its value is "\|", the before mentioned multi value separator | is placed between the components. The effect of this is described at the start of this section, it keeps its potentially multi-value nature.
- Any other value of the *separator* argument will combine the components in a single string, with this separator string between them.

4.6.1.4 Expression-1 with indices

Indices in *expression-1* are only supported in combination with the commands *remove, substring* and *replace.* (not with commands *addstart, addend, cleanup* and *calculate)*

TABLE 2

Indices examples:		Red = selected parts			Blue = not selected	
---> with command **substring** and **remove**:			element.modify {command(type=xx separator=xx)|expression-1}			
				expression_1	command *substring*	command *remove*
	element value	*separator*	*type*	*indices*	**result:**	**result:**
single value	Abc def ghi. Jkl mno pqr.		char	2 4 or -23 4	c de	Abf ghi. Jkl mno pqr.
	Abc def ghi. Jkl mno pqr.		word	2 3 or -4 3	ghi. Jkl mno	Abc def pqr.
	Abc def ghi. Jkl mno pqr.		sentence	0 1 or -2 1	Abc def ghi.	Jkl mno pqr.
	Abc def ghi. Jkl mno.		element	0 1 or -1 1	Abc def ghi. Jkl mno.	- empty -
multi start	0123456789		char	0,4,8 2	014589	2367
repeat	0123456789		char	0,3 1/5	0358	124679
multi value	Abc def ghi|Jkl mno pqr|Stu vwx yz		char	2 4	c de|l mn|u vw	Abf ghi|Jko pqr|Stx yz
	Abc def ghi|Jkl mno pqr|Stu vwx yz		char	-9 4	c de|l mn|tu v	Abf ghi|Jk opqr|Swx yz
	Abc def ghi|Jkl mno pqr|Stu vwx yz		word	1 1 or -2 1	def|mno|vwx	Abc ghi|Jkl pqr|Stu yz
	Abc def. Ghi jkl.|Mno pqr. Stu vwx yz.		sentence	0 1 or -2 1	Abc def.|Mno pqr.	Ghi jkl.|Stu vwx yz.
	Abc def ghi|Jkl mno pqr|Stu vwx yz		element	1 1 or -2 1	Jkl mno pqr	Abc def ghi|Stu vwx yz
multi start	Abc def ghi|Jkl mno pqr|Stu vwx yz		word	0,2 1	Abc ghi|Jkl pqr|Stu yz	def|mno|vwx
multi start	Abc def ghi|Jkl mno pqr|Stu vwx yz		element	0,2 1	Abc def gh|Stu vwx yz	Jkl mno pqr
w. separator	Abc|def|ghi|jkl	", "	element	1 1	def, ghi	Abc, jkl
no length	Abc def ghi. Jkl mno pqr.		char	2 or -23	c def ghi. Jkl mno pqr.	Ab
out of range	Abc def ghi. Jkl mno pqr.		char	2 50	c def ghi. Jkl mno pqr.	Ab
out of range	Abc def ghi. Jkl mno pqr.		char	50 2	- empty -	Abc def ghi. Jkl mno pqr.
out of range	Abc def ghi. Jkl mno pqr.		char	-27 5	Abc	def ghi. Jkl mno pqr.
---> with command **replace**:			element.modify {replace(type=xx)|expression-1|expression-2}			
				expression_1	*expression-2*	command *replace*
	element value		*type*	*indices*		**result:**
single value	Abc def ghi. Jkl mno pqr.		char	2 4 or -23 4	x	Abxxxxf ghi. Jkl mno pqr.
	Abc def ghi. Jkl mno pqr.		word	2 3 or -4 3	xyz	Abc def xyz xyz xyz pqr.
	Abc def ghi. Jkl mno pqr.		sentence	0 1 or -2 1	Uvw xyz.	Uvw xyz. Jkl mno pqr.
	Abc def ghi. Jkl mno pqr.		element	0 1 or -1 1	xyz	xyz
multi start	0123456789		char	0,4,8 1	xyz	xyz123xyz567xyz9
repeat	0123456789		char	0,3 1/5	xyz	xyz12xyz*4* *x* yz67xyz9
multi value	Abc def ghi|Jkl mno pqr		char	2 4 or -9 4	x	Abxxxxf ghi|Jkxxxxo pqr
	Abc def ghi|Jkl mno pqr		word	1 1 or -1 1	xyz	Abc xyz ghi|Jkl xyz pqr
	Abc def. Ghi jkl.|Mno pqr.		sentence	0 1 or -1 1	Uvw xyz.	Uvw xyz. Ghi jkl.|Uvw xyz.
	Abc def|Jkl mno pqr|Stu vwx		element	1 1 or -2 1	Xyz	Abc def|Xyz|Stu vwx
multi start	Abc def|Jkl mno pqr|Stu vwx		element	0,2 1	xyz	xyz|Jkl mno pqr|xyz

- Indices in expression-2 are not allowed.
  As explained in 4.6 - arguments, the argument *type* sets the base of indices used in *expression-1.* (see 4.6 *expression_1 and expression_2 components*, indices). These indices specify the position and the length (or number-) of a character (in case of *type=char),* or a word (in case of *type=word),* or a sentence (in case of *type=sentence),* or a paragraph (in case of *type=paragraph*) or an element value (in case of *type=element)* (which only makes sense for *multi value elements*).
- The basic structure and syntax of *expression-1* containing indices is :

element-to-modify.modify {command(type=)|'element' startpos optional-length}

Between 'element' startpos and optional-length a space is needed. If 'element' is the same as the element-to-modify, it can be left out of *expression-1*

element-to-modify.modify {command(type=)|startpos optional-length}

- *length* is optional. If left blank, the length to the end is taken.
- *startpos* can be negative, in that case the position counting from the end backwards is taken.
- *startpos* and/or *length* can also be entered by way of '*element*' in which case the integer value of '*element*' is used. In this case the element-to-modify cannot be left out! (see also *4.6.4.5.1 # Count* and *4.6.4.5.2 @ Index-of*)

- The effect on the value of element to modify can best be described with:

  'substitute the *actual value* with the result of the *expression*'

  This is also the case if the *element-to-modify* is another than '*element*':

element-to-modify.modify {command(type=)|'element' indices}

'The resulting value of the *element-to-modify* will be the result of *expression-1*'

The original value of element-to-modify will be substituted.

- Beside the basic structure and syntax, more complex forms are supported:

|startpos-1,startpos-2,..,startpos-n length/repeat}

and of course

|'element' startpos-1,startpos-2,..,startpos-n length/repeat}

– *repeat* The distance from *startpos-1* at which the sequence is repeated. Default is full length (no repeat)
– multiple *startpos* values must be separated by comma's.
– *startpos, length* and *repeat* maybe entered as 'element'

Table.2 in this chapter illustrates the results of the expressions with indices:

4.6.1.5 Expression-1 with 'regular expressions'

For a short general introduction of 'regular expressions' see *4.2.1.2 The 'regular expression' method*
Similar to indices, general expressions in *expression-1* are only supported in combination with the commands *remove, substring* and *replace* . They are not allowed in *expression-2*.
To indicate that *expression*-1 contains a regular expression the argument *type=regex* is required. A regular expression in an operation must be enclosed by " ".
A regular expression may contain elements, enclosed by ' ' which value may in itself contain parts of - or whole regular expressions, like:

"regex-part'element-containing-a-regex-part'regex-part . . .etc"

Following are possible entries:
For the commands *substring* and *remove:*

element-to-modify.modify {command(type=regex)|"regular-expression"}

If element contains the 'source data'

element-to-modify.modify {command(type=regex)|'element' "regular-expression"}

For command *replace* add expression-2 to any of the entries above, like:

element-to-modify.modify {replace(type=regex)|"regular-expression"|expression-2}

Example: Suppose the title contains the names of the director and a list of actors that must be removed and moved to the appropriate elements:
Title : *The "Pelican Brief" dir.: Alan J. Pakula, starring: Julia Roberts, Denzel Washington*
This will remove director and the actors and places them in where they belong:

```
actor.modify {substring(type=regex)|'title' "(?:starring): *(.*?)(?:, *(.*))"}
title.modify {remove(type=regex)|"((?: starring): *(?:.*?)(?:, *(?:.*)))"}
director.modify {substring(type=regex)|'index_title' "(?:dir\.):\s*(.*?),"}
title.modify {remove(type=regex)|"((?: dir\.):\s*(?:.*?),)"}
```

4.6.2 Conditional arguments

4.6.2.1 Pre-Conditional arguments

These conditions are evaluated first. If *true* the operation is executed. The following conditional *operators* can be used:

=	string, equal ?, ignore (lower or upper) case , this operator is default and the = character can be omitted
==	string, equal ?, match case

~	string, contains ?, ignore case
~~	string, contains ?, match case
not	added to one of the operators above (not= not== not~ and not~~) reverses the result
>	numerical, more ?
<	numerical, less ?
>=	numerical, more or equal ?
<=	numerical, less or equal ?

The syntax :

```
('compare-this-element' operator "to-this")
```

- 'compare-this-element': The element to evaluate. Enter the name of the element enclosed by ''. If omitted, the element-to-modify is taken. In the case of the numerical operators > < >= and <=, the element entered will be converted to a floating point number (the floating point conversion of the first number in the string - or 0 (zero) if it does not contain any numerical value)
- operator : one of the conditional operators listed above
- to-this : A "string" (enclosed by "") or an 'element' (enclosed by ''), in which case it will be expanded to the value of the element. These to-this strings may contain wildcards (see 4.2.6)

A few examples:

('element' "abc") result 'true' if the value of element is abc or Abc etc
('element' == "abc") result 'true' if the value of element is abc, false if Abc etc
('element' ~ "abc") result 'true' if the value of element contains abc or ABC etc
('element' ~~ "abc") result 'true' if the value of element contains abc, false if ABC etc
('element' not ~ "abc") result 'true' if the value of element doesn't contain abc
('element' "") result 'true' if the value of element is "" (empty)
('element' 'other-element') result 'true' if the value of element is the value of other-element
('element' ~ 'other-element') result 'true' if the value of element contains the value of other-element.

Examples when 'element' is left out and the element to modify is taken :

element.modify {addstart('other-element')|abc} 'true' if the value of element is the value of other-element, then abc will be added.
element.modify {addstart(not ~ "abc")|abc} 'true' if the value of element doesn't contain abc, then abc will be added.
element.modify {addstart("")|abc} 'true' if the value of element is "" (empty), then abc will be added.

Example of numerical conditional arguments:

('element' < "10") 'true' if the first numerical value in element is < than 10. E.g. if element = "Episode 9"
Note that these numerical values must be entered as string, enclosed by " " !!

4.6.2.2 Post-Conditional arguments

These conditions will only be evaluated if the pre-conditions are *true* (or left out).

Values : either *anycase, null* or *notnull*

- *anycase (default)* : the operation will be performed regardless any of the conditions described below.
- *null* : the operation will only be performed if the element is *empty* (in the case of addstart and addend commands)
- *notnull* : the operation will only be performed if the element is *not empty* in the case of addstart and addend commands
  : the operation will *not* be performed if the element will become *empty* during/through the operation in the case of replace and remove commands

4.6.3 Loops

4.6.3.1 Conditional loop

Loops allow to run a set of operations for a number of times or until a certain condition is met.

The syntax:

```
loop {(optional-condition optional-max)|lines}
or
loop {(optional-condition optional-max)|end} combined with end_loop
```

- *(optional) condition* : E.g. 'description' ~ "abc" . Must be *true* while the loop is running, when *false* the loop ends and the operations continue after the last line of the loop. Any of the *pre-conditional-arguments* (as in *4.6.2.1*) can be used to specify this condition. When no condition is specified its value is assumed *true.*
- *(optional) max* : E.g. *max=6 .* Can be used to set the number of times the loop will run if no condition is specified or when the loop doesn't end with a condition value *false* before reaching this *max* value. If *max* is not specified its default value is assumed 100. This value is 'zero' based, hence max=6 will run the loop 7 times.

- *lines* : The number of lines (following the line specifying the loop) that are contained in the loop.
- *end* : Alternative to *lines* specify the word *end* in combination with *end_loop* after the last line that belongs to the loop.

A simple example to illustrate:

```
element.modify {addstart|10}
loop {('element' > "0" max=20)|end}
element.modify {calculate|1 -}
end_loop
```

This loop will subtract 1 from element until condition 'element' > "0" is false, which happens if it reaches 0. The value of *max=20* will not be reached.

4.6.3.2 Each – loop

This loop runs for each of the instances of a multi value element.
The syntax:

```
loop {(each "target element" in 'source element' optional-max)|lines}
or
loop {(each "target element" in 'source element' optional-max)|end} combined with end_loop
```

- *target element :* E.g. "temp_2" , the element which will get the value of the actual instance of the *source element.* The element name must be enclosed by " " (to disable expansion into its actual value).
- *source element:* A multi value element enclosed by ' ' (to force expansion into its values).
- *lines, end, (optional) max,:* Same as above (*4.6.3.1*)

Example (from tvdb.com.bing.ini, to set the preferred language):

```
loop {(each "mdb_temp_1" in 'mdb_show_id' max=50)|end}
mdb_temp_4.modify {addend|'mdb_temp_1'/all/fr####} * change this line to set your prefered language!!
mdb_temp_5.modify {addend|'mdb_temp_1'/all/en####} * leave this as it is!
end_loop
```

4.6.4 The modify commands

4.6.4.1 Replace

Performs a replacement of all occurrences of the string value of *expression-1* in the element to modify with that of *expression-2.* These expressions may contain *text, elements* and *scrubstrings* components (see *4.6*, *expression_1 and expression_2 components*). Expression-1 may also contain indices if combined with a *type=char,word,sentence,paragraph* or *element* argument (see *4.6.1.4*) or regular expressions (see *4.6.1.5*) with a *type=regex* argument.

