

Unit Miner

Reference manual

Version 1.4.1

Table of contents

Introduction	4
Features and benefits	4
Installation	5
Installation – Unit Miner	5
Installation – Unit Miner Distributed Server	5
How to start Unit Miner Distributed Server ?	6
How to execute your wrapper scripts using Unit Miner ?	7
Command line	7
Web	7
In your php application	7
How to execute your wrapper scripts using Unit Miner Distributed Server?	8
Wrapper script language definition	9
Flow control tags	9
Main command	9
Comments	9
Line break	9
Reserved context variables	10
<Section>	10
<Section Or>	11
<Section Foreach>	12
<Section While>	13
<Section WhileANY>	14
Executive tags	15
<Pattern>	15
<Pattern CSV>	16
<Action>	18
<Action ContentFile>	18
<Action ContentURL>	19
<Action ContentVariable>	20
<Action URLToFile>	21
<Action Email>	21
<Action Eval>	22
<Action ConvertToXLS>	22
<Action ContentPOP3>	23
<Action Anonymizer>	23
<WrapperServer: Action Schedule>	24
<WrapperServer: Action Eval>	24
<Action Print>	25
<Action Sql>	25
<Action SaveDbRow>	26
<Action SaveCSV>	26
<Action Php>	27
<Logger>	28
<Logger Browser>	28
<Logger File>	28
<Logger Mail>	28
How to extend Unit Miner with new tags	29

27/12/2007

Unit Miner

Frequently asked questions 33
System requirements 34
Getting Help..... 35

Introduction

Do you search for complete solution, which will allow you to retrieve clean data from any raw content from any website in Internet? Yes, Unit Miner is the right tool for you.

Unit Miner provides the industry's best solution for data mining. This next-generation automated data mining solution deploys the concept of pattern matching to radically simplified scripts creation and maintenance.

Unit Miner satisfies the needs of both technical and no-technical users. It enables you to develop data mining algorithms for various types of problems. From basic scripts, which will match easy structured contents to complicated scripts, which can read data from multiple contents with complicated data structure. Unit Miner allows even novice users to be productive in minutes.

Unit Miner is highly flexible and user friendly tool for content retrieving and parsing clean data from any raw data. User can define all in highly flexible language designed exactly for user's needs. For retrieving clean data from html content are used configurable templates, which use pattern matching. Scripting language supports simple copy and paste pattern creation. Unit Miner's wrapper script cover wide range of data mining complexity starting with simple patterns for matching simple values on single web page up to complex scripts which consists from multiple wrapper scripts and can navigate and download automatically next web pages depending on previously matched data.

You can create web robots to grab content from other sites and store it to your database, or use it as input to your system.

Features and benefits

Ensures immediate return on investment through industry-leading ease of use environment support

Operates stand-alone as command line application, or integrated into any PHP application

Introduces pattern driven matching technology allowing fast script creation, easier maintenance and powerful data mining capability

Identifies data inside source content, even if the content change slightly formatting, enabling reliable unattended script execution

Reduce frequently repeated manual work

Simple copy & paste pattern creation

Flexible scripting language

Allows creating complex multi step procedures

Matched data save to database, export to file (in multiple formats), send via email or trigger event in external application

User interface for scheduling and editing scripts (only in Unit Miner Distributed Server release)

Installation

Installation – Unit Miner

The basic Unit Miner distribution contains all that is needed to run the wrapper scripts. Before installing Unit Miner, please read carefully part of this documentation, which describes system requirements of Unit Miner.

Installation of Unit Miner is very simple and consists just from few basic steps:

1. Download Unit Miner installation files (zipped package)
2. Extract downloaded file to any directory
 - e.g. with command: `tar -zxvf UnitMiner.tar.gz`
 - or with any other tool, which can handle compressed files
3. Installation is ready

Installation – Unit Miner Distributed Server

Distribution of Unit Miner Distributed Server contains all files needed for execution of all types of retrieving tasks. Before installation of Unit Miner Distributed Server please read documentation and all requirements of this release.

Installation steps:

1. download distribution of Unit Miner Distributed Server from our members area
2. extract files to any directory in web server root (needs to be accessible from internet)
 - distribution is compressed with tar and gzip. For extraction of files you can execute following:
 - Command on unix systems: `tar -zxvf UnitMiner.tar.gz`
 - On windows you can use any tool which supports gzip and tar (like WinZip, WindowsCommander, and many others)
3. Install empty mysql database
4. edit WrapperServer/Settings/settings.php file and define all database constants in this file
 - Make settings.php file writable for web server.
5. Open **http://yourserver/your_installation_directory/install.php** using web browser – it will start installation wizard
6. Follow instructions in installation wizard
7. Start server process – details you will find in next chapter
8. define server – you can add server with same ID as you specified in settings.php or during installation in menu entry Servers

How to start Unit Miner Distributed Server ?

On this place we will describe how to start Unit Miner Distributed Server.

For correct work of Unit Miner Distributed Server should run always in background server process, which periodically checks, if all processes needed for execution of wrapper tasks are running.

It's possible to start it in 2 different modes:

1. periodically start script InstallDir/WrapperServer/start_server_and_die.php as cron task (unix)

This setting will guarantee higher stability which will be protected against server restarts, or against killing of processes by other person (after restart will start server automatically or if process will be killed, it will start next minute automatically).

We recomend, that script will be executed each minute (this will be minimum time how mutch will wait task for execution if no process will run).

Script will try if there is any sheduled task and if is created enough processes for these tasks (maximum processes defined in server setting – menu Servers) and right after will finish.

