Ensembl website refactoring

James Smith

Js5@sanger.ac.uk

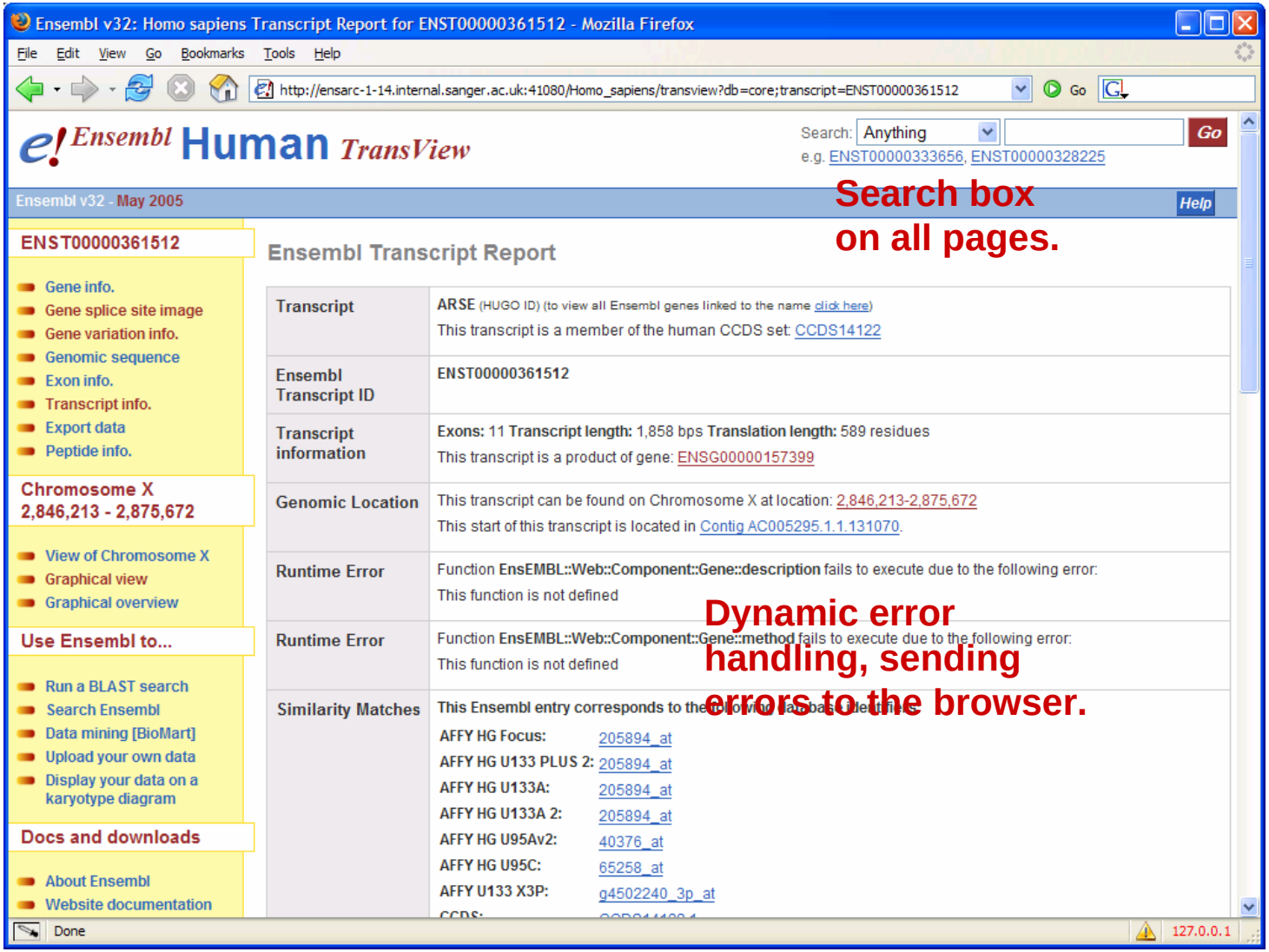
Ensembl Web Team Project Leader

New design

Information about release