Note: *In the latter case (regex) the replacement is selective (only the matched string on the position where it matches is replaced)!!*

Examples:

- `rating.modify {replace(null)|TODOS LOS PÚBLICOS|todos}` , replaces the string TODOS LOS PÚBLICOS in element rating by the string *todos*.
- `rating.modify {replace(type=word)|2|NIÑOS}` , replaces the third word with NIÑOS, result TODOS LOS NIÑOS !! When replacing sentences (type=sentence) or paragraphs (type=paragraph) make sure to add the sentence separator (.) or the paragraph separator (\n \r or \n\r) to *expression-2.*
- Example of the selective replace with type=regex:
  Suppose the following text in a description: *The third year of the third century.*
  This contains two strings *third.* The following replace command only replaces the first one.
  `description.modify {replace(type=regex)|”The\s(third)\s year”|3rd}`
  Result : *The 3rd year of the third century.*

4.6.4.2 Remove

The command *remove* comes in three variants depending on the argument *type:*

- Without argument *type* or with *type=string,* it removes all occurrences of the string value of *expression-1* from the element to modify. As with the command *replace,* the *expression-1* may contain *text, elements* and *scrubstrings* components.
  Example:
  `title.modify {remove(notnull type=string)|: 'subtitle'}` * removes the value of element subtitle in title after the colon : , which is also removed.
  Suppose `{single|<span id=programmeheading|: |</span>|<br>}` is the scrubstring for the subtitle element, the

next gives the same result:
title.modify {remove(notnull)|: '{single|<span id=programmeheading|: |</span>|<br>}'}

- With *expression-1* containing indices, combined with argument *type=char* , *word* , *sentence* , *paragraph* or *element.* In this case the part of the element, determined by the *indices* will be removed. (see *4.6.1.4*)
- With *expression_1* containing regular expressions (see *4.6.1.5*) and *type=regex*.

4.6.4.3 Substring

The command *substring* extracts parts of an element determined by the *indices* in expression-1. The result is the opposite of *remove* when this is used with indices. Command *substring* only works with *expression-1* containing indices, the argument type with one of the values *char, word, sentence* or *element* is required (see *4.6.1.4*) or *expression-1* containing regular expressions together with *type=regex* (see*4.6.1.5*)
Examples:
rating.modify {substring(type=word)|0 1} , replaces the rating value by the value of its first word.
rating.modify {substring(type=regex)|"\A(\w+?)\s+"} , does the same but using type=regex

4.6.4.4 Addstart and Addend

These commands simply add the result of *expression-1* to the *element-to-modify.* With these commands indices in *expression-1* are not supported.

Element.modify {addstart(optional arguments)|expression_1}
Element.modify {addend(optional arguments)|expression_1}

4.6.4.5 Calculate

This command allows simple arithmetic calculations. Supported are + (add) - (subtract) * (multiply) / (divide). Furthermore four special calculations are supported; # (count, see *4.6.4.5.1*), @ (index-of, see *4.6.4.5.2*), date and time calculations (see *4.6.4.5.3*) and bitwise calculations (see *4.6.4.5.4*)
Its syntax is based on RPN (Reverse Polish Notation), which differs from the standard *a + b* and uses *a b +* . Its advantage is that it avoids complex syntax like *(a + b) * c which* cannot be expressed without () in the standard way. In RPN it is simply *a b + c **
However, because it is thought that more complex calculations will seldom be necessary only a simplified version of RPN is implemented. Like the standard *(a + b) * c / (d - e)* would be *a b + c * d e - /* in full RPN. Here we must do that in three steps: (step-1) *a b + c ** then (step-2) *d e -* then (step-3) *result-1 result-2 /*
The complete syntax of calculations:

Element.modify {calculate (optional arguments)|RPN expression}
Element.modify {calculate (optional arguments)|'element' RPN expression}

It can also be used without any RPN expression; in that case the element value is only converted to a numeric value in the format specified by the *format* argument:

Element.modify {calculate (optional arguments)}
Or with the input in another element
Element.modify {calculate (optional arguments)|'element'}

A few examples :

- temp_1.modify {calculate(format=F2)|'temp_1' 240 /}
  Divides temp_1 by 240 and assigns the result back to temp_1. If temp_1 is not a numeric string, its value will be zero. If it contains numeric string(s), its value will be the value of first numeric string in it. For example suppose temp_1 has the value *width=576px,* it will be converted into 576 and the result will be 2.40
  In this first example the element to modify temp_1 is the same as the element from which the value is used. In that case the following is the same:
  temp_1.modify {calculate(format=F2)|240 /}
- temp_1.modify {calculate(not "0" format=F2)|2 +} Adds 2 to temp_1 if temp_1 is not "0". Uses the (pre)conditional expression not "0" , see *4.6.2 Conditional arguments*
- temp_1.modify {calculate(format=chartodec)} Converts the input string in temp_1 in its decimal representation in a concatenated string of numbers. E.g. if temp_1 is 'a' the result is 097 , if 'ab' result 097098, etc.
- index_start.modify {calculate(format=time)|'previous_start' 'index_temp_1'+ 'previous_index_temp_2'+}
  Use of the previous_start and previous_index_temp_2 elements . In it, the value of the 'same' element is stored of the 'previous' show. (see *4.5.1*, *4.5.2*).
- Suppose the start time is given in any of the three supported numerical values of the *format=time* argument (see 4.6 *arguments format)* the following examples will convert the element value in the hh:mm time format.:
  index_start.modify {calculate(format=time)} which is the same as
  index_start.modify {calculate(format=time)|1 *}

4.6.4.5.1 # Count

It will return the number of occurrences that a certain string is contained in an element.
Count comes in two variants:
1*. Occurrence*. Without argument *type* (or with *type=string):*
It will return the number of occurrences that a certain string is contained in an element.
The syntax:

```
element.modify{calculate(optional-arguments)|'other-element' string #} or
element.modify{calculate(optional-arguments)|string #}
```

- *'other-element'* : the element in which the number of occurrences of "string" is counted. If 'other-element' is omitted the element to modify is evaluated for the occurrence of "string".
- *string* : if entered as string value, like "; " it must be enclosed by "" to allow spaces in it. It can also be entered as 'element' , enclosed by '', like 'title', in which case the number of occurrences of the value of that element will be counted.
- # : the operant for count.

  Example of *occurrence*:

  starrating.modify {calculate(format=F0)|"*" #}

  This will count the number of * characters in starrating and assign that value to it in the F0 format (0 decimal digits). Suppose starrating is ***, then after this operation it will become 3

2. *Length*: With argument *type* values *char, word, sentence, paragraph* or *element.* Returns the *length* of the *element.* The syntax:

```
element.modify{calculate(type=xx optional-arguments)|#} or
element.modify{calculate(type=xx optional-arguments)|'other-element' #}
```

- argument *type* : See 4.6, *arguments*.
  If *type=char*, the length returned is the number of *characters* in *element* or *other-element.* Similarly, if *type=word, sentence* or *element* length is the number of *words, sentences* and *element values* respectively.
- optional-arguments: An obvious one here is *format.* (see 4.6 *arguments*)

4.6.4.5.2 @ Index-of

This will return the starting position (index) of a certain string contained in an element. The result is 'index based' (starting with 0) The syntax:

```
element.modify{calculate(optional-arguments)|'other-element' string @} or
element.modify {calculate(optional-arguments)|string @}
```

- *'other-element'* : the element in which start location of "string" is determined. If 'other-element' is omitted the element to modify is evaluated for the location of "string".
- *string* : if entered as string value, like "; " it must be enclosed by "" to allow spaces in it. It can also be entered as 'element' , enclosed by '', like 'title', in which case the start location of the value of that element will be returned.
- *@* the operant for the index-of . If "string" occurs more than once it will return the start position of the first occurrence. If prefixed by a minus-sign, like -@ , the start position of last occurrence will be returned.
- (optional) argument *type* : See 4.6 *arguments*.
  It specifies the index-base. Possible values are: *type=string* (default, the result is returned as string position), *type=char* (the result is returned as character positions), *type=word* (the result is returned as word positions), *type=sentence* (the result is returned as sentence positions), *type=paragraph* (the result is returned as paragraph positions) and *type=element* (the result is returned as element positions)
- Note: If the "string" doesn't occur in the element, -1 will be returned. Also, note that with types *word*, *sentence* and *element,* if in a word, sentence or element, the "string" is contained, the index position of it is returned.

Example: an element with value "the quick brown fox" and a "string" with value "own" . If :
  type=char : result = 12
  type=word : result = 2 (from the word : brown)
  type=sentence : result = 0 (this sentence contains string "own")
  type=element : result = 0 (this element has only one value which contains "own")

4.6.4.5.3 Date and time calculations

- With *Date and Time calculations* it is possible to add or subtract timespan values from *date*, *time* or *timespan* values.
- It can also be used to convert to - and format *date*, *time* and *timespan* values. If the input string is, or contains a numeric value, it automatically converts it to a date, time or timespan in the specified or default format. (see also further down in this section)
- And, it can convert a time into another timezone
- It will also automatically convert the following types of numeric input values:

  - Decimal day-time values, like 16.35 will be converted into 16:21.
  - Decimal timespan values, like 16.35 will be converted into 16:8:24 days or 16:21 hours
  - Integer timespan values are considered to be in minutes:
    like 80 will be converted into 1:20 (hours:minutes)
    and 1520 will be converted into 1:1:20 (days:hours:minutes)
  - Integer date values in VBA day-number format as used in MS-Office
    E.g. 40787 will be converted to 2011/09/01
  - Integer date values in UNIX date format (Seconds counting from 01/01/1970).
    E.g. 1314866400 will be converted into 8:40 or 2011/09/01
  - Integer date values in JavaScript date format (1ms units counting from 01/01/1970).
    E.g. 1314866400000 will be converted into 8:40 or 2011/09/01
  - Integer date values in .NET ticks format (100ns units counting from 01/01/0001).
    E.g. 634504632000000000 will be converted into 8:40 or 2011/09/01

Its syntax:

element.modify {calculate(opt.args. format=time/date/timespan)|timespan +/-}
or
element.modify {calculate(opt.args. format=time/date/timespan)|'element' timespan +/-}
or
element.modify {calculate(opt.args. format=time/date/timespan)}
or
element.modify {calculate(opt.args. format=time/date/timespan)|'element'}
or with the timespan in an element:
element.modify {calculate(opt.args. format=time/date/timespan)|'element' 'element' +/-}
or to convert a time to another timezone:
element.modify {calculate(timezone=timezone_id)}

- *timespan* (optional): The timespan to add or subtract. If format=time specify *hours:minutes*, if format=date specify *days:hours:minutes* or *days:hours.* If specified without timespan and +/- operator, the result is the date - or time formatted value of the input value. (See *arguments*, *format* for details about date, time and timespan formats and date-time format strings)
  In the place of this timespan an element with the value and format of a timespan (as above) can be used if enclosed by ', like in the last syntax.
- +/- operators
- *'element'* (optional): Contains the input value. If omitted the input value is taken from the element to modify.
- *'element'* can also be one of the date/time read-only elements '*urldate*', '*now*' or '*showdate*' (see *APPENDIX C Read-only elements*). Its expanded value will be formatted conform the format argument value.
- *timezone_id* : A timezone id as used specified in *4.2.7 TimeZones*

Examples:

- Add a timespan to a time:
  element.modify {calculate(format=time)|2:15 +} gives the following results:
  If element = 16:20 —-> 18:35
  If element = 6.5 —-> 08:45 (the numeric value 6.5 is first converted to 6:30)
  If element = 23.25 —-> 01:30
- Convert a decimal to time:
  element.modify {calculate(format=time,HH:mm)}
  If element = 12.33 —-> 12:20
- Convert an integer to timespan:
  element.modify {calculate(format=timespan,days)}
  If element = 265 —-> 0:4:25
  element.modify {calculate(format=timespan,hours)}
  If element = 265 —-> 4:25
- Subtract a timespan from a date:
  element.modify {calculate(format=date,yyyy/MM/d H:mm)|1:5:30 -}
  If element = 2011/11/15 9:55 —-> 2011/11/14 4:25
  If element = 1314866400 —-> 2011/08/31 3:10
  (the numeric UNIX date value 1314866400 is first converted to 2011/09/01 8:40)

- Convert a date into Unix date format:
  element.modify {calculate(format=date,unix)}
  If element = 2011/11/15 9:55 —-> 1321350900
- Timespan days:
  element.modify {calculate(format=timespan,days)|1:17 +}
  If element = 7.23 —-> 8:22:31 (days)
  If element = 2:50 —-> 5:19:0 (days)
  If element = 1520 —-> 2:18:20 (days)
- Timespan hours:
  element.modify {calculate(format=timespan,hours)|1:17 +}
  If element = 7.23 —-> 8:30 (hours)
  If element = 2:50 —-> 4:7 (hours)
  If element = 265 —-> 5:42 (hours)
- Timespan conversion, days:
  element.modify {calculate(format=timespan,days)}
  If element = 7.23 —-> 7:5:31 (days)
  If element = 1520 —-> 1:1:20 (days)
- Timespan conversion hours:
  element.modify {calculate(format=timespan,hours)}
  If element = 7.23 —-> 7:13 (hours)
  If element = 265 —-> 4:25 (hours)
- Timespan from subtracting two times
  t1 and t2 must be (converted?) in unix, java or ticks format
  element.modify {calculate(format=timespan,hours/days)|t1 t2 -}
  If t1 = 1485591900 and t2 = 1485475200 (unix) —-> 8:25 (hours) or 1:8:25 (days)
- Convert a time into another timezone
  element.modify {calculate(timezone=Africa/Mogadishu)}
  If site{timezone=Atlantic/Canary} and element=1485475200 (unix) ----> 26/01/2017 21:00:00

A practical example:
Suppose a site allows to grab more than one day of index pages at once. For that it is required to specify the start and stop date in the url_index. Calculating and formatting this stop time can be done as follows:
Get the config timespan value and add 1 day (because config_timespan_days is 0 based) (see *4.5.3 Special elements* for more info about the special global elements and *4.5.4 Read_only elements*)

    index_variable_element.modify {calculate(format=F1)|'config_timespan_days' 1 +}

convert to the proper timespan string required to add to the start date ; urldate

    index_variable_element.modify {calculate(format=timespan,days)}

get the start date from urldate and format as required

    index_temp_1.modify {calculate(format=date,yyyy-MM-dd)|'urldate'}

calculate and format the stop date by adding the number of days from 'index_variable_element'

    index_temp_2.modify {calculate(format=date,yyyy-MM-dd)|'urldate' 'index_variable_element' +}

Another syntax that might come in handy by occasion, is when the *'element'* containing the input data is replaced by a constant. This must be done by adding '>> ' (two forwards arrows and a space) to the constant, like this:

    element.modify {calculate(format=time)|3:22>> 3:12 +}   result —-> 6:34 or
    element.modify {calculate(format=timespan,days)|16.35>> }   result —-> 16:8:24

4.6.4.5.4 Bitwise Calculations

The program supports the most common operators for this type of calculations : *and*, *or, xor, shiftl* (left shift)*, shiftr* (right shift) and *not.*
Examples:
Assume *element* has the value 187

```
element.modify {calculate(format=F0)|32 and} * result element=32
element.modify {calculate(format=F0)|4 or} * result element=191
element.modify {calculate(format=F0)|85 xor} * result element=238
element.modify {calculate(format=F0)|1 shiftl} * result element=374
element.modify {calculate(format=F0)|2 shiftr} * result element=46
element.modify {calculate(format=F0)|not} * result element=68
```

It is also allowed , as with most other operations, to specify the source and target elements separately
target_element.modify {calculate(format=F0)|'element' 4 or} * result target_element=191
All relevant optional arguments like debug and conditionals are allowed.