Example of command, which will schedule periodically script:

Under unix in cron table:

```
* * * * * /usr/bin/php /www/data/WrapperServer/start_server_and_die.php
```

2. start server with script InstallDir/WrapperServer/start_server.php once and this process will run in background whole time and will check periodically if enough processes are running. Time between checks will process sleep (parameter STARTER_SLEEP in settings.php), so CPU resources of server will not be used by this process.

Administrator should ensure, that this script is always running in background. Otherwise will not be guaranteed, that server will work correctly.

Example of command under unix (start process in background):

```
/usr/bin/php /www/data/WrapperServer/start_server.php > server.log &
```

Example of command under Windows:

```
C:/php/php.exe c:/wwwroot/UnitMiner/WrapperServer/start_server.php > server.log
```

Please don't hesitate to contact our support if you will need help with installation of Unit Miner Distributed Server. Installation of this product is free for our customers.

How to execute your wrapper scripts using Unit Miner ?

Now you can try to compose your first wrapper script and execute it. We support 3 types of script execution:

Command line

With command line (your current directory should be set to the same directory, where you installed Unit Miner):

```
[path_to_php] wrapper.php name_of_script.w
```

Where [path_to_php] is complete path to your php command.

On windows wrapper.php script has to be executed with php-cli.exe

Windows: *c:/php/php.exe wrapper.php Wrapper/samples/demo/bbcnews.w*

UNIX: */usr/local/php/php wrapper.php Wrapper/samples/demo/bbcnews.w*

Web

If you prefer to execute scripts from web interface, you can use e.g. following syntax:

http://your_server_domain/path_to_UnitMiner/wrapper.php?script=your_script_name.w

Please don't forget, that maximum execution time is on most of www servers limited for php to 30 seconds. If your script will parse online contents from other servers and response time of other servers will be poor, you can easily reach the limit and script will fail with error, that maximum execution time was reached.

Example:

http://www.qualityunit.com/UnitMiner/wrapper.php?script=Wrapper/samples/online_demo/bbcnews.w

In your php application

If you like to include Unit Miner into your php application, you can do it in following way:

```
<?
// include main library
require_once('QUnit/Global.class.php');

// create executor object with script name which should be executed
$executor = QUnit_Global::newobj('Wrapper_Executor', 'your_script_name.w');

// execute Unit Miner script
$executor->execute();
?>
```

How to execute your wrapper scripts using Unit Miner Distributed Server?

Unit Miner Distributed Server is standalone web application, which can execute retrieving tasks defined by wrapper scripts.

1. After installation of Unit Miner Distributed Server you can login into application using username and password, which you specified during installation.
2. Before we start to write scripts, please check following:
 - you defined server under menu entry Servers and your server is activated
 - you started up server (there are active processes under section Processes)
3. As next step we will create category under menu entry categories. It allows you to group scripts into categories.
4. If we have already category created, we can open directly menu entry Unit Miner scripts. You can edit existing scripts or add new script with link "Add new Unit Miner script". It will open new window, where you can create new script.
5. If you created script, you can schedule script for execution under menu entry Scheduler and using link Schedule script, which will open new window where you can schedule script for execution.
6. If server is running, it will start scheduled task with first free process if it's in queue on first place.
7. Overview of tasks (not started / started / finished / failed) you will find under menu Schedules, where you can use different filters.

Wrapper script language definition

This section describes Unit Miner script language (later just *wrapper script*). Design of wrapper script language is similar to design of XML documents, which helps easy writing and understanding of wrapper script documents. Wrapper script commands are described with tags.

Flow control tags

Main command

Each wrapper script should contain command *Main*. Command defines starting tag with which will begin evaluation. Specified should be always one of tags (Section, Action, Pattern), which is defined on top layer in wrapper script and has name.

Example:

Example shows how to define section and declare, that execution should start with evaluation of this section:

```
<Section>
    Name MyFirstSection
    ...
</Section>
Main MyFirstSection
```

Comments

Wrapper script supports UNIX shell-style comments. All what follows after character # is taken as comments until end of the line.

Example:

```
#here is definition of my first section
<Section>
    Name MyFirstSection
    ...
</Section>
#start evaluation with MyFirstSection
Main MyFirstSection
```

Line break

If your script contains too long lines, you can divide them to more lines using character \. This character will mark end of line, which continues on next line without evaluating as new line.

Example:

```
<Section>
    Name This is my too long name \
        in two lines
    <Pattern>
        RegExp ^<tr align=center*><td>{${match_id}}|
            <td align=left>{${team:text}}|
            <td>{${win_team2_draw:real_null}}<td>{${time}}<br>
    </Pattern>
</Section>
```

Reserved context variables

During execution of wrapper scripts are registered some variables to context:

\$_ITERATION	- contains iteration number of each <Section While> and <Section WhileAny>
\$_SCRIPT	- contains script name, which is currently executed
\$_FILE	- if content loaded from file, inside \$_FILE is filename of this file
\$_NOW	- returns current date and time in format Y-m-j H:i:s
\$_CONTENT	- returns current content

<Section>

Defines order of sub elements evaluation. Sub element can be element of type: *Pattern, Section (including While and While ANY type), Logger, Content and Action.*

Execution will finish if any of sub elements returns FALSE.

