Core Events

A core event is a way for an addon to interact with the base software to change or add something, without needing to make any changes to the base code at all.

How it Works

Core events use observer design pattern, also known as publisher/subscriber design pattern. The "observer" is be the **geoAddon** class, and the "subscriber" is each individual addon.

An addon will be **automatically subscribed** to a core event, if 2 things happen:

- in **info.php**, the core event's name is found in the addon's **$core_events** var.
- in the util.php file, the addon's util class has a matching method that is named the same as the core event name, prepended with "core_"[1].

In the base code, to *trigger a core event*, it will look something like this:

```
geoAddon::triggerDisplay('event_name',$vars, geoAddon::FILTER);
```

Breaking that call down:

- trigger**Display**: triggerDisplay means that what the addon returns is important. There is a sister function, **geoAddon::triggerUpdate()** which does not care what is returned, when that is used it is for *notification purposes only*, anything returned by the addon is ignored[2].
- **event_name**: This is the event name. If the event is for a specific thing, it may start with the class name and function name it is being called from, to make it easier to identify where the event is used.
- **$vars**: These are the variables that will be passed to the addon when calling the core event's function in util.
- **geoAddon::FILTER**: This is what *type* of core event this is, in this case it is a filter core event. If not specified, the default type is "string" which means all the stuff returned from each addon is concatenated together (separated by the optional 4th parameter). The geoAddon::RETURN_STRING type is the original type of core event, and is one of the reasons the function is named triggerDisplay.

When a core event is triggered (by making a call similar to above) in the base code, the addon system sees which addons are subscribed to that event, and *notifies* each one by calling the before mentioned method in the util class of that addon. Depending on the core event's type, it may or may not do something with what is returned by the addon.

Core Event Types based on Return Value

Below are the different core event types based on how the returned value is used.

Core event type	class constant used for trigger	Event name beginning[3]	Description
Notification	N/A [4]	notify_	This is a notification to let the addon know that something is happening, so that the addon can do something it may need to do at that time. This type is used when an addon might want to manipulate something more complex than a simple string. For instance the event *notify_display_page*, happens at the top of geoSite→display_page() in order to allow an addon to manipulate the view class template vars, or to manipulate the geoSite class's variables. It may also be used when it is desirable to do some type of logging.
Filter	geoAddon::FILTER	filter_	A string is passed to the addon as the only variable. The addon is expected to filter/manipulate the string, then return the filtered string (or return the string un-modified if the addon doesn't want to change it).
Overload	geoAddon::OVERLOAD	overload_ then the class and method name being overloaded	This allows an addon to take over a specific function and do things its own way. A overload event allows you to replace a function with your own, where a filter event would happen at the end of a function and allow you to modify what the function is about to return. If the addon returns the constant geoAddon::NO_FILTER then the calling function keeps going on its merry way. Otherwise, if anything else is returned, then the calling function will return that value, effectively skipping what would normally be done by that function. The addon is responsible for returning the same stuff the original function would return, and for doing any input checking that might be needed, in other words it is responsible for doing everything the original function would have done.
Return a String	geoAddon::RETURN_STRING or geoAddon::ARRAY_STRING	it varies	Addon is expected to return a string (that will usually be used to display on the page). In the case of geoAddon::RETURN_STRING: The results of each addon will be concatenated together, sometimes separated by some string that is passed as the optional 4th parameter to triggerDisplay(). In the case of geoAddon::ARRAY_STRING: the results of each addon are returned in an array, one array entry per addon.
Return an Array	geoAddon::ARRAY_ARRAY	it varies	The addon is expected to return an array, if it returns anything other than an array, the returned value is ignored.

Core event type	class constant used for trigger	Event name beginning[3)]	Description
Return boolean true	geoAddon::RETURN_TRUE	it varies	If the addon returns true, no other addons are processed, and boolean true is passed back. If the addon returns anything else, it will be ignored and the rest of the addons will be processed. If all addons are processed and none return true, the trigger call will return boolean false.
Return boolean false	geoAddon::RETURN_FALSE	it varies	Acts much like return true type, but opposite: if addon returns false, no other addons are processed and triggerDisplay returns false. If no addons return false, triggerDisplay returns true.
Return not-null value	geoAddon::NOT_NULL	it varies	Related to return true and return false types, but in this case if the addon returns anything besides strict null, it stops processing addons and returns that as the results of triggerDisplay. This is used mostly in authentication type events, where you can return true to allow, false to say not allowed, or null if indifferent.