4.6.4.6 Cleanup

This can be useful to tidy-up the result of a scrubbed element. It:

- tries to remove remaining html tags. (see also the argument *tags=* further down this section)
- replaces newline \n and tabs \t characters by a space.
- removes carriage returns.
- replaces multiple spaces by single spaces
- removes leading and trailing spaces
- removes illegal xml characters.
- restores Unicode character sequences like \\u00e6 to the actual chars
- restores special html characters above char 127 , like &auml; to the actual char ä by default
- performs optional upper– and lower case conversions depending on the *style* argument.

Note that it is allowed to add newline \n and tabs \t to elements with the addstart, addend and replace command. If a cleanup is executed after this is done, they will be removed again. Cleanup should be executed before these operations in such cases. Its syntax:

Element.modify {cleanup(optional arguments)}

- optional argument: cleanup has its own dedicated arguments (see *4.6, arguments*):
  - style, The style argument can be added to specify the required style of the cleanup result.
    Possible values are :
    - style=sentence
    - style=name
    - style=upper (convert to UPPERCASE) and
    - style=lower (convert to lowercase)
    - style=regex (formats the input into a regular expression, inserts the required escape codes)

    style can also be used to convert the input string (see *4.6.4.6.3 Cleanup as string converter*)
    - style=urlencode and urldecode
    - style=uniencode and unidecode
    - style=jsondecode
    - style=htmlencodespecialchar and style=htmldecodespecialchar
    - style=sha256encode and style=md5encode
    - style=base64encode and style=base64decode
  - removeduplicates (see *4.6.4.6.1 Cleanup with argument removeduplicates*) and
  - tags (see *4.6.4.6.2 Cleanup with argument tags.*)

4.6.4.6.1 Cleanup with argument *removeduplicates*

To remove 'duplicate' members of multivalue elements. This is especially helpful in the result of showsplits when overlapping index_pages lists the same show double. Unfortunately, very often the listing of these 'duplicates' are not completely the same. A whole set of dedicated arguments allow to locate the exact duplicates.
By default two (or more) elements are defined as duplicate if, when compared, the resulting matching factor is higher than the *titlematchfactor* (as described in 4.3 *titlematchfactor*) specified in the SiteIni.
Its syntax:

Element.modify {cleanup(removeduplicates optional-dedicated-arguments)}

- argument removeduplicates : To further specify how the program decides if elements are considered duplicates a type can be added, like this removeduplicates=type :
  - type, optional. Specifies the algorithm and (optional) the matching factor that is used to determine duplicates. Possible values are *equal (*default), *title* and *name . Equal* doesn't use any special measures apart from the *matchingfactor. Title* uses a case insensitive comparison excluding certain abbreviations that is also used to compare *index_title* with the *xmltv title* as part of the incremental update process (see 4.5.1, index_title). *Name* uses a special comparison method to compare names. This can be useful to remove duplicates in credit elements such as *actor.* It finds duplicates like John Doe and J. Doe. Optionally, together with *type* , a different matching factor than the default, can be specified. E.g. *Removeduplicates=name,50* or *removeduplicates=equal,70* etc.
- Optional dedicated arguments :
  - link : Specifies another multivalue element from which its members must be removed on the same position as the duplicates found in the *element.* A typical example is the linked elements *index_site_id* and

index_site_channel.
E.g. : index_site_id.modify {cleanup(removeduplicates=equal,100 link="index_site_channel")}

- span : optional. Default: span=all. Specifies how far from each other in the input array the duplicates are accepted as duplicates. E.g. if span=1 only duplicates next to each other are accepted. This helps to remove duplicates from index_showsplit that resulted from index pages overlaps. Without this in some cases also shows that happen daily on the same time will be removed.
  E.g. : index_site_id.modify {cleanup(removeduplicates=equal,100 link="index_site_channel" span=1)}
- keepfirst or keeplast : Default keepfirst. Specifies which of the duplicates to keep after removal of the others.
  E.g. : index_showsplit.modify {cleanup(removeduplicates span=2 keeplast)}

4.6.4.6.2 Cleanup with argument *tags.*

Without this argument *cleanup* removes strings enclosed by < and > of less than 15 characters. A more complete, programmable removal of 'tag like' string components in the element can be achieved using the argument *tags.* Its syntax:

```
element.modify {cleanup(tags="start-string""end-string")}
```

- start-string . The string (or a single character) which defines the start of the 'tag' to be removed.
- end-string . The string (or a single character) which defines the end of the 'tag' to be removed.

The removed string includes *start* and *end* string.
Examples:
description.modify {cleanup(tags="<"">")} The simplest form, removes everything between < and >
description.modify {cleanup(tags="<a class""</a>")} Removes everything between *<a class* and *</a>*
description.modify {cleanup(tags="\"http://""\" .")} Removes everything between *"http://* and *" .*
A further option is to remove strings at the beginning or end of the element. For strings at the beginning use tags="/=string" and for at the end tags="string=/"
Example:
description.modify {cleanup(tags="/=\"")} removes a " at the beginning of the description.

4.6.4.6.3 Cleanup as string converter

Converts a string into another string encoding format.
A few examples:

- urlencode
  element.modify {set|query:find?subject=this is my subject}
  element.modify {cleanup(style=urlencode)}
  result: query%3afind%3fsubject%3dthis+is+my+subject
- and urldecode
  element.modify {set|query%3afind%3fsubject%3dthis+is+my+subject }
  element.modify {cleanup(style=urldecode)}
  result: query:find?subject=this is my subject
- uniencode
  element.modify {set|abcde}
  element.modify {cleanup(style=uniencode)}
  result: \u0061\u0062\u0063\u0064\u0065
- and unidecode
  element.modify {set|\u0061\u0062\u0063\u0064\u0065}
  element.modify {cleanup(style=unidecode)}
  result: abcde
  remark: In the later versions of the program, cleanup without argument style=unidecode, will have the same result!
- htmlencodespecialchar
  element.modify {set|<a href='test'>Test</a>}
  element.modify {cleanup(style=htmlencodespecialchar)}
  result: &lt;a&#160;href=test&gt;Test&lt;/a&gt;
- and htmldecodespecialchar
  element.modify {set|&lt;a&#160;href=test&gt;Test&lt;/a&gt;}
  element.modify {cleanup(style=htmldecodespecialchar)}
  result: <a href='test'>Test</a>
- jsondecode
  in json the escape chars:\"   \\   \/   \b   \f   \n   \r   \t

are represented by :\\\" \\\\ \\\/ \\\b \\\f \\\n \\\r \\\t
Use jsondecode to restore the escapes:
element.modify {set|\{"id": "A\\\\1","name": "A \\\"green\\\" door"\}}
element.modify {cleanup(style=jsondecode)}
result: {"id": "A\\1","name": "A \"green\" door"}

- base64encode
  element.modify {set|abcde}
  element.modify {cleanup(style=base64encode)}
  result: YWJjZGU=
- base64decode
  element.modify {set|YWJjZGU=}
  element.modify {cleanup(style=base64decode)}
  result: abcde
  With formatting 'decimal' :
  element.modify {cleanup(style=base64decode,decimal)}
  result: 97|98|99|100|101
- encryption encoders
  sha256encode
  element.modify {set|abc}
  element.modify {cleanup(style=sha256encode)}
  result: BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD
  and md5encode
  element.modify {set|abc}
  element.modify {cleanup(style=md5encode)}
  result: 900150983CD24FB0D6963F7D28E17F72

4.6.4.7 Clear

If it is required to clear the content of an element one is inclined to use remove, like this:

element-A.modify {remove|'element-A'}

This works for single value elements but not for a multi value one, which is a bit difficult to understand. The reason is that *remove*, without a type specification or with *type=string,* which is default, tries to locate the expanded string 'element-A' in each of the elements of element_A. As explained in *4.6.1.3 Multiple value elements and modify* , expanded multi value element values include the value of each of the elements it contains separated by the standard internal element separator | . It is obvious that it fails to locate such an expanded value in each of the elements of element_A.
To clear the content of multi value elements (and also single value elements!) one can use :
element-A.modify {remove(type=element)|'element-A' 0} or in short
element-A.modify {remove(type=element)|0}
This removes the elements of element-A , regardless their content, starting from the first (index 0) to the last (because that's default if no length is specified) (see also *4.6.1.4 Expression-1 with indices*).
Because the complication to understand the scrubstring element-A.modify {remove(type=element)|0} it is also possible to simply use :

element-A.modify {clear}

The program automatically substitutes the command *clear* by *remove(type=element)|0*

4.6.4.8 Select

Pre-conditional arguments (see *4.6.2.1*) offer a way to select an element with the operators = ~ But that may be simple for a single element but to select certain members from a multi value element involves to step through all the members in a loop and examining each member individually. This is not only time consuming but also requires a lot of lines in the SiteIni.
The command *select* makes this a lot easier and faster. It can be applied directly to multi value elements. Its syntax:

element.modify {select(optional args)|string operator}

- string: The string for which the members of element are to be selected.
  Must be enclosed by " ". May contain elements to expand.
  Examples:
  "This string"

"This 'index_temp_1'" (index_temp_1 will be expanded to its value)
"'temp_1'" (also permitted is just 'temp_1' in this case)

- operator: The comparison operator. Must be one of the following:
  - = string and member of element must fully match, case insensitive
  - == string and member of element must fully match, case sensitive
  - ~ member of element must contain string, case insensitive
  - ~~ member of element must contain string, case sensitive
  - /= member of element must start with string, case insensitive
  - /== member of element must start with string, case sensitive
  - =/ member of element must end with string, case insensitive
  - ==/ member of element must end with string, case sensitive
  - ?title string and member of element must match following the title-match algorithm and using the titlematchfactor as deciding factor (see also *4.6.4.6.1 type* for more details about this algorithm)
  - ?name string same as ?title using the name-match algorithm (also described in *4.6.4.6.1*)

Note!! Add not in front of any of the operators and the result will be reversed.
E.g. not~~ will select all members that do not contain the string.

WARNING! Unlike other commands the following is not supported:
element.modify {select(optional args)|'other-element' string operator}

4.6.4.9 Sort

An easy and fast way to sort the members of multi value elements by its value or a part of it. This command must always be accompanied by a scrub string and/or an operation to define the value of the special element *sort_by* (see APPENDIX E). Its syntax:

```
Element-to-sort.modify {sort(direction,for)}
```

and

```
sort_by.scrub {scrubstring with argument (target="element-to-sort")}
```

and/or

```
sort_by.modify {operation with argument (target="element-to-sort")}
```

- *direction* : Must be ascending or descending
- *for* : Must be string , integer or datetime
  The sorting will be carried out using the value of sort_by , if properly linked to the element to sort with the argument target= , using either a string-order if *for* is a string or an integer-number-order if *for* is an integer or a datetime-order if for is a date/time. In case of an integer or a datetime, the program will attempt to convert the value of sort_by to an integer or a date/time. If that fails, the sorting will fail with a warning.

Example:

```
index_showsplit.modify {sort(ascending,string)}
sort_by.scrub
{single(target="index_showsplit")|<activation_datetime>||</activation_datetime>|</activation_datetime>}
sort_by.modify {calculate(target="index_showsplit" format=date,unix)}
```

4.6.4.10 Set

A simple way to set an element to a value. It will replace any existing value.
syntax:

```
element.modify {set(optional arguments)|value}
```

- value : The new value of element. May contain an 'element' to expand to its value.

Examples:

```
temp_1.modify {set|0}
temp_1.modify {set('description' not "")|'description'}
```

4.6.4.10.1 Set with argument type=run

A way to run an external program which output is passed to 'element'
Any type of executable program, like .exe .bat .cmd etc can be used.
syntax:

```
element.modify {set(optional arguments type=run)|path of the external program|commandline argument(s)}
```

- type=run Activates this option
- path of the external program Specify the path and name of the program to run.
- Commandline arguments of the program to run. More than one argument must be separated with a space.