Attributes:

Name	- defines section name (optional)
NoContext	- each section before evaluation creates own context, which is assigned as child to parent context. If is defined attribute NoContext new context will not be created for this section and will be used context from parent object during evaluation. In evaluation process can other elements access variables of current context and all his parents.
TryAll	- defines if iteration will stop and will continue with next iteration in case sub element returned TRUE or if all sub elements will be executed in current iteration - All sub elements will be processed regardless of return value
EndAt	- defines end position in current content for this section. Evaluation of section will be stopped if evaluation will reach position marked by EndAt pattern. EndAt attribute should be followed by pattern text (for details how to create it see documentation of attribute RegExp in <Pattern> tag) EndAt can be used more than once in one section. This way you can specify multiple patterns defining end of section content.
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
Define	- adds variable value to current level context

Output:

Returns TRUE in case all sub elements returned TRUE or was defined attribute Optional.

Returns FALSE in case any sub element returned FALSE.

<Section Or>

Defines order of sub elements evaluation. Sub element can be element of type: *Pattern, Section (including Or, While and While ANY type), Logger and Action*.

Processing ends if any of sub elements returns TRUE or is executed also last subelement.

Attributes:

Name	- defines section name
NoContext	- each section before evaluation creates own context, which is assigned as child to parent context. If is defined attribute NoContext new context will not be created for this section and will be used context from parent object during evaluation. In evaluation process can other elements access values of current context and all his parents.
TryAll	- defines if iteration will stop and will continue with next iteration in case sub element returned TRUE or if all sub elements will be executed in current iteration
EndAt	- defines end position in current content for this section. Evaluation of section will be stopped if evaluation will reach position marked by EndAt pattern. EndAt attribute should be followed by pattern text (for details how to create it see documentation of attribute RegExp in <Pattern> tag) EndAt can be used more than once in one section. This way you can specify multiple patterns defining end of section content.
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
Define	- adds variable value to current level context

Output:

Returns TRUE in case one of sub elements returned TRUE.

Returns FALSE in case no sub element returned TRUE.

<Section Foreach>

Defines order of sub elements evaluation. Sub element can be element of type: *Pattern*, *Section (including Or, While and While ANY type)*, *Logger and Action*.

Section iterates through all elements of defined array.

Processing ends if any of sub elements returns FALSE or section finished last iteration with last element of array.

Attributes:

Name	- defines section name
NoContext	- each section before evaluation creates own context, which is assigned as child to parent context. If is defined attribute NoContext new context will not be created for this section and will be used context from parent object during evaluation. In evaluation process can other elements access values of current context and all his parents.
EndAt	- defines end position in current content for this section. Evaluation of section will be stopped if evaluation will reach position marked by EndAt pattern. EndAt attribute should be followed by pattern text (for details how to create it see documentation of attribute RegExp in <Pattern> tag) EndAt can be used more than once in one section. This way you can specify multiple patterns defining end of section content.
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
Define	- adds variable value to current level context
ForeachVariable	- defines array name, which should be used during section execution.

Example of syntax is: *ForeachVariable @myArrayVar*

In each iteration of section will be filled context variable with name \$myArrayVar with value from array.

Output:

Returns TRUE in case all iterations of section are successful.

Returns FALSE in case no sub element returned TRUE.

<Section While>

Defines order of sub elements evaluation. Sub element can be element of type: *Pattern, Section (including While and While ANY type), Logger and Action*. Number of iteration you can access in context variable \$_ITERATION.

Number of iterations:

Unlimited while all sub elements evaluates to TRUE. Iteration ends when first sub element evaluates to FALSE. Next iteration is started in case in last iteration all sub elements returned TRUE.

Attributes:

Name	- defines section name
NoContext	- each section before evaluation creates own context, which is assigned as child to parent context. If is defined attribute NoContext new context will not be created for this section and will be used context from parent object during evaluation. In evaluation process can other elements access values of current context and all his parents.
TryAll	- defines if iteration will stop and will continue with next iteration in case sub element returned TRUE or if all sub elements will be executed in current iteration
EndAt	- defines end position in current content for this section. Evaluation of section will be stopped if evaluation will reach position marked by EndAt pattern. EndAt attribute should be followed by pattern text (for details how to create it see documentation of attribute RegExp in <Pattern> tag) EndAt can be used more than once in one section. This way you can specify multiple patterns defining end of section content.
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
MaxIterations	- defines maximum number of iterations, which can be evaluated in this section in one execution. After defined number of iteration will execution continue with next tag followed after this section.
Define	- adds variable value to current level context

Output:

Returns TRUE in case one or more iterations were finished.

Returns FALSE in case during first iteration evaluates any sub element to FALSE.

<Section WhileANY>

Defines order of sub elements evaluation. Sub element can be element of type: *Pattern, Section (including While and While ANY type), Logger and Action*. Number of iteration you can access in context variable \$_ITERATION.

Number of iterations:

Unlimited while any sub element in iteration evaluates to TRUE. Iteration ends when first sub element evaluates to TRUE and starts next iteration. Next iteration is not started when all sub elements returned FALSE.

Attributes:

Name	- defines section name
TryAll	- defines if iteration will stop and will continue with next iteration in case sub element returned TRUE or if all sub elements will be executed in current iteration
NoContext	- each section before evaluation creates own context, which is assigned as child to parent context. If is defined attribute NoContext new context will not be created for this section and will be used context from parent object during evaluation. In evaluation process can other elements access values of current context and all his parents.
EndAt	- defines end position in current content for this section. Evaluation of section will be stopped if evaluation will reach position marked by EndAt pattern. EndAt attribute should be followed by pattern text (for details how to create it see documentation of attribute RegExp in <Pattern> tag) EndAt can be used more than once in one section. This way you can specify multiple patterns defining end of section content.
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
MaxIterations	- defines maximum number of iterations, which can be evaluated in this section in one execution. After defined number of iteration will execution continue with next tag followed after this section.
Define	- adds variable value to current level context

Output:

Returns TRUE in case iteration was finished (minimum one sub element returned TRUE in first iteration).

