ROBODoc System Tests

Generated with ROBODoc Version 4.99.28 (Jul 5 2006)

July 11, 2006

Contents

1 ROBODoc/ROBODoc System Tests

FUNCTION

A set of perl scripts that test ROBODoc functionallity. Each script contains one of more system tests. Each test starts ROBODoc with a specific input and then asserts that ROBODoc produces the correct output.

The tests use the Perl unittest framework, that is the modules:

- Test::More, See http://perldoc.perl.org/Test/More.html
- Test::File

There is also a custom module ROBOTestFrame (2) that contains a set of useful functions that are common to all tests.

You can run the tests with

```
prove
```

Tests go into files with the extension .t. You can run an individual set of tests with

```
prove <testfile>
```

A 'prove' should always result in a 100% score. No test should fail.

1.1 ROBODoc System Tests/Basics

FUNCTION

Tests basic ROBODoc funcionallity. These are all 'happy paths'.

1.1.1 Basics/Dummy Headers

FUNCTION

A dummy header to put into dummy source files.

SOURCE

```
my $source = <<'EOF';
/****f* Test/test
 * NAME
 *   Test
 ******
 */
EOF
```

1.1.2 Basics/One file per header

FUNCTION

Test –multidoc with –one_file_per_header for html output. We test this with one source file that contains three headers. These should result in three documentation files. To make it nasty we use some special header names.

SOURCE

```
#    A dummy header to put into dummy source files.
my $source = <<'EOF';
/****f* Test/Test
 * NAME
 *   Test foo bar
 ******
 */

/****f* Test/foo, bar
 * NAME
 *   Test foo bar
 ******
 */

/****f* Test/aa::awaw
 * NAME
 *   Test foo bar
 ******
 */

EOF

{
    add_source( "test.c", $source );
    my ( $out, $err ) = runrobo(qw(--src Src --doc Doc --one_file_per_header --multidoc --html));
    # expected results:
    is( $out, '', 'No ouput' );
    is( $err, '', '... and no error' );
    # There are three headers, so there should be three documentation files.
    file_exists_ok( 'Doc/test_cTest2FTest.html',       'Documentation for Test/Test' );
    file_exists_ok( 'Doc/test_cTest2Ffoo.html',        'Documentation for Test/foo' );
    file_exists_ok( 'Doc/test_cTest2Faa3A3Aawaw.html', 'Documentation for Test/aa::awaw' );
    # And a style sheet.
    file_exists_ok( "Doc/robodoc.css", 'and a stylesheet' );
    clean();
}
```

1.1.3 Basics/Option help

FUNCTION

Test the option –help.

SOURCE

```
{
    my ( $out, $err ) = runrobo('--help');
    like( $out, qr/ROBODoc Version/, '--help should print version number' );
    is( $err, '', '... and no error' );
}
```

1.1.4 Basics/Option multidoc

FUNCTION

Test –multidoc for html output format. (Multidoc for other modes does not make much sense). We create one source file with a simple header and create multidoc

SOURCE

```
{
    add_source( "test.c", $source );
    my ( $out, $err ) = runrobo(qw(--src Src --doc Doc --multidoc --html));
    # expected results:
    is( $out, '', 'No ouput' );
    is( $err, '', '... and no error' );
    file_exists_ok( "Doc/test_c.html", 'there should be documentation' );
    file_exists_ok( "Doc/robodoc.css", 'and a stylesheet' );

    clean();
}
```

1.1.5 Basics/Option version

FUNCTION

Test robodoc –version, this should report the current version.

SOURCE

```
{
    my ( $out, $err ) = runrobo('--version');
    like( $out, qr/\d+\.\d+.\d+/, '--version should print version number' );
    is( $err, '', '... and no error' );
}
```

1.1.6 Basics/Singledoc Outputmode with Different Formats

FUNCTION

Test singledoc mode for all supported output formats and see if it creates output in this format.

SOURCE

```
{
    add_source( "test.c", $source );

    my %output_modes = (
        '--html'  => 'html',
        '--rtf'   => 'rtf',
        '--test'  => 'xml',
        '--latex' => 'tex',
        '--dbxml' => 'xml',
    );

    foreach my $mode ( keys %output_modes ) {
        my $file_extension = $output_modes{$mode};

        my @arguments = qw(--src Src --doc testdoc --singledoc);
        push( @arguments, $mode );

        my ( $out, $err ) = runrobo(@arguments);
        is( $out, '', 'No ouput' );
        is( $err, '', '... and no error' );
        file_exists_ok( "testdoc.$file_extension",
            'there should be documentation' );
        if ( $mode eq "--html" ) {
            file_exists_ok( "testdoc.css", 'and a stylesheet' );
        }

        unlink("testdoc.$file_extension") if -e "testdoc.$file_extension";
        unlink('testdoc.css')             if -e 'testdoc.css';

    }

    clean();
}
```

1.2 ROBODoc System Tests/Handling Errors

FUNCTION

Test wether calling ROBODoc with wrong options or input leads to the correct error messages.