Context sensitive links

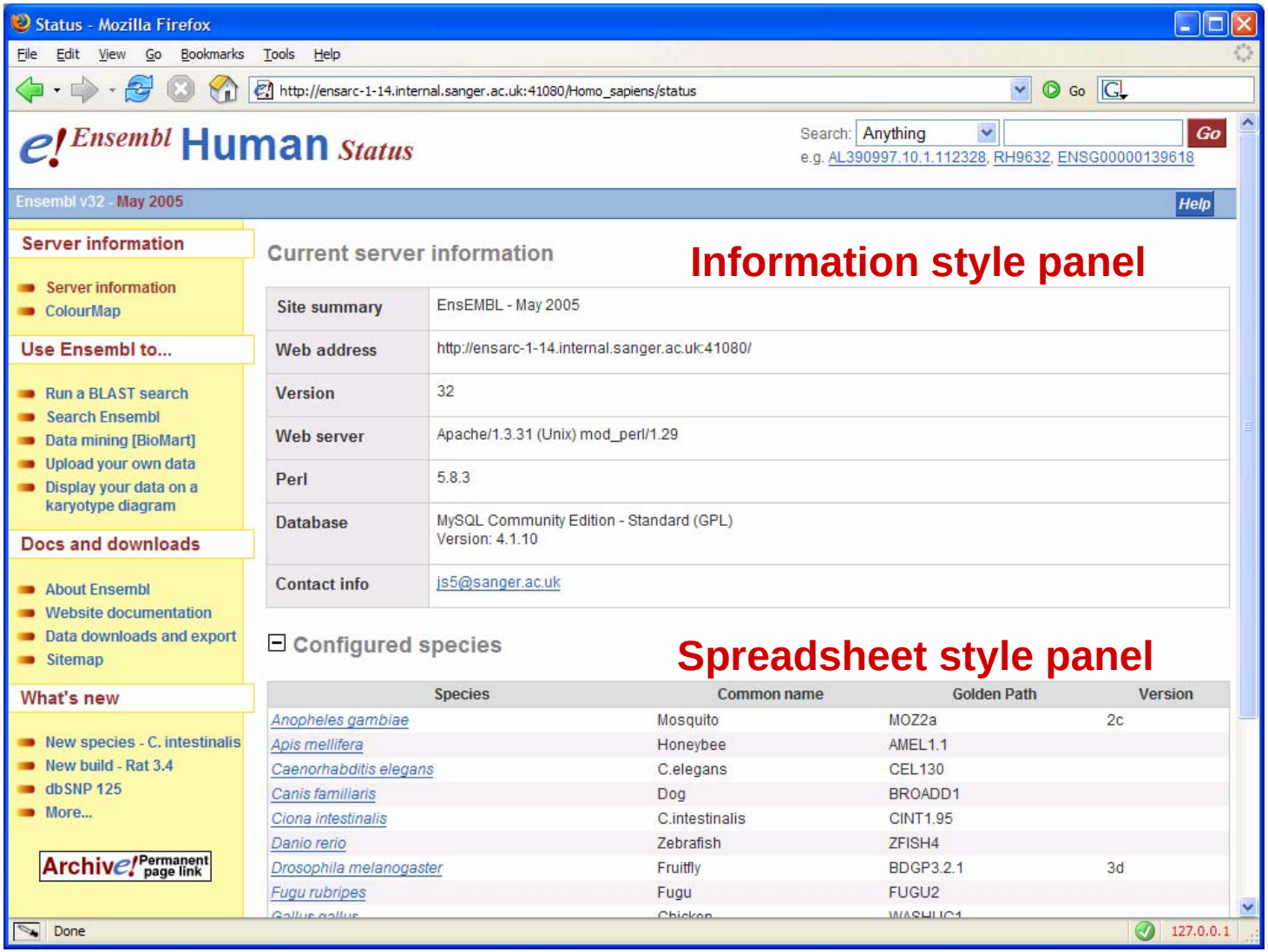
How-to style links



Search box on all pages.

Dynamic error handling, sending errors to the browser.