Element will get the value of the output of the program to run that is written to the console. Therefor the program to run needs to be programmed such that it only writes the result to the console and nothing else! (because everything written to the console will be passed to 'element')

A small example to illustrate the procedure:
Suppose the following batch file b.bat in the same folder as the config file:

```
@echo off
set a="%1%"
set b="%2%"
set/a c=%a%+%b%
echo %c%
```

(It takes two arguments, %1% and %2% into a and b, adds them together in c, and writes the c to the console.)
And suppose this in the SiteIni:
temp_1.modify {set(type=run)|b.bat|5 7}
Result: temp_1 value 12
Or like this:
temp_1.modify {set(type=run)|b.bat|'temp_2' 7}

4.6.5 Operations (and scrubs) with argument 'pattern'

Argument pattern is different from the other arguments mentioned in *(4.6 Operations)* and *(4.2.5 Arguments)* . They are the only arguments that can be used both in scrubs and in modify operations . On the other hand they are limited to certain elements , the time elements start and stop for a (date-time) pattern and the episode element for a (episode-number) pattern. In fact, they perform an operation, even when used in a scrub. It tries to match the input data with the value of the pattern and if the match is succesfull it converts the data to a certain format.
Note that this 'operation' is done on the data that is the result of the scrub or the modify operation, using all the specified arguments *except!!* the pattern argument! The operation for pattern argument is performed as last.
As mentioned above, the pattern argument comes in two forms :

4.6.5.1 Date-time patterns

See also *(4.5.2.1 Time elements)* and *(pattern argument)*
These only work for the elements start and stop, combined with any of the possible prefixes.
The value of 'pattern' must be enclosed by " " and follows the notations of the custom date and time format strings *https://msdn.microsoft.com/en-us/library/8kb3ddd4(v=vs.110).aspx* or the standard date and time format strings *https://msdn.microsoft.com/en-us/library/az4se3k1(v=vs.110).aspx*
The converted date-time output value follows the format that is custom for the cultureinfo value as in *(4.3 General Site dependent data)*
Examples:
- Suppose a very simple index show value of :
  showtitle,20170109T06:00#3:50 (in which the undelined part is the datetime string)
  and cultureinfo=en-US
- Get the start time with a scrub and a pattern:
  index_start.scrub {regex(pattern="yyyyMMddTHH:mm")||,(.+?)#||}
  result :1/9/2017 6:00 AM
- Get the start time with a modify command:
  try another cultureinfo=nl-NL
  index_temp_1.scrub {regex||,(.+?)#||} * returns 20170109T06:00
  index_start.modify {set(pattern="yyyyMMddTHH:mm")|'index_temp_1'}
  result : 9-1-2017 06:00

This last example illustrates that the 'pattern' operation is done at the end. Command *set* is performed first and results in a value of 20170109T06:00 , next the *pattern* operation recognises the data as a match and converts it.
One would be inclined to use *(4.6.4.5.3 Date and time calculations)* together with pattern but that is tricky because the calculation command does its own converting and formatting, so it will only work if the calculate result is such that it can match the pattern value in which case it is probably not necessary to use a pattern anymore.

4.6.5.2 Episode-number patterns

See also *pattern* and *episode_element*
The pattern argument used in scrubs or modify operation of the episode element (any possible prefix) enables to recognise and convert custom episode number values into one of two possible standards: xmltv_ns (see *xmltv.dtd*) or a simple 'onscreen' format. The use of an episode number standard is required for most PVR xmltv import utilities.

The favorite episode number standard for this purpose is *xmltv_ns*. This is also important if the MDB postprocessor is used to extract series data with episode-number as identification of the series. (see *6.4 Series episode details*)
With the help of the 'pattern' argument most custom episode number formats can be converted into one of these standards. Which of the two is determined by the episodesystem value as in *(4.3 General Site dependent data)*

- The 'onscreen' format used by the program as output of the conversion:
  [Sn[/St]]En[Et][Pn[/Pt]]
  S season, Sn season number , St total number of seasons
  E episode, En episode number, Et total number of episodes
  P part, Pn part number, Pt total number of parts
  [] is optional
  Spaces between S E and P allowed
- Examples :
  S2/4 E12/20 P1/2 (season 2 of 4 episode 12 of 20 part 1 of 2)
  or without the 'totals' :
  S2 E12 P1
  just the episode:
  E12 or with total E12/24
  season and episode, no space:
  S2E12
- The pattern specification
  the pattern value must be composed of the following elements:
  strings , like *Season* or *Temporada* or *Aflevering* or *Folge* , whatever strings or characters accompany the numbers.
  - One or more of the following standard expressions to indicate the meaning of the numbers (including the ' '):
  'S0' or 'S1' the season number, 'S0' for zero based numbers (first season is season 0) 'S1' for one based numbers
  'St0' or 'St1' the total number of seasons (if present)
  'E0' or 'E1' and eventual 'Et0' or 'Et1' for the episode number
  'P0' or 'P1' and eventual 'Pt0' or 'Pt1' for the part number
  * spaces in patterns are ignored
  * patterns are case insensitive
  * multiple patterns can be specified if a site uses different mixed formats. The program automatically selects the one that gives the best matching. Each pattern must be enclosed by " ".

Examples :

Input value:	Season 5, Episode 12	pattern : "Season'S1',Epsode'E1'" (for one based numbers)
or :	Ep12/24 of Sn3	pattern : "Ep'E0'/'Et1'ofSn'S0'" (partly zero based)
or :	Season 3 of 4, Episode 3 of 12	pattern : "Season'S1'of'St1',Episode'E1'of'Et1'"

Lets try the last one:
- Suppose a very simple index show value of :
  showtitle,20170109T06:00#<u>3:50&Season 3 of 4, Episode 3 of 12</u> (the underlined part is the episode number)
  and episodesystem=onscreen
- Get the episode element with a scrub and a pattern:
  index_episode.scrub {regex(pattern="Season'S1'of'St1',Episode'E1'of'Et1'")||&(.+)\z||}
  result : S3/4 E3/12
- Get the episode element with a modify command:
  try the another episodesystem=xmltv_ns
  index_temp_1.scrub {regex||&(.+)\z||} * returns: Season 3 of 4, Episode 3 of 12
  index_episode.modify {set(pattern="Season'S1'of'St1',Episode'E1'of'Et1'")|'index_temp_1'}
  result : 2/4.2/12.

Often in one site or even channel, parts of the episode number are missing or different from others. In such cases a second (or third etc) pattern can be added.
Example :
Suppose the next index show of the same channel as the one above is like:
  showtitle,20170109T09:50#<u>2:00&Episode 4, Part 2</u>
  for this we can specify a second pattern:
  index_episode.scrub {regex(pattern="Season'S1'of'St1',Episode'E1'of'Et1'"<u>"Episode'E1',Part'p1'"</u>)||&(.+)\z||}
  result (xmltv_ns) : .3.1
  result (onscreen) : E4 P2

The result for the first show remain as above. The program uses the pattern that matches the input value best!

4.6.6 Examples of operations

- description.modify {addstart("")|no details} adds - no details - to an empty description
- description.modify {addstart(null)|no details} same as above, adds - no details - to an empty description
- subtitle.modify {addstart(not "")|Episode: } adds -- Episode: -- before the subtitle, but only if subtitle wasn't empty before the action.
- rating.modify {replace("")|nine|9} replaces the word 'nine' by the number 9, also if the word 'nine' is the whole rating
- description.modify {replace(not "")|Afl.: 'subtitle'.| This episode:} replaces the subtitle listing like - Afl.:Heads Up. - in the description by -- This episode: -- but only if the action doesn't replace the whole description.
- ratingicon.modify {addstart("")|'rating'.png} adds a ratingicon if the site doesn't list one.
- As an example of the use of a scrubstring in expression-1, consider the following html:
  <p class="verhaal">Amerikaanse (USA) Drama uit 1995 van Taylor Hackford. Met: Kathy Bates, Jennifer Jason Leigh, Judy Parfitt, Christopher Plummer e.a. Huishoudster Dolores wordt beschuldigd van de moord op haar vervelende en veeleisende werkgeefster, bij wie ze al jarenlang in dienst was. Door deze gebeurtenis wordt ook de dood van haar man, twintig jaar geleden, weer opgerakeld en rechercheur John Macky is vastbesloten Dolores dit keer wel voor moord achter slot en grendel te krijgen. Haar dochter Selena, een succesvolle journaliste in New York, keert voor de zaak terug naar haar geboorteplaats in het kille Maine, waar ze gelijk weer met haar eigen jeugd geconfronteerd wordt.</p>
  <table style="width: 60%;">
  <tr><th width="65">Genre</th><td>Film</td></tr><tr><th>Acteur</th><td>Kathy Bates, Jennifer Jason Leigh, Judy Parfitt, Christopher Plummer</td></tr>
  <tr><th>Regisseur</th><td>Taylor Hackford</td></tr>
  </table>
  This site lists the actors double, in the description after - Met: - and later following <th>Acteur .
  To scrub the description and the actor we use:
  ```
  description.scrub {single|<p class="verhaal">||</p>|<table}
  actor.scrub {single(separator=","|<th>Acteur</th><td>||</td></tr>}
  ```
  That leaves use with a description that contains all the actors which isn't perfect. However they can be removed with the following:
  ```
  description.modify{remove(null)|Met: '{single|<p class="verhaal">|Met:|e.a.|<table}' e.a.}
  ```
  We cannot use the element actor in expression-1 as in
  ```
  description.modify{remove(null)|Met: 'actor' e.a.}
  ```
  because actor is not a single value xmltv element anymore due to the use of the argument — separator="," - in the actor scrubstring.
  We allow - null - to be sure that the actors listing is removed even if it is the whole description. We add the following to have at least something in the description:
  ```
  description.modify {addstart(null)|No details}
  ```
- An example with calculate and numeric conditional arguments:
  Suppose the subtitle of a show is the first sentence in the description on the html detail page. So we use something like :
  ```
  subtitle.scrub {single(separator=". " include=first)|. . . . . . }
  ```
  However not all shows have a subtitle, consequently the result can also contain just the first sentence of the description. To distinguish between subtitle and a normal sentence, count the words .. max 3 words is considered a subtitle (or at least most probable)
  ```
  temp_1.modify {calculate(not "")|'subtitle' " " #} * count the spaces
  subtitle.modify {remove('temp_1' > "2")|'subtitle'} * clear subtitle if more than 3 words
  description.modify {remove('temp_1' < "3")|'subtitle'}  * remove the subtitle from the description.
  ```

5. Special Procedures and Tricks

5.1 Special procedures

5.1.1 How to configure a SiteIni file for a site using the POST Http protocol

As a preparation for the development of a SiteIni using the POST method of a HttpWebRequest it is necessary to determine the required header content. There are numerous ways to do that, a simple one is to use the development tools of your internet browser. All respectable web browsers have such tools to inspect the traffic. For IE it can be found under Tools, select Developer Tools (F12), Network (Ctrl+4), Enable Traffic (F5). All traffic will be captured after this and can be inspected. It is essential to familiarize with this tool. With it the required request headers including the postdata content (in this tool called "Request Body") can be determined and must be copied to the header specifications as described in *4.4.1.1 HTTP Headers, method GET, POST, POST-BACK and SOAP*

5.1.2 How to configure a SiteIni file for a site using the POST_BACK Http protocol

First : read *4.4.1.1*. Different from a 'regular' POST HttpWebRequest, the postdata which specifies the requested content to the server (site) is partly send to the client (in this case WG++) in response on a first request done using the GET method. Any following requests (e.g. for following days) use the POST method with postdata derived from the data received in response on the first GET request. WG++ follows a build in procedure when a POST_BACK request is started:

- Setup of a GET HttpWebRequest using the url_index as specified in the SiteIni.
- If a valid response is received, a 'special' build-in scrub procedure (see below) is started to extract the required data to compose the required Postdata.
- The program then issues the next HttpWebRequest using method POST with this postdata and the same url_index as in step 1. (So, in fact it grabs this first page twice, once to extract the postdata and a second time as part of the regular grabbing).
- Any following request is done using the POST method as described above.
- Prior to designing the SiteIni entries, determine the necessary headers and postdata in the way described in *5.1.1*. Match the postdata content with data found on the html page in response of the GET request. Tip: Locate the data that contains the string VIEWSTATE. Also Locate the possible variables and their format like channel , urldate and subpage. These variables must be added to the postdata header specification and not to the elements specified in the 'special scrub procedure'. (This procedure is only executed once, immediately after the first GET response, so will not be evaluated after the following POST responses if specified there)
- The 'special' scrub procedure as mentioned in 2. must be specified in the SiteIni using these rules:
- Use scope=urlindex e.g. within scope.range {(urlindex)|end} and end_scope
- Only the elements any of the global and the 9 available index_temp_1 to 9 are allowed.
- At the end of the procedure, the extracted data required for the following POST request must be placed in the element index_variable_element.
- two header specifications are required as a minimum:
  url_index.headers {method=POST_BACK} to initiate this method and activate the special scrub
  url_index.headers {postdata= —- the required postdata —- }

An example:

- url_index with the channel variable as discovered in the variable part of the postdata:
  url_index{url()|http://etfarag.com/Programs/ChannelDisplay.aspx?ChannelID=|channel}
- the required urldate format:
  urldate.format {datestring|dd"%2F"MM"%2F"yyyy}
- the headers:
  url_index.headers {method=POST_BACK}
  *url_index.headers {accept=application/x-ms-application, image/jpeg, application/xaml+xml, image/gif, image/pjpeg, application/x-ms-xbap, */*}*
- the postdata containing the index_variable_element containing the data scrubbed from the GET response, added to it a variable part containing urldate:
  url_index.headers {postdata=__EVENTTARGET=ctl00%24MainContent%24ChannelDisplay2%24txtDate%24txtDate&__EVENTARGUMENT=&__LASTFOCUS=&__VIEWSTATE='index_variable_element'&ctl00%24Login1%24txtUserName=&ctl00%24Login1%24txtPassword=&ctl00%24Timezone1%24drpTimezone=60&ctl00%24Search1%24txtSearch=&ctl00%24MainContent%24ChannelDisplay2%24drpChannls='channel'&ctl00%24MainContent%24ChannelDisplay2%24txtDate%24txtDate='urldate'&ctl00%24MainContent%24ChannelDisplay2%24txtDateDown%24txtDate='urldate'&ctl00%24Register1%24txtName=&ctl00%24Register1%24txtPass=&ctl00%24Register1%24txtRePass=&ctl00%24Register1%24txtEmail=&ctl00%24Register1%24dtCtlBirthDate%24txtDate=&ctl00%24Register1%24ddlTimezone=-1&ctl00%24Register1%24txtMobile=&ctl00%24Register1%24ddlCountries=-1&ctl00%24Register1%24ddlLanguages=-1}
- The special scrub procedure to extract the post_back postdata:
  scope.range {(urlindex)|end}

scrub the VIEWSTATE content:
index_variable_element.scrub {single|id="__VIEWSTATE"|value="|" />|" />}
In most cases the following control chars must be entered by their html char code in the VIEWSTATE value (this could be site specific):
index_variable_element.modify {replace|+|%2B}
index_variable_element.modify {replace|/|%2F}
index_variable_element.modify {replace|$|%24}
index_variable_element.modify {replace|\||%7C}
end_scope