1.2.1 Handling Errors/Error on Duplicate Options

FUNCTION

ROBODoc should complain about options that are specifed more than once.

SOURCE

```
{
    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw( --src Src --doc Doc --multidoc --test --test --test) );
    print $out;
    like($out, qr/Usage/, 'Duplicate options should print usage' );
    print $err;
    like($err, qr/than\sonce/, 'and an error message'   );
    clean();
}
```

1.2.2 Handling Errors/Error on Mutual Excluding Options

FUNCTION

ROBODoc should complain about options that can not be used together.

SOURCE

```
{
    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Doc --multidoc --singledoc --html) );
    like($out, qr/Usage/, 'Mutual excluding options should print usage' );
    print $err;
    like($err, qr/together/, 'and an error message'   );
    clean();
}
```

1.2.3 Handling Errors/Error on Mutual Excluding Options II

FUNCTION

ROBODoc should complain about options that can not be used together.

SOURCE

```
{
    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Doc --multidoc --test --dbxml --html) );
    like($out, qr/Usage/, 'Mutual excluding options should print usage' );
    print $err;
    like($err, qr/together/, 'and an error message'   );
    clean();
}
```

1.2.4 Handling Errors/Error on Non-existing Option

FUNCTION

ROBODoc should complain about non-existing options.

SOURCE

```
{
    my ($out, $err) = runrobo( '--foobar' );
    like($out, qr/Usage/, 'Unknown option should print usage' );
    like($err, qr/Invalid/, 'and an error message'   );
}
```

1.2.5 Handling Errors/Error on Non-existing rc file

FUNCTION

When given a non-existing .rc file, ROBODc should at least report the name of the .rc file.

SOURCE

```
{
    my ($out, $err) = runrobo( '--rc foobar.rc' );
    like($err, qr/foobar/, 'should give an error message about foorbar.rc' );
}
```

1.2.6 Handling Errors/Header without end marker

FUNCTION

Test ROBODoc's response on a header without an end marker. ROBODoc should detect this.

SOURCE

```
{
    my $source = <<'EOF';
/****f* Test/test
 * NAME
 *   Test
 * FUNCTION
 *   Test2
 * SOURCE
 */

some source

and no end of the header  ....

EOF

    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Doc --multidoc --test) );
    is( $out, '', 'no output' );
    like( $err, qr/end\smarker/, 'error about end marker' );
    clean();
}
```

1.2.7 Handling Errors/Headers with duplicate names

FUNCTION

Test ROBODoc's response to a file with two headers that have the same name. This should be reported as an error.

SOURCE

```
{
    my $source = <<'EOF';
/****f* Test/test
 * NAME
 *   Test
 * FUNCTION
 *   Test2
 *****
 */

/****f* Test/test
 * NAME
 *   Test
 * FUNCTION
 *   Test2
 *****
 */

EOF

    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Doc --multidoc --test) );
    is( $out, '', 'no output' );
    like( $err, qr/already\sexists/, 'error duplicate header' );
    clean();
}
```

1.2.8 Handling Errors/Impossible output file

FUNCTION

When given a impossible output filename ROBODoc should at least report the filename.

SOURCE

```
{
    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Foo/Bar/document --singledoc --html) );
    like($err, qr/document/, 'should give an error message about document' );
}
```

1.2.9 Handling Errors/Non-existing css file

FUNCTION

When given a impossible css filename ROBODoc should at least report the filename.

SOURCE

```
{
    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Docs --multidoc --html --css Foo/Bar/cascade.css) );
    like($err, qr/cascade/, 'should give an error message about css file' );
}
```

1.3 ROBODoc System Tests/Header Test

FUNCTION

Tests the parsing of ROBODoc headers.

1.3.1 Header Test/Happy Path

FUNCTION

Happy Path Simple plain header. This definitely should work

SOURCE

```
{
    my $source = <<'EOF';
/****f* Test/test
 * NAME
 *   Test
 * FUNCTION
 *   Test2
 ******
 */
EOF

    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Doc --multidoc --test) );
    is( $out, '', 'no output' );
    is( $err, '', 'and no error' );
    clean();
}
```

1.3.2 Header Test/Multiple Names with Spaces

FUNCTION

Try several header with names that contain spaces. These should be accepted.

SOURCE

```
{
    my $source = <<'EOF';
/****f* Test Foo Bar/Name With Spaces, And Anotherone,
 *                   And One More, More
 * NAME
 *   Test
 * FUNCTION
 *   Test2
 ******
 */
EOF

    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Doc --multidoc --test) );
    is( $out, '', 'no output' );
    is( $err, '', 'and no error' );
    my $documentation = XMLin( 'Doc/test_c.xml' );
    my $header = $documentation->{'header'};
    is ( $header->{'name'}, 'Test Foo Bar/Name With Spaces', 'Header name' );
#    clean();
}
```

1.3.3 Header Test/Names with Spaces

FUNCTION

Try a header name with spaces and some '*' at the end. The '*' should be ignored.