Returns FALSE in case all sub elements return FALSE in first iteration.

Executive tags

<Pattern>

Matches defined pattern in content from current position. Match moves cursor in content at the end of match. Sub element can be element of type: *Action* and *Logger*. Inside Pattern definition can be used variables, which values will be after successful match added to current context.

Attributes:

Name	– defines pattern name
Mandatory	– defines if pattern has to be evaluated to TRUE, otherwise is logged to all relevant loggers error. This attribute is useful to set if you like to monitor, if script works fine. Example can be, that you know, that in retrieved document is always at the end of document any word. If for any reason document doesn't contain this word, it means, that retrieving process had possibly any problems (e.g. design of page was changed). In this case you can define logger which will log all errors and send them as email to administrator or person, which is responsible for wrapper scripts execution.
RegExp	– defines pattern, which is searched from current cursor position in content Patter can contain variables definitions, which are loaded from content to context variables and will be accessible for other tags in followed execution. Variable name is defined in form: <pre>{\$variable_name[:variable_type]}</pre> Variable type can contain following values: <i>date, datetime, time, int, integer, real, real_null, email, url, text</i> Or can contain regular expression: <pre>{\$variable_name[:regexp(/a-z*)]}</pre> Or alternatively <pre>{\$variable_name[:re(/a-z*)]}</pre> If variable_type parameter is not used, default will by used type <i>text</i> . Pattern can contain also wildcard characters: * - will match any characters until '<' ([^<]* - regular expression) *? – lazy (as less as possible) wildcard for any character until character '<' ([^<]*? - regular expression) ^ - match beginning of the content \$ - match end of the content As wildcard can be used any regular expression with syntax: <pre>{:regexp(/a-z*)}</pre> Or alternatively <pre>{:regexp(/a-z*)}</pre> If RegExp contains more rows, end of line should be marked with character \ Example: <pre>RegExp ^<tr align=center*><td>{\$match_id}\ <td align=left>{\$team:text}\ <td>{\$win_team2_draw:real_null}<td>{\$time}
</pre>
MultiLine	- if attribute defined in pattern tag, regular expressions and variables inside pattern will match also newline characters as other characters

Substitution	- defines substitution of text with defined variable value before pattern is applied Usefull for dynamically generated patterns. Attribute should be followed by string identifying position in pattern, which should be replaced and name of variable, which should be used during replacement Substitution what_to_replace \$variable Example: <Pattern> <i>Substitution replace_symbol \$symbol</i> <i>RegExp <td*?><a*?>*/a>replace_symbol</td></i> </Pattern>
Trim	- trims spaces from matched variables values
Compact	- strips excess whitespace from matched variable values
DontMovePosition	- after match don't move position of content. This switch will allow to use more patterns in script without keeping the order of patterns as they appear in source text document loaded in content.
Optional	- evaluates pattern TRUE regardless of pattern match
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
URLEncode	- returns a variable in which all non-alphanumeric characters except <code>_-.</code> have been replaced with a percent (%) sign followed by two hex digits and spaces encoded as plus (+) signs. It is encoded the same way that the posted data from a WWW form is encoded.
URLRawEncode	- returns a variable in which all non-alphanumeric characters except <code>_-.</code> have been replaced with a percent (%) sign followed by two hex digits.
URLDecode	- decodes any <code>%##</code> encoding in the given pattern. Decodes plus symbols (+) into spaces.
URLRawDecode	- returns a variable in which the sequences with percent (%) signs followed by two hex digits have been replaced with literal characters. Does not decode plus symbols (+) into spaces.
TagsToStrip	- defines list of tags, which will be removed from loaded content
HtmlSpecialChars	- transforms HTML special chars like <code>&amp;</code> to theirs readable values

Output:

Returns TRUE in case pattern matched data in content. Move position cursor to new position (end of matched data). Fill in matched variables to context variables.

Returns FALSE in case pattern didn't mach.

<Pattern CSV>

Matches all columns on current position of csv file. Tag will match values into named columns until will be reached end of file. Each iteration will match just one row from CSV file loaded into content. Default name of column is \$col1, \$col2, ... \$colN.

Attributes:

Name	- defines pattern name
Column	- gives a name to column - defines name of variable, where will be loaded value from column. Order of Column attributes is the same as columns in csv file. Value from column will be loaded to variable name with defined name.

Example:

Column \$first_column

Column \$second_column

Output:

Returns TRUE in case pattern matched data in content. Move position cursor to next row. Fill in matched columns to context variables.

Returns FALSE in case pattern didn't match – e.g. reached end of file.

<Action>**<Action ContentFile>**

Action ContentFile replaces current content value with content of specified file.

Attributes:

Name	- defines action name
NewLineToBR	- replaces new line characters in loaded content with tag
BRToNewLine	- replaces tags with new line characters in loaded content
NewLineToSpace	- replaces new line characters in loaded content with spaces
RemoveNewLine	- removes from loaded content new line characters
TagsToStrip	- defines list of tags, which will be removed from loaded content
KeepOnlyTags	- defines tags which should not be removed, all other will be removed from content
ReplaceString	- defines pattern and its replacement separated by space; all occurrences of pattern in content will be replaced with replacement. For details how to create pattern see documentation of attribute RegExp in <Pattern> tag.
CompactBefore	- calls Compact as the first operation
CompactAfter	- calls Compact as the last operation
StripAttributes	- strips attributes from HTML tags
StripAttributesFor	- strips attributes from given HTML tags
StripTagsWhereClass	- strips attributes from HTML tags, which have given class attribute
FileName	- name of file, which should be loaded to current content
Directory	- loads files from selected directory
FileSuffix	- defines filter for filenames loaded from specified directory in attribute Directory
BufferSize	- defines size of buffer, if not specified complete file will be loaded to memory - for files bigger as 2MB is better to use BufferSize attribute, because of performance reasons. - good value as buffer size is 2 kilo bytes (BufferSize 2000) - you cannot use BufferSize when using ReplaceString
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.