5.1.3 How to configure a SiteIni file for a site using the SOAP http protocol

- First : read *4.4.1.1* Although, like the POST and the POST_BACK method, SOAP also uses request data to be send to the server. This method uses a XML file 'SOAPENVELOPE' which is automatically generated by the program after being filled with the content of the postdata header from the SiteIni.
- Unfortunately, the content of this data cannot be figured out with the tools used for the POST and the POST_BACK method (see *5.1.1* and *5.1.2*). Instead a dedicated 'sourceforge' program 'soapui' can be used to configure a SOAP request. (*https://www.soapui.org/downloads/soapui/source-forge.html*) An example (Dutch!)of the use of this program is can be read in Example-Use-Of-SoapUi.pdf available @ *http://www.webgrabplus.com/sites/default/files/download/documentation/Set%20of%20help%20files/help-files.zip*
  With the data collected that way, the SiteIni can be constructed. As example the following lines from schedulesdirect.org.ini:

url_index {url|http://webservices.schedulesdirect.tmsdatadirect.com/schedulesdirect/tvlistings/xtvdService}
url_index.headers {methode=SOAP}
* The headers, notice the username and password as required by this site
url_index.headers {customheader=SOAPAction=urn:TMSWebServices:xtvdWebService#download}
url_index.headers {customheader=Accept-Encoding=gzip,deflate}
url_index.headers {credentials=ENTER_USERNAME,ENTER_PASSWORD}
url_index.headers {accept=text/xml|contenttype=text/xml;charset="utf-8"}
url_index.headers {postdata='index_variable_element'}
scope.range {(urlindex)|end}
*
* timespan calculation to enable to add the requested timespan from the config
* add 1 day because config_timespan_days is 0 based:
index_variable_element.modify {calculate(format=F1)|'config_timespan_days' 1 +}
index_variable_element.modify {calculate(format=timespan,hours)} * convert to the proper timespan string required for the date calculation in index_temp_2
****
index_temp_1.modify {calculate(format=date,yyyy-MM-dd)|'urldate'}
index_temp_2.modify {calculate(format=date,yyyy-MM-dd)|'urldate' 'index_variable_element' +}
*index_variable_element.modify {clear} * clear the timespan value*
index_variable_element.modify {addstart()|<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><SOAP-ENV:Body><m:download xmlns:m="urn:TMSWebServices" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><startTime xsi:type="xsd:dateTime">'index_temp_1'T00:00:00Z</startTime><endTime xsi:type="xsd:dateTime">'index_temp_2'T00:00:00Z</endTime></m:download></SOAP-ENV:Body></SOAP-ENV:Envelope>}
end_scope

5.2 Tricks

- Force a show update like this: title.modify{addend(~ "NOS WK Voetbal")|(!)}
  By addition of (!) (or also (?)) to the title WebGrab+Plus will update the show despite the update decision outcome. This can be useful for shows that could have last minute changes that are not apparent from the index show times and index title. To get this update scheduled for already existing shows in the xmltv file a one-time full update run is necessary (set the update attribute to - f - in the channel entry of the WebGrab++.config.xml file for one run, like: <channel update="f" site="tvgids.nl" site_id="1" xmltv_id="NED1-tvgids">NED1</channel>)
- Avoid unnecessary show update:
  If the index_title is different from the title on the show detail page and a lower setting of the title match factor is unacceptable (too low for reliable title comparison) try the following:
  title.modify{replace(null)|Sterren 24|'index_title'}
  The show in this example has - *Sterren.nl Extra* - as index_title and - *Sterren 24* - as show detail title, which is too much difference even for a title match factor of 50. With this we simply replace the title with the index_title for a show with title - *Sterren 24* - only.
- Full rating to Short rating
  Sites normally list ratings like KIJKWIJZER ratings in a sentence like - *Afgeraden voor kinderen jonger dan 9 jaar* - of - *Let op met kinderen tot 9 jaar* - of - *drugs- en/of alcoholmisbruik* -
  The ratingicon is normally listed as a link to a picture file like - *http://u.omroep.nl/gids/pics/icons/kijkwijzer/negen.png- or - http://u.omroep.nl/gids/pics/icons/kijkwijzer/groftaalgebruik.png-*
  With the help of the modify operations they can be simplified easily (e.g. for site tvgids.nl):
  rating.modify {replace(null)|Afgeraden voor kinderen jonger dan 6 jaar|6+}
  rating.modify {replace(null)|Let op met kinderen tot 9 jaar|9+}
  rating.modify {replace(null)|Afgeraden voor kinderen jonger dan 12 jaar|12+}
  rating.modify {replace(null)|Niet voor personen tot 16 jaar|16+}
  rating.modify {replace(null)|Grof taalgebruik|Grof}
  rating.modify {replace(null)|Drugs- en/of alcoholmisbruik|Drugs}
  ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/zes.png|6.png}
  ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/negen.png|9.png}
  ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/twaalf.png|12.png}ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/16.png|16.png}ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/seks.png|seks.png}ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/eng.png|angst.png}ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/geweld.png|geweld.png}
  ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/groftaalgebruik.png|grof.png}
  ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/discriminatie.png|discriminatie.png}
  ratingicon.modify {replace(null)|http://u.omroep.nl/gids/pics/icons/kijkwijzer/drugs.png|drugs.png}
  A set of ratingicon files for the Dutch 'KIJKWIJZER' ratingsystem with the filenames as used in this example is included in the WebGrab+Plus distribution.
- Some *exclude* and *include* tricks:
  - Exclude elements with more than one word (sentence):
    element.scrub {single (exclude=" ")|scrubstring}
    This is very helpful to filter out (in general) one-word elements like category. A lot of sites use unsystematic html structures. Especially secondary elements like category can be mixed with other elements like title or subtitle behind the same separatorstrings bs es ee and bs values. This filter can help to separate them.
  - Include a list of standard values:
    category.scrub {single (include="film""serie""documentary""sports")|scrubstring}
    Another way to filter a category out of a mixed html scrubbed element. The list of strings to include should contain all the possible categories which occur in the html scrub.
  - A combination of *include* and *exclude* to further refine the result.
    Consider the following scrubstring:
    category.scrub {single (include="film""serie""documentary""sports" exclude=" ")|scrubstring}
    The include list allows to yield an element with the value - memories of a seriekiller - , which probably is a title and not a category. The exclude=" " removes this element value.

6. MDBIni file

6.1 Introduction

- The MDB postprocessor, as shortly mentioned in *3.2*, allows to add additional movies or series data grabbed from online movie and serie data-bases to the xmltv file created by the frontend WebGrab+Plus grabber. To achieve this it performs the following steps:
- Select 'candidate' shows from the xmltv input file.
- Match the selected show 'candidates' with shows in the online MDB in two steps:
  - A *'primary search'* with a general search site like BING, ASK, GOOGLE or directly in the MDB site if that supports search. This results are a number of possible *show-id*s for the verification step.
  - For series an addititinal step is required: Locating the matching *'episode_id'* for the *'show_id'* found with 'primary search'. This is done by iterating through a list of episodes and selecting the one that has the best match of the 'episode title' (= xmltv sub-title) and/or the 'episode number' (= xmltv episode-num)
  - Verify the results of the primary search in a MDB-site, like IMDb, until a 'match' is found.
    This is done by iterating through all 'candidates' and selecting the one that has the best matching of the 'matchmovie' or 'matchserie' values (like 'title', subtitle, actor etc) in the configfile 'mdb.config.xml'.
- Grab the MDB data of the matched show or series episode from the MDB site
- Merge the grabbed MDB data with the epg data from existing xmltv file.
- Due to the flexible programmable approach, much like the frontend grabber, it is possible, in principle, to grab this additional data from any of the existing online movie and serie data-bases 'MDB-sites' like IMDb.com, Allocine.com, TheTVDb.com etc. This is achieved with a MDBIni file, similar to the SiteIni file of the frontend grabber, in which all the MDB-site dependent settings and operations are specified.
- The MDB postprocessor, like the frontend grabber, uses its own configuration file, *MDB.config.xml* in which the user can specify the location of relevant files, the selection and matching parameters and what, where and how to merge the grabbed MDB data with the existing xmltv file.
  (See *3.2* and *http://www.webgrabplus.com/documentation/configuration-mdb/mdbconfigxml*)
- Besides this config file, it uses one or more *MDBIni* files, optimized for different MDB_sites or for using a different *primary search-sites* or optimized for movie or series data grabbing. These MDBIni files use the same scrubstring and operations syntax as the SiteIni files with a few small exceptions and additions. Thus, all the commands and syntax specifications described in chapter 4 are also valid for these MDBIni files besides these small exceptions and additions. (see *6.3 Differences between MDBIni and SiteIni syntax.*) It is obvious that this postprocessor has its own set of supported elements, different from the ones of the SiteIni files. (see *6.2*)

6.2 MDB Elements

Table.3 lists all the supported MDB element names and the General MDB-site dependent settings.

6.2.1 Variables in URL element values

In *url_primarysearch* :

- *'title'*	: expands to the title of the show in the xmltv source file
- *'credit'*	: expands to the first director or if not available the first actor of the show in the xmltv file
- *'productiondate'*	: expands to the prouctiondate (element <date>) of the show in the xmltv file

In *url_mdb_pn* :

- *'mdb_show_id'*	: expands to the show_id found with the primary search
- *'mdb_episode_id'*	: (series only) expands to the episode_id found with in the first MDB page

Examples of URL specifications in a MDBIni:

```
url_primarysearch {url(urlencode=1,2,3,4)|http://www.ask.com/web?&q=|imdb+|'title'|+|'credit'|&/NCR}
url_mdb_p1 {url|primary|http://www.imdb.com/title/tt|'mdb_show_id'|/}
url_mdb_p2.modify {addstart|http://www.imdb.com/title/tt'mdb_episode_id'} * the episode detail page
```

Table.3

Element Name	**source page**	**type:**		**action:**				**Description**
		movie	**serie**	**.url**	**.headers**	**.scub**	**.modify**	
url_primarysearch		✓	✓	✓	✓		✓	The url of the primary search
url_mdb_p1 *upto* url_mdb_p8		✓	✓	✓	✓		✓	The url of the mdb-site data pages
mdb_show_id	primary	✓	✓			✓	✓	The show_id that the MDB-site uses as show reference
mdb_episodetitlelist	p1 - p8		✓			✓	✓	The list of episode titles for a given show_id
mdb_episodenumlist	p1 - p8		✓			✓	✓	The list of episode numbers for a given show_id
mdb_episode_id	p1 - p8		✓			✓	✓	The episode_id of a series episode that the MDB-site uses as reference
mdb_episode	p1 - p8	✓	✓			✓	✓	The episode number, like seaon, episode, part
mdb_title	p1 - p8	✓	✓			✓	✓	The show title as present in the MDB-site
mdb_subtitle	p1 - p8	✓	✓			✓	✓	In case of a series episode, the episode title or just a show subtitle
mdb_description	p1 - p8	✓	✓			✓	✓	The show description as present in the MDB-site
mdb_starrating	p1 - p8	✓	✓			✓	✓	The MDB-site starrating
mdb_starratingvotes	p1 - p8	✓	✓			✓	✓	The number of viewer votes that contributed to the mdb_starrating
mdb_plot	p1 - p8	✓	✓			✓	✓	A short description of the show
mdb_commentsummery	p1 - p8	✓	✓			✓	✓	A selection of viewers 'ono line' comment summaries
mdb_review	p1 - p8	✓	✓			✓	✓	A selection of viewers reviews
mdb_actor	p1 - p8	✓	✓			✓	✓	A selection of actors present in the MDB-site of the show or serie
mdb_director	p1 - p8	✓	✓			✓	✓	A selection of directors present in the MDB-site of the show or serie
mdb_showicon	p1 - p8	✓	✓			✓	✓	The showicon as available in the MDB-site
mdb_category	p1 - p8	✓	✓			✓	✓	The categories as available in the MDB-site
mdb_productiondate	p1 - p8	✓	✓			✓	✓	The year of production of the show as given by the MDB-site
mdb_temp_1 *upto* mdb_temp_9	any	✓	✓			✓	✓	Usefull temporary elements in operations
mdb_variable_element	any	✓	✓			✓	✓	The MDB variant of index_variable_element
Read-Only elements, can be used in operations and in url strings and will be expanded if enclosed by ' ' characters:								
x_title								expands to the title of the xmltv input file
x_subtitle								expands to the sub-title of the xmltv input file
x_productiondate								expands to the productiondate of the xmltv input file
x_actor								expands to the first actor of the xmltv input file
x_director								expands to the first director of the xmltv input file

General MDB-site dependent settings :

	optional:		
url		The url of the 'home' page of the MDB-site	e.g. IMDb.com
mdbinitype		Specifies the purpose of the mdbini	movie or serie
culturenfo		The culture-info for the MDB-site	e.g. en-UK
charset *		The charset in which the MDB-site page are coded	e.g. utf-8
matchfactor		A number that sets the accuracy of the string matching	e.g. 80
episodesystem	✓	A string that describes the value of the episode	xmltv_ns or onscreen
searchsite	✓	The primary search site used	e.g. bing

* charset: If the charset in use for the primarysearch site is different from the one for the MDB site, specify both, separated by a comma, primarysearch first.