SOURCE

```
{
    my $source = <<'EOF';
/****f* Test Foo Bar/Name With Spaces ****
 * NAME
 *   Test
 * FUNCTION
 *   Test2
 ******
 */
EOF

    add_source( "test.c", $source );
    my ($out, $err) = runrobo( qw(--src Src --doc Doc --multidoc --test) );
    is( $out, '', 'no output' );
    is( $err, '', 'and no error' );
    my $documentation = XMLin( 'Doc/test_c.xml' );
    my $header = $documentation->{'header'};
    is ( $header->{'name'}, 'Test Foo Bar/Name With Spaces', 'Header name' );
    clean();
}
```

1.4 ROBODoc System Tests/Wiki Formatting

FUNCTION

Tests that test the Wiki like formatting that ROBODoc supports.

1.4.1 Wiki Formatting/Wiki Basics

FUNCTION

Test a simple header: contains three lists, some paragraphs, and some source. All should be recognized.

SOURCE

```
{
    my $source = <<'EOF';
/****f* Test/Test
 * NAME
 *
 *   Implements serializers for the following
 *   files:
 *   - DZB_ACG - SAP accounting file record.
 *   - DZB_RRP - regularoty reporting file record.
 *   - DZB_MVT - Exchange Position File Record.
 *
 *
 *   Implements the following
 *   functions:
 *   - S99304_SERIALIZE_DZB_ACG
 *   - S99304_SERIALIZE_DZB_ACG_TBL
 *   - S99304_SERIALIZE_DZB_MVT
 *   and the functions:
 *   - S99304_SERIALIZE_DZB_MVT_TBL
 *   - S99304_SERIALIZE_DZB_RRP
 *   - S99304_SERIALIZE_DZB_RRP_TBL
 * SOURCE
 */
    test()

 /******/

EOF

    add_source( "test.c", $source );
    my ( $out, $err ) = runrobo(qw(--src Src --doc Doc --nopre --multidoc --test));
    # expected results:
    is( $out, '', 'No ouput' );
    is( $err, '', '... and no error' );

    my $documentation = XMLin( 'Doc/test_c.xml' );
    my $header = $documentation->{'header'};
    is ( $header->{'name'}, 'Test/Test', 'Header is named Test/Test' );
    my $items = $header->{'item'};
    ok ( exists( $items->{'NAME'} ),   'header has an item NAME' );
    ok ( exists( $items->{'SOURCE'} ), 'header has an item SOURCE' );
    my $body = $items->{'NAME'}->{'item_body'};

    # There are paragraphs.
    ok ( exists( $body->{'para'} ),    'item has paragraphs' );
```

```
    # There are three lists.
    is ( scalar( @{ $body->{'list'} } ), 3, 'item has two lists' );
    clean();
}
```

2 ROBODoc/ROBOTestFrame

FUNCTION

A Perl module with a set of handy functions to create test scripts.

These function are:

- runrobo (2.5)
- add_source (2.1)
- clean (2.2)

2.1 ROBOTestFrame/add_source

FUNCTION

Add a single source file somewhere in Src/

SOURCE

```
sub add_source
{
    my $filepath = shift;
    my $source   = shift;

    my $dir_name = "$src" . dirname( $filepath );

    if ( ! -e "$dir_name" ) {
        mkpath $dir_name or die "can't create $dir_name";
    }

    my $full_filepath = "$src/$filepath";
    my $file = IO::File->new(">$full_filepath") or die "Can't open $full_filepath";
    print $file $source;
    $file->close();
}
```

2.2 ROBOTestFrame/clean

FUNCTION

Clean source and documentation directories.

SOURCE

```
sub clean
{
    if ( -e $src ) {
        rmtree( $src ) or die;
    }
    if ( -e $doc ) {
        rmtree( $doc ) or die;
    }
}
```

2.3 ROBOTestFrame/doc

FUNCTION

Name of the documentation directorie use to test ROBODoc.

2.4 ROBOTestFrame/robo

FUNCTION

Location of the ROBODoc executable.

SOURCE

```
my $robo = "../robodoc.exe";
```

2.5 ROBOTestFrame/runrobo

FUNCTION

Run robodoc with the given set of arguments and capture all output to stdout en stderr.

SOURCE

```
sub runrobo
{
    run( [ $robo, @_ ], \my( $in, $out, $err ) );
    return ($out, $err);
}
```

2.6 ROBOTestFrame/src

FUNCTION

Name of the source directory used to test ROBODoc.

SOURCE

```
my $src  = "Src";
```

Index