geoTransaction

This is the object used for a transaction in the order and invoice system. A transaction will be assigned to an invoice, that invoice will be assigned to an order that the invoice is "paying" for. There will be many transactions to one invoice. Since this is an object that represents a single transaction in the system, there can be **many instances** of it.

A transaction with a negative "amount" means it is a charge, that is what is "owed" from the "buyer" to the seller (which in most cases will be the "site"). A positive amount represents a payment from the "buyer" to the "seller" (seller and buyer are set for the invoice that the transaction is attached to)

Create a Transaction

If you are creating a new transaction, you would start off by creating a new geoTransaction object, then assigning settings for it. Here is an example:

```
$transaction = new geoTransaction;
$transaction->setSeller(0);//payment to the site
$transaction->setBuyer($userId);
$transaction->setAmount($amount);
//...
```

setGatewayTransaction($custom_id)

Can be used by gateways, to have an easily search able field[1)] that goes with an external transaction id used in the external payment gateway's system, to link up the Geo transaction with the ID provided by the gateway.

Common Pitfalls:

Max length is 255: If a gateway needs a longer ID length, store the first 255 chars using setGatewayTransaction() for faster and easier "lookup" of the transaction. But then store the full ID in the transaction's registry. See the code snippet further below for an example of how to code for it.

ID Cannot be numeric: If the ID passed into geoTransaction::getTransaction($id) is numeric, it only looks up transactions with that ID as the internal ID. So if the "gateway transaction ID" is numeric, the transaction will not be found when you try to use geoTransaction::getTransaction($id), unless you add a string to it to ensure it is not numeric. See the code snippet below for an example of how to code for this:

Code snippet to account for both Pitfalls

```
//creating a new transaction
$transaction = new geoTransaction;
```

```
//set all the transaction data
//...

//prepend some string to the gateway transaction ID,
//to make sure it does not eval to be numeric
//Note that what is added can be what you want as long as it does
//not evaluate to be numeric.
$tranId = 'gateway'.$tranId;

//Make sure the ID is not more than 255 chars
$shortId = substr($tranId,0,255);
//set the ID
$transaction->setGatewayTransaction($shortId);
if ($shortId != $tranId) {
    //The ID was indeed shortened, save the full version
    //to make sure we get the correct transaction in the future
    $transaction->set('full_transaction_id', $tranId);
}

// ... Later, when retrieving the transaction object based on $tranId which
is transaction ID in
// the payment gateway's system:
$tranId = 'gateway'.$tranId;//remember to re-add the text that was added
before when saving
$shortId = substr($tranId,0,255);
$transaction = geoTransaction::getTransaction($shortId);
//check to make sure we got the right transaction
if (!$transaction) die ('invalid transaction');//dieing is a little harsh,
but this is just an example...

if ($tranId != $shortId && $transaction->get('full_transaction_id') !=
$tranId) {
    //the length was shortened, and the full ID does not match, this is not
the transaction we want
    die('invalid transaction');//again, don't need to die, this is just for
dramatic effect.
}
//It gets this far, you have $transaction set to the original transaction
object!
```

Of course if you know the format of the "external ID" used by the gateway for the transaction, and know that the length will never be over 255 chars and will always have alpha chars in it, the above is **way overkill**.

1)

Since the field in the DB is a varchar, looking up a transaction by this field should be fast, by using geoTransaction::getTransaction($gateway_transaction_id).