Debug Messages

This page is still a work in progress, it is more a place we put up a bunch of info pulled from old internal documentation. It still needs to be organized into a more usable format, until then just bear with us.

Turn on Debug Display

These are instructions for using the **Show Debug Messages** addon to display debug messages.

1. In the admin at **Addons > Manage Addons** install and enable the **Show Debug Messages** addon. There are other addons that also allow doing things with debug messages, but this is the one that displays them to the page. You could also create a custom addon that handles debug messages a certain way suited for certain circumstances, for instance if you needed to e-mail the debug messages.
2. In the URL, put **index.php?debug=type+list** to turn on debug output for those filter types. This allows us to do step #1 on a live site and not worry about all users seeing the output. For the type list, enter all types you want to show, with + between each one. See the List of Common Debug Types, the most common one is to use **index.php?debug=stats**.
3. When you are done, be sure to disable the debug addon, as any debugging addon will add a small amount of overhead to each page load.

List of Common Debug Types

This is a list of the most commonly used or useful debug types.

- **all+stats** - *[SPECIAL]*: shows all debug output. There is not actually any debug messages that use "all" as the keyword (or at least that is not the intent).
- **stats** - Most common & useful type, it shows statistic geared messages. We've placed debug messages at common bottlenecks as well, since each debug message has the time it allows looking for where slow-downs occur. *[SPECIAL]*: This will also display a table of DB queries used in the software, along with execution time for each query.
- **sql** - show sql messages and errors.
- **cart** - show output from the geoCart class and from other areas related to the cart operations.
- **site_class** - show output from site class. (possibly not used much)
- **sendmail** - show output from sending e-mail.
- **index** - show output from index page. (possibly not used much)
- **cache** - messages relating to cache.

Debug Output in the Page

When the above is turned on the software will display lines of output beyond the bottom of each page loaded. To see it once debug display has been turned on you only need to scroll down the page to see it. There could be quite a lot of it so if you know the type of message you are looking for you can choose it from the above list to narrow the output down a bit. The output will look something like that

below though the color may change from page load to page load:

Within that output you'll notice:

1. This is the timestamp for that line. That is the time in microseconds from the beginning of that page load to that point in the script run time. This number will only grow as you go down the page until you get to the end of the output. This is a good way to see the full amount of time the software takes to build the page requested. If this says 1.5 seconds at the end of all output and it takes a total of 10 seconds for your page to load this stat should help you narrow down page load issues. This stat is also useful in finding out if there is a particular section of the page load is slow. You can scan these values as you go down the page and see if there are any points where there is a lot of time between one line or another. You should be able to narrow down the possible section of a slowdown.
2. This the memory used by the script up to that point in the page load. This value may go up AND down in processing. The debug output does produces a stat at the end where it provides the highest memory usage during processing of that request so you don't need to search the page for the highest memory usage.
3. This is the output displayed by coding. Whatever message the coder wanted to output is there and sometimes includes sql queries, variable values at given steps of processing,...etc anything that might help building or debugging a feature.
4. This is the file the output was generated from.
5. This is the line number that output was generated from. Note that this value may not be the EXACT line as files include files and code from different sources and line counts may not be accurate.

Note that the software doesn't "output debug" for EVERYTHING the software does. So if you see a jump in time there just may not be much debug output in the particular section of code. We try not to put output in the middle of a loop that will iterate hundreds of times producing hundreds of lines of mainly useless output.

Debug Message Syntax

If you want to edit the base php code and insert your own debug output for your own feature creation or debugging use the folllowing in creating those **debug triggers** :

Debug message : just a general message.

```
trigger_error('DEBUG TYPE_1 TYPE_2: Insert debug message here.');
```

Error message : if something is wrong, for instance an SQL query error.

```
trigger_error('ERROR TYPE_1 TYPE_2: Insert error message here.');
```

Explanation of parts:

DEBUG or ERROR: First word in message needs to be either DEBUG or ERROR, depending on if this is just a general message or if it is reporting something that has gone wrong.

Debug types: AKA debug "keywords": A list of "debug types" this message applied so, separated by spaces, all upper case. Must be A-Z and underscore "_" any other characters can cause the message to not be handled as a Geo debug message.

: After the list of debug types, is a colon ":" which signifies the start of the actual debug message.

Message: Everything after the first colon ":" is treated as the debug message.

Future debug types: If you want to create a special debug type (so that it can get its own color or whatever) all you have to do is start using a unique type, like YOUR_UNIQUE_TYPE, then in the URL (if using show debug messages addon) put debug=your_unique_type to see that message.

Troubleshooting

The thing about addons is that things are not initialized until that functionality is actually used. This can be a problem if you are trying to get debug messages to be logged on a page where no core events are ever triggered. Now consider that the **errorhandle** core event[1] is never "fired" until AFTER the addons are initialized, in order to prevent a chicken and egg type of scenario. Combine these 2 facts, and that means that any page load when there is no core events fired for the entire page load, none of the debug messages will ever get sent through the errorhandle core event.

If you think this might be happening, the "fix" is to force a core event to be triggered on every page load. To do so, in **app_top.common.php** near the bottom, add (or un-comment if in Geo version 4.1.*):

```
/**
 * If experiencing a problem where addons are not being called for error
 * handling, un-comment the following line to "force" a triggered event,
which
 * will force addons to be loaded, and thus allow error handle core events
 * to be called on page loads that never trigger an addon even otherwise.
 */
geoAddon::triggerUpdate('forceInit');
```

The most common places you might need this are when processing a "signal" from a payment gateway, in calls to either of the **transaction_process.php** or the **recurring_process.php** files. It could also happen in any other type of "minimal" page load such as AJAX calls and the like.

Design Ideas

The idea behind having addons doing error handling, is that we can easily create a new addon that (for instance) only displays SQL errors, or only displays debug messages, etc. Or the addon can do different things with the information, for instance one addon could display the debug messages on the page, another could log them to a file (Debug Logger Addon), and another could e-mail the messages. If there are no addons that use errorhandle, then the performance hit for using trigger_error is insignificant.

1)

The crux of any debugging addon that logs or displays debug messages created by the system.