6.3 Differences between MDBIni and SiteIni syntax.

6.3.1 Element prefix

In SiteIni's elements the prefix , like *index_ detail_* and *subdetail_* is used to specify the (html) source page from which the element content is to be scrubbed. Not so in a MDBIni, in these, all elements that get their data from any grabbed html MDB-site page have a standard *mdb_* prefix. The html source page from which the content is to be scrubbed is specified before the separatorstrings sequence |bs|es|ee|be} (see *4.2.1.1 The 'separator strings' method*) or before the regex specification (see *4.2.1.2 The 'regular expression' method*) as follows:

```
Element.scrub {type(optional arguments)|page|bs|es|ee|be}
Element.scrub {regex(optional arguments)|page||regular expression||}
```

- *page ,* the (html) source page from which the data content is to be scrubbed.
  Possible values are :
  - *primary ,* the primary search page.
  - *p1 .. p8 ,* any one of the 8 possible MDB-Site grabbed pages
- *bs, es, ee, be* and *arguments* : Same as in a SiteIni , see *4.2.1.1 The 'separator strings' method*
- *regular expression:* Same as in a SiteIni, see *4.2.1.2 The 'regular expression' method*

6.3.2 Argument urlencode

The argument *urlencode=pos1,pos2,..,posn* allows to encode the resulting url string, it converts characters that are not allowed in a URL into character-entity equivalents. For example each space character will be converted to a plus character (+) and a plus character to %2b and more. This is required here because the variables *'title'* and *'credit'* often contain space and possibly other characters that are not allowed in URL's. One can split the url specification into pieces, separated by the well-known | character and use the *pos* values (0 based) to point to which of the pieces should be urlencoded.
Example:

url_primarysearch {url(urlencode=1,2,3,4)|http://www.ask.com/web?&q=|imdb+|'title'|+|'credit'|&/NCR}

pos=1	*imdb+*	urlencoded to *imdb%2b*
pos=2	*'title'*	e.g. expanded to *Take this waltz* urlencoded to *Take+this+waltz*
pos=3	*+*	urlencoded to *%2b*
pos=4	*'credit'*	e.g. expanded to *Tim Story* urlencoded to *Tim+Story*

* Note : in type *url* scrubstrings the variables (here *'title'* and *'credit'*) must be placed between || (see 4.4.2) and therefore have their own pos (2 and 4) and cannot be combined with the others.
Result:

http://www.ask.com/web?&q=imdb%2bTake+this+waltz%2bTim+Story&/NCR

6.3.3 MDBini scopes

Similar to SiteIni scopes as discussed in *(4.6.1.1 and 4.6.1.2)* control the effect of operations for subsequent actions of the MDB postprocessor.
The MDB processing is done in three major steps and equaly named scopes:

- *scope = primarysearch* (program sections GetPrimaryUrl, GetShowID and GetMdbUrls)
  Select a list of show 'candidates' matching the primarysearch selection criteria 'selectmovie' or 'selectserie' in the config file mdb.config.xml' using the primarysearch site.
- *scope = match* (program sections GetTitles, GetMustHaveCompare)
  Find the matching MDB show or MDB series episode using the matchmovie or matchserie values in MDB.config.xml
- *scope = getelements* (program sections GetEpisodeID, GetOptionalCompare, GetElements)
  Get all the MDB element values as requested by the MdbIni.

Similar to the use of scopes in SiteIni files, a MdbIni must be split into sections, each with its own scope. See *(6.4 Series episode details)* as example.

6.4 Series episode details

(it is recommended to also read *http://www.webgrabplus.com/content/mdb-postprocess-v20-beta-15625* for better understanding)
If the source xmltv file lists series shows of which the subtitle is the episode title and/or with a unique episode number, it is possible to obtain more episode details from a MDB-site that contains series data. (like *IMDb.com* and *thetvdb.com*).
It must be clear that series episode details cannot be located in the MDB site by using the series title alone, a series normally has a number of episodes, some over a period of several years. To locate the specific episode at least the episode-title or the episode-number must match between xmltv input and the MDB series data. <u>The program tries to match the episode-title first and if that fails or is impossible because the episode-title is not present in the xmltv, it will try the episode number.</u>
The procedure to accomplish this is a little more complicated than that for a movie , and requires a dedicated *MdbIni for series*. The objective is to get the *mdb_episode_id* that identifies the series episode in the MDB site. Only with that, the episode details can be grabbed:

MDB.config.xml settings for series episode detail grabbing:

- Obviously, in mdb.config.xml, this dedicated *MdbIni for series* must be selected e.g.:
  <site series="imdb.com.imdb.series" movies="imdb.com.ask,imdb.com.imdb"></site>
- And, also in mdb.config.xml, the subtitle and the episode number is the required as *mustmatch*
  <matchserie optional="" mustmatch="title,sub-title,episode-num" minimum="2"></matchserie>
  Set *minumum* to 2 or higher to ensure either sub-title and/or episode-num are forced to match!
- Optionally, if episode number matching is requested and the xmltv episode-num doesn't obey one of the standards described below, specify episodenum_pattern:
  Episode number formats in epg's vary from site to site and sometimes even within sites for different channels. Further, also WG++ siteini's sometimes convert them to formats prefered by the user or to the xmltv_ns standard. At the other end, even the mdb sites use various formats. The program must bring (convert) these to one standard internally else a matching is not possible. For the episode-num values in xmltv source file this conversion is done automatically for two input formats: *xmltv_ns* and the *onscreen format Sx Ey* (e.g. S3 E5 = season 3 episode 5). Other onscreen formats need a pattern specification in mdb.config.xml, e.g. :
  <episodenum_pattern site="se.timefor.tv">"Season'S1'Episode'E1'"</episodenum_pattern>
  - *site* is the site name for which the pattern is valid. This must be the site_id value.
  - The *episode_pattern* specification is the same as for SiteIni's. See *(4.6.5.2 Episode-number patterns)*
  - multiple patterns can be specified if a site uses various formats in the xmltv source. The program automatically selects the one that gives the best matching. Each pattern must be enclosed by " ", e.g. :
  <episodenum_pattern site="se.timefor.tv">"Season'S1'Episode'E1'""Season'S1'Episode'E1'/'Et1'"</episodenum_pattern>

MdbIni for series , guidelines:

- See *(6.4.1 MdbIni for series example)*
- *(mdbinitype)* Important value in the general site dependant settings : *site {mdbinitype=series}*
  This will force the postprocessor into the series procedures.
- *(primarysearch)* Set scope = primarysearch:
- Configure an *(url_primarysearch)* with *'title'* and scrub the *(show_id)* candidates (the id of the series as a whole, not one particular episode)
- With this *show_id* configure an *(url_mdb)* that results in a (html) page that contains a list of all the episodes of the series with this *show_id.*
- Configure all the other url's *(urls_series_detail)* that contain series episode details.
- *(match)* Set scope = match
- Next, scrub all the episode titles in *(mdb_episodetitlelist)* and the episode numbers in *(mdb_episodenumlist)*
- The program will now automatically start a matching routine that will select the one *mdb_subtitle* or the *mdb_episode* that matches the (xmltv) values of these best, (following the rules described in *(title_or_number?)* ,resulting in a single value *mdb_subtitle* or *mdb_episode* that matches the xmltv data! No scrubstrings for these elements is necessary because they are created by this routine.
- Scrub *(mdb_title)* because it is on the same html page.
- Scrub the episode details of all the available episodes in a multi value temp element *(all_episodes)*
- *(getelements)* Set scope = getelements
- Configure a temp element to contain the index number of the matched mdb_subtitle (= episode title) *(index_of_episode_title)* or mdb_episode (= episode number) *(index_of_episodenumber)* in the temp element containing *(all_episodes)*
- *(episode_data)* Using this index extract the episode data from *(all_episodes)*
- With this we can scrub all the remaining episode data. Because that is mostly on other html pages we need the *(mdb_episode_id)* first to get the url's expanded.
- With this grab the *(rest_of_the_elements)* using the urls as configured in scope *(primarysearch)*
- The element mdb_episode (the episode number) will have a value already if a match was found between the xmltv episode number and one in the episodenumlist. In that case it will have a fixed format that is used internally to do the matching. If another format is required, grab the element again after clearing the existing one. *(The mdb_episode again)* This must be done at the end of the MdbIni because the program needs the internally obtained value.

6.4.1 MdbIni for series example

The following is taken from MdbIni thetvdb.com.bing.ini, revision 2. Excluding the header section.

```
site {url=imdb.com|mdbinitype=serie|cultureinfo=en-GB|charset=UTF-
8|matchfactor=70|searchsite=imdb|episodesystem=onscreen}
scope.range {(primarysearch)|end}
    *primary search (using bing):
url_primarysearch {url()|https://www.bing.com/search?q=IMDb+|'title'|}
    * https://www.bing.com/search?q=IMDb+Unge+kommissarie+Morse
url_primarysearch.modify {replace| |+}
url_primarysearch.headers {customheader=Accept-Encoding=gzip,deflate}
mdb_show_id.scrub {regex|primary||<a href="http://(?:www\.)?imdb.com/title/tt(\d+?)/"||}
    * imdb url's:
url_mdb.headers {customheader=Accept-Encoding=gzip,deflate}
    * all the episodes data sorted with episode title and mdb_episode_id:
url_mdb_p1.modify {addstart()|http://www.imdb.com/title/tt'mdb_show_id'/epdate}
    * the episode detail page:
url_mdb_p2.modify {addstart()|http://www.imdb.com/title/tt'mdb_episode_id'}
    * the full synopsis:
url_mdb_p3.modify {addstart|http://www.imdb.com/title/tt'mdb_episode_id'/synopsis?ref_=tt_stry_pl}
    * full cast and crew (director, writer, actor):
url_mdb_p4.modify {addstart|http://www.imdb.com/title/tt'mdb_episode_id'/fullcredits?ref_=tt_ql_1}
    * plot summary (not used):
url_mdb_p5.modify {addstart|http://www.imdb.com/title/tt'mdb_episode_id'/plotsummary?ref_=tt_ql_5}
    * user reviews:
url_mdb_p6.modify {addstart|http://www.imdb.com/title/tt'mdb_episode_id'/reviews?ref_=tt_ql_7}
end_scope
scope.range {(match)|end}
    * possible mustmatch elements;
    * episodetitle (sub-title):
mdb_episodetitlelist.scrub {multi()|p1|<a href="/title/tt|/">|</a>|</td>}
    * and/or episode-number:
mdb_episodenumlist.scrub {regex(pattern="'S1'.'E1'")|p1||<td align="right" bgcolor="#eeeeee">(.+?)</td>||}
    * title from p1
    * the 'origina'l title:
mdb_title.scrub {single(separator="(" include=first exclude="</span>")|p1|<span class="title-
extra">||<i>|</span>}
    *  title:
mdb_title.scrub {single(separator="(" include=first)|p1|<title>||</title>|</title>}
mdb_title.modify {cleanup(tags="/=\"")} * removes starting "
mdb_title.modify {cleanup(tags="\"=/")}
    *  all the episodes:
mdb_temp_6.scrub {regex()|p1||<tr>\s+?(<td align="right".+?)</tr>||}
end_scope
scope.range {(getelements)|end}
    * in case of matched subtitle
    * index of the episode in the episodetitlelist:
mdb_temp_1.modify {calculate('mdb_episodetitlelist' not "" type=element format=F0)|'mdb_episodetitlelist'
'mdb_subtitle' @}
    * in case of matched episodenum
    * index of the episode in the episodenumlist:
mdb_temp_1.modify {calculate('mdb_episodenumlist' not "" type=element format=F0)|'mdb_episodenumlist'
'mdb_episode' @}
    * extract the episode data from 'all the episode' in temp_6 using the index in temp_1
mdb_temp_1.modify {substring(type=element)|'mdb_temp_6' 'mdb_temp_1' 1}
    * next, get the elements from mdb_temp_1 (the episode):
    * the mdb_episode_id (= tt nbr):
mdb_episode_id.modify {substring(type=regex)|'mdb_temp_1' "(\d\{7\})/\">"}
    * the following elements are taken from the episode detail page mdb-p2, p4 and p6
mdb_productiondate.scrub {single()|p2|<meta itemprop="datePublished"|content="|"|/>}
mdb_productiondate.modify {calculate(format=productiondate)} * only year allowed!
mdb_category.scrub {regex()|p2||<span class=\"itemprop\" itemprop=\"genre\">(.+?)</span></a>||}
mdb_actor.scrub {multi()|p4|?ref_=ttfc_fc_cl_t|itemprop="name">|</span>|</a>}
mdb_director.scrub {multi|p4|?ref_=ttfc_fc_dr|" >|</a>|</td>}
mdb_starrating.scrub {single()|p2|<div class="ratingValue">|itemprop="ratingValue">|</span>|</div>}
mdb_starratingvotes.scrub {single|p2|<div class="ratingValue">|based on|user ratings|</div>}
mdb_showicon.scrub {single|p2|Poster"|src="|"|"image" />}
mdb_commentsummary.scrub {multi(exclude="SPOILERS ARE INCLUDED""This review may contain spoilers""Add
another review" include=first)|p6|<a href="reviews-index?">|<h2>|</h2>|Add another review}
mdb_review.scrub {multi(exclude="SPOILERS ARE INCLUDED""This review may contain spoilers""Add another
review" include=first)|p6|<a href="reviews-index?">|<p>|</p>\n\n|Add another review}
mdb_plot.scrub {single(separator="<em" include=first)|p2|<h2>Storyline</h2>|<p>|</p>|</div>}
mdb_description.scrub {regex|p2||<meta name=\"description\" content=\"(.+?)\" />||}
    * subtitle when not already done with episodetitlelist
mdb_subtitle.modify {substring("" type=regex)|'mdb_temp_1' "<td><a href=\"/title/tt\d+?/\">(.+?)</a></td>"}
    * episode must be last because it is used to get mdb_temp_1  (the actual episode data from mdb_temp_6)
    * and the following will chance the format which makes it impossible to use it to get temp_1
mdb_episode.modify {clear}
    * use of a dummy loop to set the condition 'mdb_episode' "" for all the operations within the loop:
loop {('mdb_episode' "" max=1)|end}
mdb_episode.modify {substring(type=regex)|'mdb_temp_1' "<td align=\"right\" bgcolor=\"#eeeeee\">(.+?)</td>"}
mdb_temp_3.modify {substring(type=regex)|'mdb_episode' "\.(\d*)"} * episode part
mdb_episode.modify {substring(type=regex)|'mdb_episode' "(\d*)\."} * the season part
    * onsceen format:
mdb_episode.modify {addstart(not "")|S}
mdb_episode.modify {addend('mdb_temp_3' not "")|E'mdb_temp_3'}
    * convert to xmltv_ns if required:
*mdb_temp_3.modify {calculate(not "" format=F0)|1 -}
*mdb_episode.modify {substring(type=regex)|'mdb_episode' "(\d*)\."} * the season part
*mdb_episode.modify {calculate(not "" format=F0)|1 -}
*mdb_episode.modify {addend()|.'mdb_temp_3'.}
end_loop
end_scope
```

7. How to develop a new SiteIni file

7.1 Preparation

- Familiarize yourself with the basics of the SiteIni as described in chapter 4. of this document.
- To develop a new SiteIni file it is necessary to collect information about the url, the structure of the html pages of the EPG of the site for which the SiteIni file is to be created. For that use your internet browser and familiarize yourself with the 'developer' tools that your browser provides. E.g.:
  - In Microsoft IE or Edge this tool is supplied as standard. It can be activated with F12
  - For FireFox an add-on 'Firebug' must be installed
  - Chrome: https://developers.google.com/web/tools/chrome-devtools/
  - Other tools: Fiddler Web Debugger, …
  - More information about these developer tools can be found in *http://devtoolsecrets.com/*
- Development environment: Install and familiarize yourself with SiteIniIDE (see 7.2)
- It helps to look at a few example SiteIni files as provided @*http://www.webgrabplus.com/epg-channels*

7.2 SiteIniIDE

This tool provides a dedicated development environment for SiteIni's. It can be obtained @
http://webgrabplus.com/sites/default/files/downloads/Misc/SiteIniIDE_V0.12.zip
The basics are described in a readme.txt document. Unfortunately, it is limited to Windows due to the use of NotePad++ as editor.
Notice!! At the time of writing this version of this document, a new SiteIniIDE is in preparation that is based on the cross platform ATOM text editor. *(http://www.webgrabplus.com/content/atom-package-github)* The next chapter is still written for SiteIniIDE_V0.12 but the basics will remain the same after the release of the ATOM's base IDE.

