**Error: Macro TranslatedPages(None) failed**

```
expected string or buffer
```

### FAQ Contents


1. General Questions
2. Installing YAM
3. Using YAM
4. ARexx Scripting
5. Debugging


# Debugging?


1. Basics about YAM's debugging facilities


## Basics about YAM's debugging facilities

This article discusses how you can get more detailed information from YAM in case you run into a problem which might point out being an actual bug in YAM or any other respective component used by YAM. For the development of YAM, but also for identifying the root of a problem, certain so-called debugging statements are included in the source code of YAM. Part of these statements are normally only available in so-called debug versions of YAM. But some statements are even present in the normal final release versions.

So in case you have been instructed by a YAM developer to run a debug version or you want to see if there might be an issue you would like to investigate a bit deeper, the following instructions might help you. Please note, however, that most of the things discussed here are hardly of any help for novice users. So please proceed with debugging only in case you have been instructed by a YAM developer to do so.

IMPORTANT NOTE:

Debug output is meant to help us to find bugs in YAM and may contain private data, like passwords, user names, etc. Please make sure to strip such data from the catched output **before** making it public in any form (e.g. in the bug tracker). We take absolutely **no responsibility** for possibly hacked mail accounts due to accidently published private information. So please be aware of that when dealing with debug output.

Different debugging facilities in YAM:

YAM provides two different debugging facilities:

1. debugging and output of all text based communication between YAM and the server you are communicating to (POP3, SMTP, etc.).
2. a full blown debugging log containing lots of internal stuff

The first one is available in every YAM executable (even the release version) and is enabled by either adding the `DEBUG` ToolType to YAM's icon or by running YAM from a Shell via `"YAM DEBUG"`. The latter method should be the preferred one, because the output can be redirected to a file (`"YAM DEBUG >debug.log"`) then, which you can edit afterwards.

The second facility is not contained in the normal release version of YAM, but in a special debug executable. This executable is also available on YAM's download page. Please make sure to catch the right debug archive for you platform and install the "YAM.debug" executable beside your normal "YAM" executable.

To make use of this special debug version, you also need a way to catch the serial output that this debug version sends its debugging information to. This can either be achieved by using a null-modem connection to a second computer (connected to your Amiga via a serial nullmodem cable) or via the Sashimi application available on AmiNet (for OS3/68k, for Morphos/PPC or for AmigaOS4/PPC). Sashimi is an application to cath all output sent to the serial line of your computer and as such allows you to catch the debugging information sent by YAM. In case you are not using Sashimi to capture the debug output, it will be send over the serial line to the remote computer and must be logged there with an appropriate terminal program (e.g. HyperTerminal or TeraTerm Pro under Windows or miniterm under Unix).

Using Sashimi:

If you are using Sashimi to log the debugging information locally, please run Sashimi before running YAM like this:

```
run <>nil: sashimi on noprompt >debug.log
```

Sashimi offers some more options to influence its behaviour. Please refer to Sashimi's documentation for further information.

Other required third-party debug tools:

Depending on the platform you are running YAM you should have the following tools installed and running before you start running a debug session of YAM:

AmigaOS3 or MorphOS:

- Enforcer or CyberGuard - to catch illegal memory accesses.
- SegTracker - to enable Enforcer/CyberGuard to show the hunk and source code information of the memory fault.

AmigaOS4:

- MemGuard - to enable OS4 to track for memory leaks and such

All these tools are available mostly on AmiNet or from other public places . Please read their documentation carefully. Usually no special options need to be specified to get a usable output in case of a crash. The only important point is to start those tools before running YAM.

How to define which debugging information to get:

Once you are able to catch the serial output that is sent by the debug version of YAM, you need to tell YAM what kind of debug information you want to have. This is done by using an environment variable (ENV variable) called "yamdebug". You just need to set this variable to a string which contains certain command which YAM will interpret as soon as it will be started (see below).

YAM's internal debugging facility is split into two parts: debug classes and debug flags. Debug flags are normally used to define a certain debug output to a function or area of the application (e.g. information about arexx, the startup, tcp/ip related, etc.). Debug classes, however, are used to define that the debug output/information is of a certain nature (e.g. a warning, an error or such). So most of the time you will probably want to set debug flags and only define a whole debug class to be shown or hided in case you receive too much or too less information.

All currently available debug flags are:

- always - print out always, not specifically related

- startup - startup/shutdown stuff
- timerio - timer.device stuff
- config - all about configuration handling
- filter - mail filtering (including spam filtering)
- folder - folder management
- mail - stuff related to single mails
- mime - MIME stuff
- gui - GUI handling
- rexx - ARexx stuff
- net - network and TCP/IP stuff
- util - miscellaneous utility stuff
- import - mail import and export
- xpk - the XPK compression system
- image - image handling and caching
- update - YAM's automatic update feature
- html - HTML document handling
- spam - information about the spam filter engine
- uidl - UIDL handling ("avoid duplicate mails")
- hash - hash table stuff
- print - printing of mails
- theme - theme stuff
- thread - thread handling
- all - all of the above

By default only the always and startup flags are active. That means, in case you start the debug version of YAM without having the "yamdebug" ENV: variable setup it will only print minor information about the startup and shutdown of YAM. Several flags can be combined and must be separated by either a space, a comma or a semicolon. Flags can also be excluded (negated) by prepending a '!'.

The following debug classes are available:

- @ctrace - generate a trace of all functions being entered and left
- @report -print out report information
- @assert - print out assertions
- @timeval - print out timing measurements
- @debug - print out normal debugging messages
- @error - print out error messages
- @warning - print out warning messages
- @mtrack - print out memory tracking warnings
- @tags - print out tag lists
- @all - all of the above

By default the classes @report, @assert, @debug, @error, @warning and @mtrack are active.

Debug classes must be prepended by a '@' character as can be seen by the above list. Several classes can be combined and must be separated by either a space, a comma or a semicolon. Classes can also be excluded (negated) by prepending a '!'.

There are two special classes which influence the way the output is generated.

Adding "stdout" will output all messages to the Shell window YAM was started from (or to the automatically opened window when YAM was started from the Workbench). These messages can be redirected to a file.

Adding "file:path_to_file" will generate a file named "path_to_file" and write all debugging messages directly to this file.

Examples:

```
setenv yamdebug "net image !startup"
```

This will enable debugging of the network stuff and image handling, but anything related to the startup/shutdown process will be excluded.All debug messages will be transmitted via the serial line to a connected terminal. Optionally these can be captured with a tool like Sashimi.

```
setenv yamdebug all,@!all,@error,stdout
```

This will enable all debug flags, but only for error messages. All debug messages will be printed to the Shell window YAM was started from.

```
setenv yamdebug @ctrace,@error,@!assert,stdout
```

This will enable trace information and error messages, but no assertions

```
setenv yamdebug @all,file:ram:t/debuglog.txt
```

This will enable all debug classes. Beware: the debug log file will become HUGE!!! All debug messages will be directly written to the file "ram:t/debuglog.txt" instead of the serial line.

Debug classes and flags can even be combined on one line. That means, e.g. `setenv yamdebug @all,all` will enable all debug classes and flags and produce the most verbose debug log possible (really, really HUGE!!!)