Output:

Action returns always TRUE during execution. Content of file replaces current content on which runs execution.

<Action ContentURL>

Action ContentURL replaces current content value with content of web page specified with URL.

Attributes:

Name	- defines action name
NewLineToBR	- replaces new line characters in loaded content with tag
BRToNewLine	- replaces tags with new line characters in loaded content
NewLineToSpace	- replaces new line characters in loaded content with spaces
RemoveNewLine	- removes from loaded content new line characters
TagsToStrip	- defines list of tags, which will be removed from loaded content
KeepOnlyTags	- defines tags which should not be removed, all other will be removed from content
ReplaceString	- defines pattern and its replacement separated by space; all occurrences of pattern in content will be replaced with replacement. For details how to create pattern see documentation of attribute RegExp in <Pattern> tag.
CompactBefore	- calls Compact as the first operation
CompactAfter	- calls Compact as the last operation
StripAttributes	- strips attributes from HTML tags
StripAttributesFor	- strips attributes from given HTML tags
StripTagsWhereClass	- strips attributes from HTML tags, which have given class attribute
URL	- url to web page, which should be loaded to current content
Referer	- HTTP Referer URL
Post	- defines variables, which should be sent through POST method
Cookie	- defines variables, which should be sent through COOKIE
CookieFileName	- defines name of file, where should be stored cookies
Header	- inserts special header into request, attribute can be repeated for each header line
AuthUser	- defines user name used for basic authentication
AuthPassword	- defines password used in basic authentication
Timeout	- defines maximum timeout, which should script wait for answer from server
Multipart	- sends post variables in multipart form
NoRedirect	- doesn't follow header redirections
ProxyIP	- defines IP address or host of proxy server
ProxyPort	- defines port of proxy server (default 8080 when not specified)
ProxyUser	- defines username required for proxy server authentication
ProxyPassword	- defines password required for proxy server authentication
AutoredirectWithCookie	- follows automatically redirects sent from server and send to redirected page also cookies registered from server in request. This option is often needed for asp applications.
GetHeaders	- if this attribute will be specified, as content will be returned also Header sent from server. It's useful, if you need to parse also e.g. cookie values returned from server.
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
UseAnonymizer	- defines the use of anonymizer (proxy server)

RandomizeProxy	- defines that proxies will be used randomly
RandomizeProxyOnError	- defines that proxies will be used randomly when an error occurs
LowSpeedLimit	- defines the transfer speed in B; if transfer is below this speed for LowSpeedTime seconds it will be aborted
LowSpeedTime	- defines the time in seconds; transfer below LowSpeedLimit will be aborted after this time
AutoRetryNoContent	- will retry if no content was downloaded (e.g. problem with server); has two parameters separated by blank: time in seconds to wait before retry and maximum number of retries
AutoRetryHTTPErrors	- defines comma separated list of HTTP error codes (e.g. 404,403). If these codes are returned after download of content, it is considered an error and will retry. Number of retries is defined by AutoRetryNoContent.
AutoRetryPattern	- will retry if defined pattern was found
AutoRetryNoPattern	- will retry if defined pattern was not found
DownloadLimit	- maximum number of bytes downloaded to content

Output:

Action returns always TRUE during execution. Content of web page replace current content on which runs execution.

<Action ContentVariable>

Action ContentVariable replaces current content value with content of context variable filled during previous execution. Example of use can be, when you know borders of value, which you like to parse with next script.

Attributes:

Name	- defines action name
NewLineToBR	- replaces new line characters in loaded content with tag
BRToNewLine	- replaces tags with new line characters in loaded content
NewLineToSpace	- replaces new line characters in loaded content with spaces
RemoveNewLine	- removes from loaded content new line characters
TagsToStrip	- defines list of tags, which will be removed from loaded content
KeepOnlyTags	- defines tags which should not be removed, all other will be removed from content
ReplaceString	- defines pattern and its replacement separated by space; all occurrences of pattern in content will be replaced with replacement. For details how to create pattern see documentation of attribute RegExp in <Pattern> tag.
CompactBefore	- calls Compact as the first operation
CompactAfter	- calls Compact as the last operation
StripAttributes	- strips attributes from HTML tags
StripAttributesFor	- strips attributes from given HTML tags
StripTagsWhereClass	- strips attributes from HTML tags, which have given class attribute
Variable	- defines context variable name which contains new value of content

Output:

Action returns always TRUE during execution. Content of context variable will replace current content on which runs execution.

<Action URLToFile>

Action loads content from specified URL and store it to specified file on disk.

Attributes:

Name	- defines action name
URL	- url to web page, which should be loaded to current content
Post	- defines variables, which should be sent through POST method
Cookie	- defines variables, which should be sent thought COOKIE
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
FileName	- filename, where should be stored loaded content from URL
DontSaveIf	- defines size of a file, which won't be downloaded

Output:

Action returns TRUE if loading and saving of content was successful. If any error will occur during execution, returns FALSE.