7.3 Development steps

1. start SiteIniIDE WG++IDE.exe. It uses a third party text editor NotePad++ which is equipped with the SiteIniIDE language pack as described in *http://www.webgrabplus.com/download/utility/notepad-syntax-highlighting* that provides coloured highlighting for a SiteIni listing.
2. Enter Alt+N to load a template for a new SiteIni. It will prompt for the name. It is essential to use a name that reflects the basic url of the tvguide site for which you want to make the SiteIni. E.g. tvguide.com
   This command will create a folder in the debug area of SiteIniIDE and will also create a SiteIni with the name you have chosen with the .ini file extention (tvguide.com.ini) and filled the basic structure and template scrubstrings and a WebGrab++.config.xml file.
3. Open the SiteIni file. A lot of the scrubstrings in it are disabled (* at the beginning of the line), but some are already enabled to start with. E.g.
   site {url=your_site_name|timezone=UTC+00:00|maxdays=6|cultureinfo=en-GB|charset=UTF-8|titlematchfactor=90|nopageoverlaps}
   urldate.format {daycounter|0}
   url_index{url|http://www.your_site_name}
   url_index.headers {customheader=Accept-Encoding=gzip,deflate} * to speed up the downloading of the index pages
   index_showsplit.scrub {multi(debug)||||}
4. In your internet browser, go to the webpage that displays the tvguide of a certain channel for 'today'. Enable the developer tool as mentioned in *7.1 Preparation* and make notes of:
   - URL : locate the url for the page that contains the actual tvguide data (that is not always the same as the url that shows in the address bar of the browser!). For this url find out :
   - The Webrequest method GET, POST, POST-BACK or SOAP (consult *4.4.1.1*)
     For POST, POST-BACK and SOAP follow the special procedures described in *5.1*
   - The Webrequest headers (see *4.4.1.1*)
   - How the date is specified? Generally, especially in Webrequest method GET, it is part of the url. It can be just a simple number or any other date string. Read *4.4.2.1 urldate format*. It can also be part of the postdata header in case of a POST or an POST-BACK Webrequest method.
   - Similar to date find out where and with what string the requested channel is specified.
   - The URL just located is the URL of the index_page (the page that list all the shows for a certain timespan, just one day in most cases, and for a certain channel) (But there are all kind of other index_page structures like multi page ('subpage'), multi channel, multi day, time fragmented etc)

5. Compose and enter url_index. (*4.4.2 url_index* and an example in *4.4.2.3*)
   Add the debug argument (*4.2.5.5*)
6. Enter urldate.format. The date format as found above in step 4. (Read *4.4.2.1*)
7. (if the index_page is split into several pages, use subpage.format) (Read *4.4.2.2*)
8. Add all the url_index.headers .
9. Enter a few simple values in the line that starts with site (see *4.3 General Site dependent data*):
   – timezone: Enter the timezone in which the data on the index_page is given. This is mostly the timezone of the country. But sometimes the index_page is given in the UTC timezone. See *4.2.7 TimeZones*
   – cultureinfo: The culture info string for the country and the language
   – maxdays: Figure out for how many days the site provides tvguide data.
   – charset: The charset in which the index_page is written. The value is often found near the top of the index_page. If unclear, start with utf-8. If the result of the first run looks garbled, try other values.
10. Leave index_showsplit as given in 3.: index_showsplit.scrub {multi(*debug*)||||}
    This will simply copy the complete index_page into the element index_showsplit and because of the argument *debug* also in the logfile *WebGrab++.log.txt*
11. Now open the config file: *WebGrab++.config.xml*
    http://www.webgrabplus.com/download/config_files contains an example config file including an explanation of all the values.
    Locate the sample channel entry :
    <channel update="f" site="your_site_name" site_id="" xmltv_id="dummy">dummy</channel>
    – Check if the value for site is the name of the SiteIni (ex the .ini)
      E.g as example name given in step 1 : site="tvguide.com"
    – Now enter the value of site_id. This must be the channel name as found in step 4.
      This site_id value will be used as the channel data mentioned in step 4
    – Leave the other values as they are for now
    – Save the config file.
12. Run WebGrab+Plus. To do that press Alt+W with either the config file or the SiteIni file 'open'
13. If all the settings and the SiteIni content is correct, the index_page is downloaded.
    Open the log file, WebGrab++.log.txt .
    Check if the url is properly created as intended. It is listed after the line: *url_index created:*
    Correct the errors and run again if necessary.
14. Open the file named html.source.htm
    That contains the response of the webrequest with the url and the headers as in the SiteIni.
    If all is OK it's the index_page, else review all the settings and check any errors listed in the log file.
    If the index_page is still not downloaded as expected, a cookie file might be needed. Follow the instruction as in *http://www.webgrabplus.com/documentation/configuration/cookie* and try again
15. In the htlml.source.htm file, locate the index_shows. It helps to copy one of them, from start to finish and paste it in an empty new file. You will need it again.
16. Compose the index_showsplit scrubstring. (see *4.5.1* index_showsplit) As described in *4.2.1 scrubstrings* there are two methods available to extract data from webpages: The *'separator string method'* and the *'regular expression method'*. If you are familiar with regular expressions, use that method (see *4.2.1.2* and *4.2.4.3*) and figure out a regular expression that extracts each index_show individually. The scrubstring is then composed like this: index_showsplit.scrub {regex(debug)||regular expression||}
    For the alternative, the *'separator string method'* four separatorstrings have to be determined:
    bs – blockstart, es – element start, ee – element end and be – block end, as explained in *4.2.1.1*
    The scrubstring must be composed like this index_showsplit.scrub {multi(debug)|bs|es|ee|be} or any of the variants described in *4.2.1.1*
17. Run WebGrab+Plus again. Select the logfile tab (still open from step 13), you need to reload it (right click – reload).
    If your scrubstring was using the 'separator string method' you will find the result of the index_showsplit after the line
    [Debug] Elements , type multi applied in this log file.
    If the 'regular expression method' was used you will find the result after a line like (.. is the number of matches):
    [Debug] Found .. match(es):
    Check if the result is as you intended. Correct if not.
    If correct, remove the debug argument from the index_showsplit scrubstring to keep the logfile clean for the next steps.

18. The next step is to get the start and stop times of the index shows. See *4.5.2.1 Time elements*
    (in rare cases the time values are not provided by the tv-guide site in the index shows, but somewhere else, e.g. in the show detail page. The program is able to handle that but we won't discuss that here.) (*4.5.2.1.1 Times from the detail page*).
    You will need to compose the index_start scrubstring as a minimum. The other time elements (index_stop, index_duration and index_date are optional)
    As in step 16, both mentioned methods (as with all the following scrubstrings) can be used to compose this.
    Locate the start time listing in the index shows in the index_showsplit result from step 17. In most cases it will be just a time without a date component. Don't worry, the program will add the date automatically. Compose a scrubstring for this start time using index_start.scrub . Add a debug argument.
    If however, a date component is available, compose the scrubstring to get the whole date and time.
    In most cases the program will be able to recognize the result as a date/time or time, even if the listing uses non-standard date or time format. If a date time error occurs because this recognition fails, you can add a pattern argument, as described in *4.5.2.1 Time elements*
19. Run and check the logfile if the time element(s) are correctly scrubbed.
    If correct, remove the debug argument from the time element(s) scrubstrings
20. Next is the index_title. The procedure is the same, find the regular expression or the separator strings that extract(s) the description. Again add the debug argument to the index_title scrubstring.
21. Same as 19. Step 20 adds the last element that is needed for a *Time and Title* (minimal) xmltv guide. Therefore, if all is right you will now see the shows being grabbed in the console window at the bottom of the SiteIniIde display: innnnnnnnnnnnnnnnnnnnnnn (read the explanation in *2.1 The show update process*)
22. It's time to have a look at your xmltv output file, named guide.xml by default by the SiteIniIde. (If you want another name, change it in the config file, <filename>other path + other name </filename>)
    You can also look at the result with the simple Epg viewer build into the SiteIniIde by opening the file epg.html in your browser. Keep looking at these files during the rest of the steps.
23. Add all the other elements that are listed in the index show, the same way as in step 20 for the index_title.
    E.g. index_subtitle, index_description etc.
24. Show details: Most sites list the show details on separate pages. That is the case when clicking on an index show in your browser opens a new page with more details about that show. If that is not the case, your site has all available details on the index page. A so called 'index_only' tvguide site. If it is such an 'index_only' site, optimize the xmltv listing using the Operations as described in *4.6 Operations* and continue with step 35 (channel list file creation + SiteIni header)
25. In the case of show details on a separate page: Every such page has its own url and set of headers. In your browser, with the development tool enabled as described in *7.1 Preparation*, click on an index_show in the browser and find the url, the Webrequest method and the headers in the same way as in step 4 for the index_page.
    Tip: The url for such a show detail page, very often contains a kind of 'show_id', a string or a number, which you will also find in the index_show listing. Try to match that part of the url with the same string or number in the index_show.
26. index_urlshow. Section *4.4.3 other url elements* explains how to compose this. Add the debug argument.
    Section 4.4.3 uses the simplest way, adding the result of a scrub action to a leadstring using this syntax:
    index_urlshow {url(debug)|leadstring|bs|optional es|ee|optional be}.
    If it is not possible to use this simple method, scrub the 'show_id' separately, e.g. in an index_temp element (see *4.5.3 Special elements*) and compose the index_urlshow with one or more operations (see *4.6 Operations*), e.g. addstart to add strings and the scrubbed 'show_id' together.
27. Run as in step 19 and check the result of the urlshow in the logfile. If correct, remove the debug argument.
28. The next step is to get the show detail elements. For that we first try to copy one of the show detail pages into the file html.source.htm . as follows:
    – add a 'dummy' temp_1 scrub : temp_1.scrub {single(debug)||||}
      This copies the whole show detail page into the temp_1 element, and also writes it into the html.source.htm file because of the debug argument.
    – In the config file, change <timespan>0</timespan> into <timespan>0 *hh:mm*</timespan>,
      in which *hh:mm* is a time within the timeframe of one of the shows, enabling the special 'one show only' mode. For this select a show with as much details as possible.
    – Run again.
29. Now that you have the content of the show detail page in html.source.htm, you can add scrubstrings for all the detail elements you want to add to your listing, starting with the title.

30. Try a few other shows by changing *hh:mm* in the <timespan>
31. Try more a full day by removing the *hh:mm* from the <timespan>
32. Try more days, e.g. 3, by changing <timespan>0</timespan> into <timespan>2</timespan>
33. (In rare cases some of the showdetails are on one or more subdetail page(s). See *4.4.3* and *4.4.3.1*)
34. If needed, use operations to optimize, modify or clean the results. Add argument scope to optimize the SiteIni as explained in *4.6.1.1* and *4.6.1.2*
35. Next steps: Create a Channel File, a xml file that contains the correct channel data for all available channels to be used as channel selection in the config file. As in step 4, in your browser with the development tool enabled, locate a page from the site that contains a kind of listing of all the available channels. Part of that list must have the channel name that you used in step 11 for the site_id, together with another name that is normally used to 'call it' and that people are familiar with. The program uses the first as value for site_id and the latter for xmltv_id and channel name. For example, if the site uses a channel number, like 12 to select the BBC1 channel in the URL, the site_id="12" and xmltv_id="BBC1". The resulting <channel> element in the config for that will look as follows:
    <channel update="f" site="tvguide" site_id="12" xmltv_id="BBC1">BBC1</channel>
    The target for creating the Channel File is to make a list of lines as above stored in a file for all available channels. This file, called 'site'.channels.xml, in the case of this example tvguide.com.channels.xml, will be automatically created by the procedure in the following steps.
36. Locate the following lines at the bottom of the SiteIni :
    ```
    **      #####  CHANNEL FILE CREATION (only to create the xxx-channel.xml file)
    **
    ** @auto_xml_channel_start
    *index_site_channel.scrub {multi|}
    *index_site_id.scrub {multi|}
    *scope.range {(channellist)|end}
    *index_site_id.modify {cleanup(removeduplicates=equal,100 link="index_site_channel")}
    *end_scope
    ** @auto_xml_channel_end
    ```
    This is the section that, when enabled and with the proper scrubstrings for the dedicated elements index_site_channel and index_site_id, will create the channel file.
37. URL : The url of the page that contains the channel list data as mentioned in step 35. By default, the program uses the url_index as already in the SiteIni (step 5.) assuming the channel data is available on the index page. (which is often the case). If it happens to be on another page, the url for that has to be added to the block. Just specify another url_index in the same way as in step 4 and 5. You don't have to disable the one from step 5.
38. The element index_site_channel is used to scrub the channel name and (obviously) index_site_id is for the site_id. Use the same methods as described above to find the scrubstrings. Use the debug argument as before.
39. Enable all lines with a single * at the start of the line. The section also contains a proper scope setting and a line that removes eventual duplicates.
40. Select just one channel in the config file and set timespan to 0 (one day) and run.
    Check if the correct channel file as mentioned in step 35 is created. If correct, don't forget to disable all the lines enabled in the previous step by adding the * at the beginning of the line.
41. Header: Fill out the lines at the top after ** @header_start*
    ```
    * @Site: tvguide.com
    * @MinSWversion: (the program version you used to develop the SiteIni) e.g. 1.1.1/54
    * @Revision 0 – [dd/MM/yyyy] your name
    *    - e.g. creation or update
    * @Remarks: Whatever you like to mention
    ```
    !! Leave the starting * characters !!
42. Save the SiteIni and run the macro WG++ Cleanup (that removes the now obsolete remaining debug arguments)

Share it with other users by posting it on the forum http://webgrabplus.com/forums/ini-files *or ask for help.*
Well done!!.

