BMO Unit Testing

Note: THIS IS UNIMPLEMENTED, AND THIS PAGE IS HERE FOR DISCUSSION.

To assist in development of Modules, BMO provides an extremely basic Unit Testing framework.

Your module is required to implement the 'doTests()' function, which should return (bool) true, or, throw an Exception if a fault is discovered.

These tests can be run by calling FreePBX->Tests->runAllTests() or FreePBX->Tests->runTests('modulename'), or, by 'amportal a runtests'.

Before the tests are run, a small test SQLite ***(??? mysql?? suggestions?)*** database is created and propagated with random, but valid, data. This raw PDO Object is handed as the only parameter to the doTests() function, if you wish to add/change settings.

All BMO Features will reference this test database, and any BMO file writes will transparently have /tmp/ prepended to them (eg, if you call BMO::WriteFile(array("testfile" => "testfile contents")), rather than creating /etc/asterisk/testfile, it will create /tmp/etc/asterisk/testfile and set the contents to "testfile contents")

When your doTests function is called, it is expected to run through a variety of tests and checks to ensure it is functioning correctly.

Unit Testing - Why?

The best thing about Unit Testing is that you can be **sure** that your code works as designed. If you're doing a bunch of complex code, and you need to change the return variable from one, you can either search through the code for everything that uses that return variable, or, you can run your tests and see what has changed. Often, changing one value can have unintended consequences far down the chain. By writing unit tests as you write your code, you're always going to be certain that everything is producing the output it's expected to, from the input you give it.

This is only a quick summary - there are many, MANY, documents on why and how on the 'net.

Unit Testing - How?

The easiest way to do basic Unit testing in BMO is to test the entire thing.

```
public static $dbDefaults = array ( "bestpony" => "Rainbow Dash" );

public function doTests($db) {
    $_REQUEST['bestpony'] = "Twilight Sparkle";
    $_REQUEST['display'] = "extensions";
    $_REQUEST['Submit'] = "submit";
    if ($this->getConfig("bestpony") != "Rainbow Dash") {
        throw new Exception("BMO Didn't load the default for Best Pony");
    }
    $this->doConfigPageInit($_REQUEST['display']);
    // Now, assuming your configPageInit captured 'bestpony' and set its
config var to that..
    if ($this->getConfig("bestpony") != "Twilight Sparkle") {
        throw new Exception("My doConfigPageInit didn't capture bestpony");
    }
    return true;
}
```

When you discover a bug, write a test that finds that bug:

```
public function addOne($var) {
    // Add one to $var
    return $var + 2;
}

public function doTests() {
    if ($this->addOne(1) != 2) {
        throw new Exception("1 + 1 didn't equal 2");
    }
    return true;
}
```

Once you've written that test, and fixed the bug **don't remove the test**. There's no need to. Since you've written the test, it's always there, and if you accidentally break addOne in the future, you will immediately pick it up when you run the tests.