<Action Email>

Sends email to specified recipients. Example of usage can be, that you will match all email addresses in retrieved content and will send email to each email address. Or next example can be, that clean data will be sent to recipient, which waits for these data.

Attributes:

Name	- defines action name
To	- defines list of recipients (separated by comma)
Cc	- defines list of recipients (separated by comma) for carbon copy
Bcc	- defines list of recipients (separated by comma) for blind carbon copy
From	- defines email, from which was sent email (sender)
Subject	- defines subject of email
Body	- defines body of email
Attachment	- defines file name of attachement (full or relative path). Attribute can be used in same tag more times if needed to attaché more files

Output:

Returns true if email was sent without errors.

<Action Eval>

Executes wrapper script stored in specified file. Evaluated wrapper script will share context and global loggers with currently evaluated script. Content will not be shared between scripts. Also context variables filled in evaluated script will not be accessible in parent script, which called evaluated script.

Attributes:

Name	- defines action name
File	- source of wrapper script, which should be evaluated
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.

Output:

Returns result of defined external wrapper script evaluation.

<Action Exec>

Executes external command.

Attributes:

Name	- defines action name
CMD	- defines external command
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.

Output:

Returns result of defined external command.

<Action ConvertToXLS>

Converts specified CSV file to Excel (XLS) format.

Attributes:

Name	- defines action name
InputFile	- defines input file name
OutputFile	- defines output (converted) file name
Separator	- defines values separator which is used in CSV file
WorkSheet	- defines name of worksheet in XLS file
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
Encoding	- defines encoding of the XLS file
OmitFirst	- omits the first line of the CSV file (usually names of the columns)

Output:

Returns TRUE if there were no errors, otherwise returns FALSE.

<Action ContentPOP3>

Action ContentPOP3 enables to read content from POP3 server (download email messages).

Action ContentPOP3 replaces current content value with actual email message downloaded from server. It is usually used in Section While statement to download all messages.

Attributes:

Name	- defines action name
Host	- defines mail server host
Port	- defines mail server port
Username	- defines login to server
Password	- defines password
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.

Output:

Returns TRUE if connection to server was successful an message was downloaded, otherwise returns FALSE.

<Action Anonymizer>

Defines anonymizer (proxy server) which will be user in Action ContentURL.

Attributes:

Name	- defines action name
ProxyIp	- define IP of proxy server
ProxyPort	- define port of proxy server
ProxyUser	- define login to server
ProxyPassword	- define password
CheckURL	- verifies if the data received from proxy server are relevant. Defines URL which needs to be checked.
CheckPattern	- defines pattern which needs to be found on site defined by CheckURL
GetStatus	- if not set, returns always true

Output:

If proxy server responds, adds it to the list of proxy servers. Always returns true.

<WrapperServer: Action Schedule>

This tag exists just in Unit Miner Distributed Server. Tag will schedule execution of script to tasks queue at specific time with defined priority. Execution of this task will start right after defined time if will exist any free process, which can execute this task. This way of tasks execution (scheduling for next process) will enable you to use parallelism as much as possible and developer should use this tag on all possible places. Sometimes is needed, that we need to know result of evaluation from script. In that case is not possible to use this tag, because it will return always true if task was scheduled in queue of tasks.

Attributes:

Name	- defines action name
ScriptName Optional	- name of scheduled script (script has to exist in list of scripts under menu Unit Miner scripts) - evaluates section always to TRUE (also in case any subelement will return FALSE)
Priority	- defines priority of script in queue of tasks (default 10)
Variable	- defines name of parent task context variable, which should be accessible as context variable in child task. This attribute should be repeated for each variable, which you need to share in child task from parent task
ScheduleAt	- defines time when should be task started (if not specified, start immediately with first free process)

Output:

Return true if task was scheduled successfully.

<WrapperServer: Action Eval>

This tag exists just in Unit Miner Distributed Server. Should be used just in case, that in parent script you need status of execution of executed child script. Otherwise should be used in Distributed Server release tag <WrapperServer: Action Schedule>, which will schedule execution of task for next process and optimize performance.

Attributes:

Name	- defines action name
ScriptName Optional	- name of executed script (script has to exist in list of scripts under menu Unit Miner scripts) - evaluates section always to TRUE (also in case any subelement will return FALSE)

Output:

Return TRUE if whole script was evaluated to TRUE, otherwise returns FALSE.

<Action Print>

Prints text to standard output or to file, if filename will be defined. Replace variable names with their values from context.

Attributes:

Name	- defines action name
Text	- defines string, which should be printed to standard output, variables are replaced with their values in context.
FileName	- if not defined, all is printed to standard output, otherwise to defined file
FileMode	- if set to Write, text is written to clean file and not appended to existing file
DontStripSlashes	- if defined, slashes from print content will not be stripped
Flush	- if action contains this attribute and is used just for printing to default output, after each execution will flush buffer (e.g. will write output to browser immediately)

Output:

Returns always TRUE

<Action Sql>

Executes specified sql command. If will be executed SQL Select, than the record will be stored to context variables. If will be executed tag more times (e.g. in <Section While>), than will be stored to context each iteration one row from record set. If no next row will be found in record set, tag will return false. All other statements will be executed each time.

Attributes:

Name	- defines action name
DBType	- database type – tested with: mysql, mssql, oracle, available are also other types of databases (sqlite, sybase, odbc, postgres, borland_ibase, db2, firebird, informix, ...)
Server	- database server name
Database	- database name
Username	- database user name
Password	- database user's password
SQL	- sql string, which should be executed (can contain context variables)
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.