APPENDIX A WebGrab+Plus Features

- Runs in *Windows*, *Linux*, *OSX*, *Rasberry Pi* and *Synology NAS*
- Can grab from *multiple sites* in one run, programmable by user through a *SiteIni* file.
- As of Januari 2017 SiteIni files available for *464 TV-guide sites worldwide* in *86* countries (see *http://www.webgrabplus.com/epg-channels*)
- *Very fast* through *incremental* grabbing (only changed and new shows grabbed)
- Ability to *import epg data* from (other source) xmltv files and *merge* that with that of the grabbed data.
- *Programmable* through editing commands that enable *changing, filtering, adding, moving, removing (parts)* and *calculation* of (parts of) the xmltv elements.
- Support of *combi-channels*, channels that show programs from different source channels at specific periods of the day.
- Support of *time-offset channels*, channels that only differ from another one through a time shift.
- Ability to grab from a *very wide range of epg structures* of the supported html pages. E.g. *single -day, multi- day, multi-page single-day, channel-fragmented single-day* and many other variants.
- Supports grabbing from *compressed* feeds (gzip and deflate) through build-in decompression.
- Can extract xmltv elements from a very *wide range of webpage data formats* like *html, htmls, ftp, xml, xmltv, mxf, csv , json* and others.
- Very *flexible url-builder* (that builds the url of the index-page) with support for *day-number, weekday-name, weekday-number, date-string, date-number, date-lists* and *sub-pages*.
- Support for *Http* WebRequest methods *GET, POST, POST_BACK* and *SOAP* and the required *header specifications*.
- Support for *Ftp* WebRequest, including access credentials username and password.
- Support for input of a local *File* in stead of a WebRequest for development and testing
- Support of grabbing of *nearly all (27) xmltv* elements.
- Support of grabbing in all *languages* (that can be represented in a standard character set), including the non-alphabetic like Chinese, Russian, Greek, Japanese etc.
- Can grab from *1, 2 or 3 web pages* (index - , detail - and subdetail page)(even multiple subdetail pages)
- Automatic forward looking *DST* (daylight saving time) adaptions .
- Integrated worldwide *TimeZones* database and their DST rules.
- *Fair use of site resources* through user programmable delays between subsequent channels, index pages and detail pages.
- Programmable *retry* and *time-out* settings for bad or slow internet connections.
- Conforms to the '*ROBOTS exclusion standard'* (http://www.robotstxt.org/wc/robots.html) through a screen warning and a user-agent signature.
- Optional *MDB postprocessor* that automatically grabs additional data from *IMDb* and other online movie and serie databases.
- Optional *REX postprocessor* that allows *re-allocation* and *merging* of xmltv elements
- *IDE* available to facilitate development and update of SiteIni files

APPENDIX B Times, time-zones and DST corrections

The two time-zones:

- The first thing to realize is that there are two time-zones involved in the times in your final EPG, each with its own DST rules; the time-zone for which the times of the guide is provided (the 'source' time-zone) and the time-zone of the user (the 'target' time-zone). In the majority of cases these two are equal, but that doesn't change the fact that there are two to consider.
- In principle, the guide times must be corrected for UTC offset and DST changes for both these two time-zones: The first correction, the one for the 'source' time-zone, is done by WG++ and the second, for the 'target' (should be done) by some piece of software at the viewers side, probably the PVR xmltv importer.

The wrong way:

- In earlier versions of WG++ (before version 1.1.1.53) the program was unable to identify the 'source' time-zone correctly, the only info available about it was the time-zone value in the siteini. But that value was limited to just an UTC offset, not the proper time-zone_id and the DST rules for that. (The reason for that limitation lies in the fact that timezone_id's are not standardised among the varies computer platforms, Windows, Linux, OSX, for which WG++ should work) Therefore the DST correction that WG++ applied for the 'source' time-zone was done with the DST rules of the 'target' time-zone. Consequently, the DST changeover dates applied for the source time-zone were the DST changeover dates of the target time-zone. Although wrong in principle, it is only a problem if these DST changeover dates of source and target time-zones are different, which is possible but unlikely. (no problem if the user lives in the 'source' time-zone).
- Situations when this goes wrong? Examples:
    - If a user in Europe wants to use a guide from the US (they have different DST changeover dates), or any other mismatch of DST changeover dates. Or,
    - if the source time-zone has no DST rules (no DST in that zone) and the user lives in a time-zone with DST rules. Or the opposite. Or,
    - if the source time-zone is UTC (the guide times are not in any particular time-zone, just plain UTC or GMT). Any user living in a time-zone with DST rules will run into problems because WG++ will apply the DST rules of the viewer to the xmltv startime which was meant to be UTC without any DST rule.
- The problems explained above can be 'corrected', temporary, by either changing the time-zone value in the siteini or by using the small utility xmltv_time_correct, (available @ http://www.webgrabplus.com/sites/default/files/download/utility/xmltv_time_correct/xmltv_time_correct.zip on the download page) for the period of the mismatch of the DST rules.

The right way:

- To solve these problems, later versions of WG++ include an integrated international 'cross platform' time-zone database that correctly identifies the source time-zone and the DST rule for that.

  With that, WG++ doesn't use the target time-zone any more. The two above mentioned DST corrections are completely separated, WG++ for the source, and the PVR's xmltv importer at the user end for the target.

How to interpret the times in the xmltv file created by WG++? :

- By definition, the xmltv times are in UTC (GMT)!!

- The xmltv specification allows several formats for that and WG++ uses the one with an UTC offset, format *yearmonthdayhourminutesecond +/-utcoffset* or *yyyyMMddhhmmss +/-hhmm*. The first part of that is (in most cases) the date and time presented in the tvguide of the source site, the second part is the actual offset of that with respect to UTC. E.g *start="20140602141500 -0500"* represents a local time at the source of 2014/06/02 14:15 (or 2:15pm) in a time-zone that is 5 hours behind (-) UTC. So, because the result is UTC by definition, this is 2014/06/02 19:15 UTC .
- The eventual DST correction for the source time-zone is done by WG++ through changing the UTC offset with the effect that the result always remains the correct UTC source time for the start of this tv-program.

What about the start-time for the viewer in his time-zone? :

- The calculation, to be carried out by some piece of program at the viewer's side, probably the xmltv importer of the PVR program, must simply correct for the actual local UTC offset (corrected for DST if necessary). As example, a few cases:
  - Suppose the viewer is in the same time-zone as that of the source, which is quite normal. Then the start-time value of 2014/06/02 19:15 UTC will be corrected back to 2014/06/02 14:15.
  - Suppose the viewer's actual UTC offset is +8:00 (Singapore or there about), then this start-time is corrected to 2014/06/03 3:15 (early morning, next day)

What if the times in your EPG appear to be wrong?

- Make sure you have the latest WG++ version that has the intergrated time-zone database.
- Check if the time-zone in the SiteInii you are using is the correct one for the epg source site.
- Check if the times in the xmltv file created by WG++ are indeed as described above (local epg source time and UTC offset)
- Make sure your computer is set to the correct time-zone of your location, enable DST and check the time and date setting
- Try to figure out if the xmltv importer of the PVR you are using works as described above (correct for the actual local UTC offset). (Some importers have problems with xmltv files containing different UTC offsets, reading only the first one found and assuming this to be the only one). In that case a small utility *WG2MP* can help.
- Some other importers use the UTC offset in the xmltv rather than the target timezone offset, in that case there is only : the final resort, temporary offset the guide times by using *http://www.webgrabplus.com/sites/default/files/download/utility/xmltv_time_correct/xmltv_time_correct.zip*

---###############---

APPENDIX C Read-only elements

Read-Only elements, can be used in operation and will be expanded if enclosed by ' ' characters:	
previous_start _stop _duration	Read-Only elements with the value of the previous scrub. (see 4.5.3) They can be useful in operations of other elements.
previous_index_temp_1 to _9	
previous_temp_1 to _9 or (=same): previous_detail_temp_1 to _9	
previous_subdetail_temp_1 to _9	
channel	expands to site_id if used in url builder elements
urldate	expands to the urldate of the url_index with which the actual html page is grabbed
now	expands to the date and time of the moment of expansion
showdate	expands to the epg date of the actual show being processed
config_site_id	expands to the site_id of the channel being processed.
config_site_channel	expands to the site_channel of the channel being processed.
config_xmltv_id	expands to the xmltv_id of the channel being processed.
config_display_name	expands to the display_name of the channel being processed.
config_timespan_days	expands to timespan days of the channel being processed.
config_credentials_user	expands to the credentials username of the channel being processed.
config_credentials_password	expands to the credentials password of the channel being processed.

APPENDIX D Site Dependent settings

General site dependent settings (see 4.3 for more details):			
SiteIni name	**optional**	**explanation**	**example value**
url		The URL of the home page of the site	
timezone		The timezone for which the the tvguide data is given	Europe/Brussels, UTC
maxdays		The maximum amount of days of epg data in the site	14
cultureinfo		The culture-info string for the country of the site	en-UK, fr-FR
charset		The charset(s) in which the html pages are coded	UTF-8 or ISO-8859-1
titlematchfactor		A number that sets the quality of the title comparison	80
ratingsystem	✓	A string that describes the rating system used by the site.	Kijkwijzer, MPAA
episodesystem	✓	A string that describes the value of the episode.	standard values are : xmltv_ns, onscreen
grabengine	✓	No longer in use!	
firstshow	✓	A number that determines which indexshow is the first to use.	3
firstday	✓	An array of numbers that determines the first day in a weekly multiday index page	1234560
subtitlestype	✓	Sets the value of the type attribute of the subtitles element.	teletext , deaf-signed
retry	✓	overrules the <retry> settings in the config file	same syntax as in the config, <retry time-out="5">4</retry>
keeptabs	✓	overrules the standard replacement by spaces of tabs in the html pages	keeptabs
keepindexpage	✓	saves the indexpage(s) for other channels of the same site.	keepindexpage
loadcookie	✓	sends a cookie (saved in a file by the user) as part of the http request	yelo.be.cookie.txt
skip	✓	overrules the <skip> settings in the config file,	same syntax as in the config, <skip>14,1</skip>
compression	✓	enables decompression of compressed site pages and sets the decompression standard	gzip, deflate
nopageoverlaps	✓	to indicate that the site's indexpages have no time overlaps	nopageovelaps
allowlastpageoverflow	✓	allows shows beyond the timespan value if listed on the index page	allowlastpageoverflow

APPENDIX E Element names

SiteIni name	optional	prefix:				Xmltv name (ref xmltv.dtd)	action:					multiple xmltv	multiple scrub	remarks
		global_	index_	none or detail_	subdetail_		.url	.headers	.format	.scrub	.modify			
url_index						-	✓	✓			✓			the url of the show index page
url_preload	✓					-	✓	✓						the preload url if needed
urldate						-			✓					date format for url builder
subpage	✓					-			✓					if the show index page has subpages, format for the url builder
urlshow	*		✓			-	✓	✓			✓			* Only if details from a showdetail html page needs to be grabbed
urlsubdetail	*		✓	✓		-	✓	✓			✓			* Only if details from a subdetail html page needs to be grabbed
urlchannellogo	✓		✓			src *				✓	✓			* attribute of sub-element icon of element channel
showsplit			✓			-				✓	✓		✓	splits the indexpage in shows
date	✓		✓			start/stop *				✓	✓			* date part of xmltv start and stop, when not used, today is used
start			✓	✓		start				✓	✓			
stop	*		✓	✓		stop				✓	✓			* automatic alternative = nextstart
duration	✓		✓	✓		stop *				✓	✓			* when used, stop = start + duration
title	*		✓	✓	✓	title				✓	✓		*	* index_title is obligatory and <u>not</u> multiple scrub
titleoriginal	✓		✓	✓	✓	title *								* distingueshed from title by other lang attribute
subtitle	✓		✓	✓	✓	sub-title				✓	✓		✓	
description	✓		✓	✓	✓	desc				✓	✓		✓	
director	✓		✓	✓	✓	director *				✓	✓	✓	✓	
actor	✓		✓	✓	✓	actor * role **			✓	✓	✓	✓	✓	** actor.format to specify actor-role pattern : role attribute
presenter	✓		✓	✓	✓	presenter *				✓	✓	✓	✓	* sub-elements of element credits
writer	✓		✓	✓	✓	writer *				✓	✓	✓	✓	
producer	✓		✓	✓	✓	producer *				✓	✓	✓	✓	
composer	✓		✓	✓	✓	composer *				✓	✓	✓	✓	
commentator	✓		✓	✓	✓	commentator *				✓	✓	✓	✓	
rating	✓		✓	✓	✓	value *				✓	✓	✓	✓	* sub-element of element rating
ratingicon	✓		✓	✓	✓	icon *				✓	✓	✓	✓	* sub-element of element rating
category	✓		✓	✓	✓	category				✓	✓	✓	✓	
productiondate	✓		✓	✓	✓	date *				✓	✓		✓	* year of production
starrating	✓		✓	✓	✓	value *				✓	✓	✓	✓	* sub-element of element star-rating
episode	✓		✓	✓	✓	episode-num				✓	✓		✓	
showicon	✓		✓	✓	✓	src *				✓	✓		✓	* attribute of element icon
country	✓		✓	✓	✓	country				✓	✓		✓	
url	✓		✓	✓	✓	url				✓	✓		✓	
subtitles	✓		✓	✓	✓	subtitles *				✓	✓		✓	* 'boolean' type elements
premiere	✓		✓	✓	✓	premiere *				✓	✓		✓	no value, when 'true' listed like
previousshown	✓		✓	✓	✓	previously-shown *				✓	✓		✓	<subtitles/>
videoaspect	✓		✓	✓	✓	aspect *				✓	✓		✓	* sub-element of video
videoquality	✓		✓	✓	✓	quality *				✓	✓		✓	
temp_1 to temp_9	✓	✓	✓	✓	✓	-				✓	✓			general purpose 'none xmltv' elements (see 4.5.3)
variable_element	✓		✓			-				✓	✓			a variable in scrubstrings (see 4.5.3)
site_channel	✓		✓			-				✓	✓		✓	to create a channel-
site_id	✓		✓			-				✓	✓		✓	list file
sort_by		none				-				✓	✓			required with command sort (see 4.6.4.9)