Core Event Types based on What they Do

Most core events fall into one category or another based on what they are used for. Below is more details on each.

For more information on each individual core event, see the documentation in the example addon.

filter

The string to filter is passed in as $var to the addon. The addon is expected to return the filtered version of that string. If it wasn't obvious, this falls into the "Filter" return type. Events will be prepended with **filter_**.

A lot of the filter events are used to filter the results of a particular function, in those cases the name of the event is usually **filter_[CLASS NAME]_[FUNCTION NAME]**.

overload

Used to take over a particular task and skip the built-in functionality that would normally be done by the base code. Events will be prepended with **overload_**, the entire event name usually follows something like **overload_[class name]_[function name]**.

notify

This event type is to notify the addon of a certain *event*, to allow the addon to do things at a specific point. The value returned by the addon for this event type is ignored. Events will be prepended with **notify_**.

email

This is a special case core event, it doesn't fall into any particular category, it is its own category.

Right now, this event is used as the only way that e-mails are sent by the software, through the use of the Send Mail Direct addon. Without an addon that uses the email core event, no e-mails would be sent by the software.

Something to be aware of for future versions: We plan to move the main mailing functionality to a base geoMail class in the future.

auth

Authorization core events. Most commonly used to see if a user has authorization to edit or delete a listing that was not started by the user. See the "Return not null" type of core event above for how the returned value is treated. Events will be prepended with **auth_**.

session

These core events are to deal with the user session or session data. In all events so far, it is done in addition to, not instead of, the built in stuff. It works kind of like a filter, except that the return values are ignored. It is passed the session info, and that is it. Events will be prepended with **session_**.

These are what the Bridge addon uses to do it's thing for syncing up login/logout with other installations.

user

These are for stuff dealing with things happening to a user, like registering a new user, editing a user's info, etc. This is in addition to, not instead of. Events will be prepended with **user_**.

The applicable user info is passed in, and the return value is ignored.

errorhandle

This is a special case, this is the name of a core event that is it's own category. It is triggered every time an error is triggered in the system (even notifications).

In the software, we use trigger_error() as a debugging message tool, the message syntax will match:

```
[ERROR|DEBUG] [TAG1[ TAG2...]]: Message
```

It can have any number of "tags", all uppercase, all separated by spaces, and ended with :. This allows us to log or display all messages that relate to a particular thing, for instance the tag name "STATS" is used for common bottleneck locations and other areas related to helping with software optimization, to help us speed up the software.

There are 2 addons included with the software that make use of this, one displays them on the page based on what "type" is currently activated, the other logs them in a log file based on settings. Using these built in addons will be the subject of another page in the wiki. The point is, we already have one addon demonstrating how to display on the page, another demonstrating how to log it to a file. You could create your own addon that might e-mail the admin user any time some certain thing happened, for example.

app_bottom

This is another core event that is it's own category. "Bottom" in this case, refers to the end of the page load, where the script is wrapping things up, not the physical location somewhere on a page. This core event is used to allow you to do things right before the page is done, but after the page is already displayed. In other words: It's closing time, you don't have to go home but you can't stay here.

1)
This is to keep things nice and tidy in the util class, so that it is clear which functions are called by the addon system as core events, and which are utility functioned called from somewhere in the addon itself.
2)
although the addon can still affect things, for instance an addon might manipulate the template vars stored in the geoView class
3)
Core events will sometimes be pre-pended with something so that it is obvious what type of core event it is. It is not an absolute rule, just a *rule of thumb* that the event name will start with what is noted.
4)
No class constant is passed during trigger, because this is the one and only type of core event that is triggered using geoAddon::triggerUpdate() which does not have a variable for event type.