Output:

If SQL command fails or other error will occur during connection to database, returns FALSE, otherwise returns TRUE.

Will return FALSE also in case, if select statement will not return next row from record set.

<Action SaveDbRow>

Save context values to database table.

Attributes:

Name	- defines action name
DBType	- database type – tested with: mysql, mssql, oracle, available are also other types of databases (sqlite, sybase, odbc, postgres, borland_ibase, db2, firebird, informix, ...)
Server	- database server name
Database	- database name
Username	- database user name
Password	- database user's password
Optional	- evaluates section always to TRUE (also in case any subelement will return FALSE)
Negate	- negates result of section evaluation. If Optional is set, Negate is ignored.
TableName	- defines name of table, where should be inserted values
Primary	- defines primary key of table (column names separated by comma)
ColumnDef	- table column definition in format: ColumnDef column_name, \$variable_name where \$variable_name will defines context variable, which values will be inserted to column.
Mode	- defines mode in which will be row saved to database, possible values: insert – try just insert row, update – try just update values, update_insert – if update of values will fail, try insert of values

Output:

If SQL command fails or other error will occur during connection to database, returns FALSE, otherwise returns TRUE.

<Action SaveCSV>

Save context values to csv file.

Attributes:

Name	- defines action name
FileName	- filename of csv file, where will be written values
Separator	- separator of values (default is ,)
Delimiter	- separator of rows (default is newline)
Column	- column name in format: variablename,column title (example: Column \$var1,Variable1 title)

Output:

If values will be written to file TRUE, else in case of any error FALSE.

<Action Php>

Execute php code loaded from file or specified directly in tag.

Attributes:

Name	- defines action name
FileName	- defines name of file, in which is stored php code, which should be executed
TemplateText	- defines text of php code, which should be executed

Example:

```
#Set context variable named $url (load value from $_REQUEST - form input field search)
<Action Php>
  TemplateText $context->setVariable('$url', 1
    "http://www.prweb.com/cgi-bin/search/search.pl?Terms=" . $_REQUEST['search'] . 1
    "&Match=0&Realm=prweb_inject&submit=Search");
</Action>
```

Output:

Returns TRUE if php code was executed successfully.

Returns FALSE, if occurs problem during evaluation of given php code (e.g. parse error).

<Logger>

Unit Miner during evaluation of each script generates different types of messages. For logging these messages and manipulating with these messages are used Loggers. Messages logged by this tag will be printed to standard output in plain text format suitable for text console.

Attributes:

Name	– defines logger name.
Level	– defines level of logging. If message level is equal or less then defined logger level, message is stored by current logger. Otherwise message is ignored by current logger. Possible logger levels are: <i>Error, Warning, Notice, and Debug</i>
Global	– defines global logger

<Logger Browser>

Tag is inherited from tag <Logger>. Messages logged by this tag will be printed to standard output in form suitable for browser.

Attributes:

All attributes inherited from tag <Logger>

Output:

Messages formatted in html style.

<Logger File>

Messages logged by this tag will be stored to output file in plain text format.

Attributes:

All attributes inherited from tag <Logger> plus:

FileName	– defines output file name. If file exist, messages will be added to existing file at the end.
-----------------	--

Output:

Messages formatted in plain text style stored in specified file.

<Logger Mail>

Messages logged by this tag will be stored to mail in plain text format and sent to specified recipients.

Attributes:

All attributes inherited from tag <Logger> plus:

From	– defines email address from which was sent email
To	– defines email recipients
Subject	– defines subject of email

Output:

Messages formatted in plain text style stored in email will be sent to specified recipient.

How to extend Unit Miner with new tags

In this chapter we will describe how you can extend wrapper script with new customer specific tags. Tag extensions are useful, when you need during evaluation execute special steps, which can't be written in wrapper script with set of tags, which are supported by default.

Let's say, that we have following problem:

We need periodically (e.g. each 15 minutes) check if all employees of our partner companies have access to our support application.

Each partner company have own structure of user list published on internet, so we will need to write for each partner web site new wrapper script (mostly it will be copy and paste work).

Our support application grants to users access depending on LDAP directory.

Our goal will be to get clean data from partner website (e.g.: username, first name, last name) and check if user exists in LDAP directory. If it's not inside, we will register new user to our LDAP directory.

Partner1 can have following structure of data (HTML code):

```
<html>
<body>
  <table>
    <tr>
      <th>Username</th>
      <th>First name</th>
      <th>Last name</th>
    </tr>
    <tr>
      <td>smalkovic</td>
      <td>Stanley</td>
      <td>Malkovic</td>
    </tr>
    <tr>
      <td>mmusterman</td>
      <td>Max</td>
      <td>Musterman</td>
    </tr>
    <tr>
      <td>sloeb</td>
      <td>Sebastian</td>
      <td>Loeb</td>
    </tr>
  </table>
</body>
</html>
```

To match data of Partner1, we will need to write following wrapper script:

```
<Section>
  Name Partner1

  # define variable $main_url and assign it value
  Define $main_url http://my_partner1

  # load content
  <Action ContentURL>
    URL {$main_url}/employee.html
    #remove all new line characters from content
    RemoveNewLine
  </Action>

  #Match beginning of users table
  <Pattern>
    RegExp <body><table>
  </Pattern>

  #iterate for each matched person
  <Section While>
    Name persons
    # match person
    <Pattern>
      RegExp <td>{$username}</td>\
            <td>{$first_name}</td>\
            <td>{$last_name}</td>
    </Pattern>

    #display matched variables to default output for testing reasons
    <Action Print>
      Text {$username}, {$first_name}, {$last_name}\n
    </Action>
  </Section>
</Section>
Main Partner1
```

Wrapper script above will match the data and each matched person will print to standard output. Result of script will be:

```
smalkovic, Stanley, Malkovic
mmusterman, Max, Musterman
sloeb, Sebastian, Loeb
```

We know, that script does the first step well and data we receive in clean form.

Now we have to add new action with name *CheckLDAP*, which will use matched data.

Let's say, that our project has following structure:

```
Adodb          - directory comes with Unit Miner
Wrapper        - directory comes with Unit Miner
SupportApp     - our support application
```

We need create new directory structure inside our support application directory:

```
SupportApp
```

```

    Wrapper
        Action

```

Inside this directory structure we have to create file: SupportApp/Wrapper/Action/CheckLDAP.class.php

Content of file can be following:

```

<?php
QUnit_Global::includeClass('Wrapper_Action');

class SupportApp_Wrapper_Action_CheckLDAP extends Wrapper_Action {

    function isUserInLDAP(&$context) {
        $oLDAP = new MyLDAPConnection();
        $oLDAP->username = $context->getVariable('$username');
        $oLDAP->firstname = $context->getVariable('$first_name');
        $oLDAP->lastname = $context->getVariable('$last_name');
        return $oLDAP->isUserRegistered()
    }

    function createLDAPUser(&$context) {
        $oLDAP = new MyLDAPConnection();
        $oLDAP->username = $context->getVariable('$username');
        $oLDAP->firstname = $context->getVariable('$first_name');
        $oLDAP->lastname = $context->getVariable('$last_name');
        return $oLDAP->registerNewUser()
    }

    function _execute(&$state, &$context) {
        if (!$this->isUserInLDAP($context)) {
            $this->createLDAPUser($context);
        }
        return true;
    }
}
?>

```

Now we can use our new php class as new tag in our script in following way:

```

<SupportApp:Action CheckLDAP>
<Action>

```

So finally our wrapper script will looks like:

```

<Section>
    Name Partner1

    # define variable $main_url and assign it value
    Define $main_url http://my_partner1

    # load content
    <Action ContentURL>
        URL {$main_url}/employee.html
        #remove all new line characters from content
        RemoveNewLine
    </Action>

```

```
#Match beginning of users table
<Pattern>
  RegExp <tr><th>*</th><th>*</th><th>*</th></tr>
</Pattern>

#iterate for each matched person
<Section While>
  Name persons
  # match person
  <Pattern>
    RegExp <tr><td>{$username}</td>\
          <td>{$first_name}</td>\
          <td>{$last_name}</td></tr>
  </Pattern>

  #check if user exists in our LDAP directory
  #if it's new user, create new user in LDAP directory
  <SupportApp:Action CheckLDAP>
  <Action>

  </Section>
</Section>

Main Partner1
```

Frequently asked questions

Can we integrate Unit Miner into our php application?

Yes. Unit Miner offers interface for communication with other applications inside wrapper script.

I need to download frequently price list from our contractors and import it to our database. Can Unit Miner automate this task?

Yes. Unit Miner was developed also for this type of tasks.

Can members of your support team help us with implementation of Unit Miner into our system?

Yes. Please contact our support team and we will help you to integrate Unit Miner with any system based on PHP.

After successful installation I'm still getting error: "Internal Server Error, this is an error with your script, check your error log for more information." What can be the problem?

Your version of Zend Optimizer is not supported. Unit Miner requires minimum version of Zend Optimizer 2.5.5.

After successful installation I'm getting error: "Fatal error: Unable to read XXX bytes in /your_path_to_unitminer/Wrapper/Lexer/StartTag.class.php on line 0". What did I wrong?

The problem is often cause by corrupted binary files. Files of Unit Miner are encoded by Zend Encoder into binary form. If you transmit file with ftp client and you switched client to ASCII mode, files will be corrupted. Therefore will be solution to transmit files in BINARY mode.

Is possible to retrieve also millions of pages from source site with Unit Miner ?

Yes, it's possible. But with single process version of Unit Miner it will take too much time to finish this task. For large sites we recommend to use **Unit Miner Distributed Server** version, which is capable to execute multiple scripts in same time and schedule evaluation to more processes and more servers, which work together on one retrieving task. Distributed Server version removes all limitations on speed of Unit Miner and only limitation of this process will become just speed of your internet connection.

System requirements

Operating system:

Operating system independent.

Tested with: Windows (NT, 2000, XP), UNIX, Linux

Should work on all systems, which are supported by PHP

PHP:

Version 4.3.9 and upper

CURL extension (standard PHP extension installed together with PHP and enabled in php.ini)

ZEND optimizer 2.5.5 or above (Unit Miner source files are encoded and optimized for best performance)

Database:

Database is required only in case data have to be stored to database by your script or by Unit Miner Distributed Server (for Distributed Server release is required MySql).

Supported are all types of database, which are supported in adodb, but mainly were tested:

MySQL, MSSQL, Oracle

Web server:

Required only for Unit Miner Distributed Server. There is no restriction on version or type of web server. Only requirement is, that it should be correctly installed with php.

Getting Help

For our customers is prepared our support team, which will answer kindly all your questions or offer solutions for your business needs. Our support team can help to our customers with implementation of Unit Miner to customer specific systems to reach the best performance and usability in your system environment. Support team has great experiences with all kinds of problems and we will offer to you ready solutions.

Please contact us:

Email: support@qualityunit.com

Unit Miner homepage: <http://www.unitminer.com>

Company homepage: <http://www.qualityunit.com>