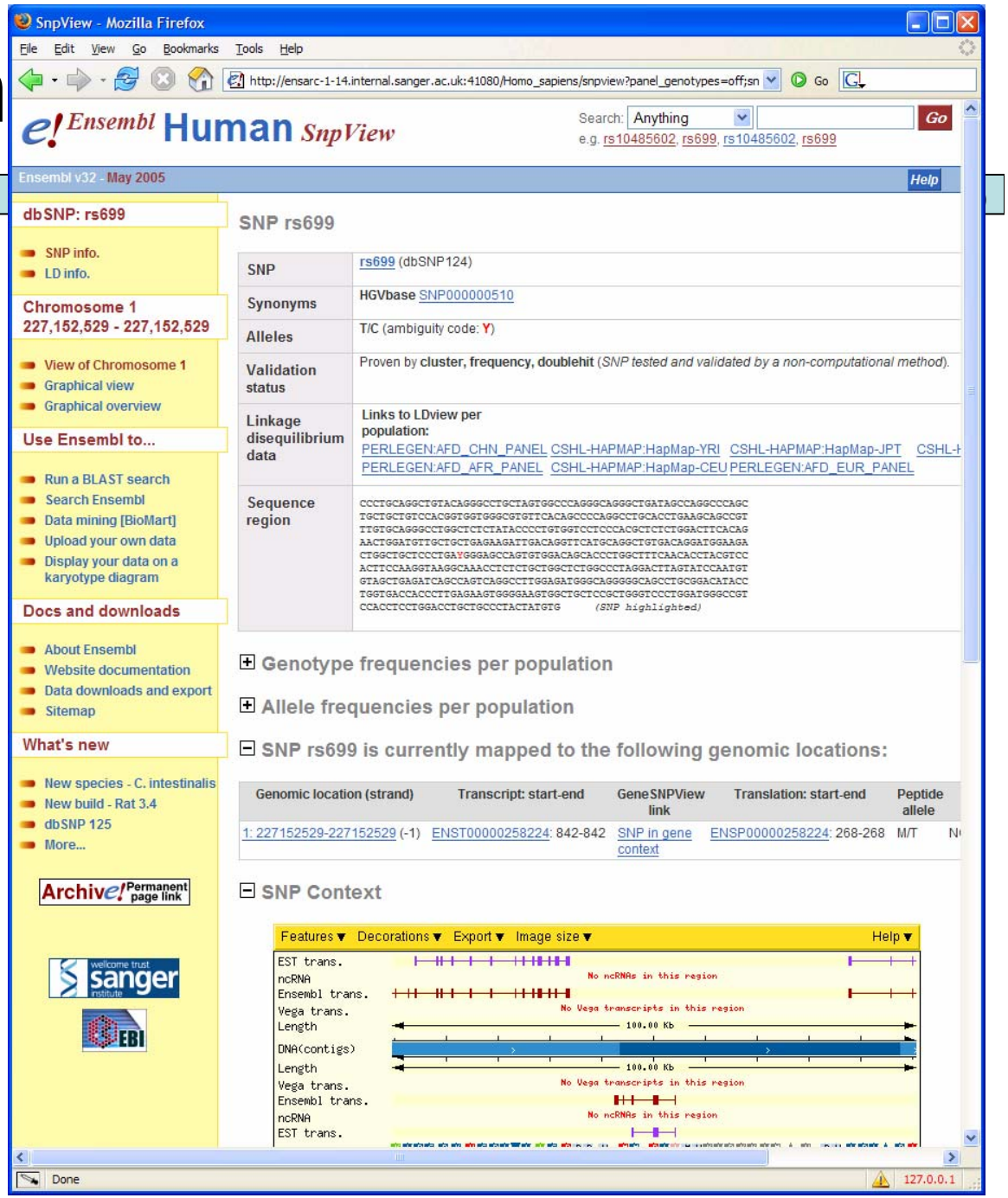
New design


Status - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://ensarc-1-14.internal.sanger.ac.uk:41080/Homo_sapiens/status
Ensembl Human Status
Search: Anything Go
e.g. AL390997.10.1.112328, RH9632, ENSG00000139618
Ensembl v32 - May 2005
Help
Server information
Server information
ColourMap
Use Ensembl to...
Run a BLAST search
Search Ensembl
Data mining [BioMart]
Upload your own data
Display your data on a karyotype diagram
Docs and downloads
About Ensembl
Website documentation
Data downloads and export
Sitemap
What's new
New species - C. intestinalis
New build - Rat 3.4
dbSNP 125
More...
Archive! Permanent page link
Current server information
Information style panel
Site summary EnsEMBL - May 2005
Web address http://ensarc-1-14.internal.sanger.ac.uk:41080/
Version 32
Web server Apache/1.3.31 (Unix) mod_perl/1.29
Perl 5.8.3
Database MySQL Community Edition - Standard (GPL) Version: 4.1.10
Contact info js5@sanger.ac.uk
Configured species
Spreadsheet style panel
Species Common name Golden Path Version
Anopheles gambiae Mosquito MOZ2a 2c
Apis mellifera Honeybee AMEL1.1
Caenorhabditis elegans C.elegans CEL130
Canis familiaris Dog BROADD1
Ciona intestinalis C.intestinalis CINT1.95
Danio rerio Zebrafish ZFISH4
Drosophila melanogaster Fruitfly BDGP3.2.1 3d
Fugu rubripes Fugu FUGU2
Done
127.0.0.1

New design

Ensembl website refactoring

Print previews have been styled so that the output is rendered without a lot of the decoration and navigation of the webpage.



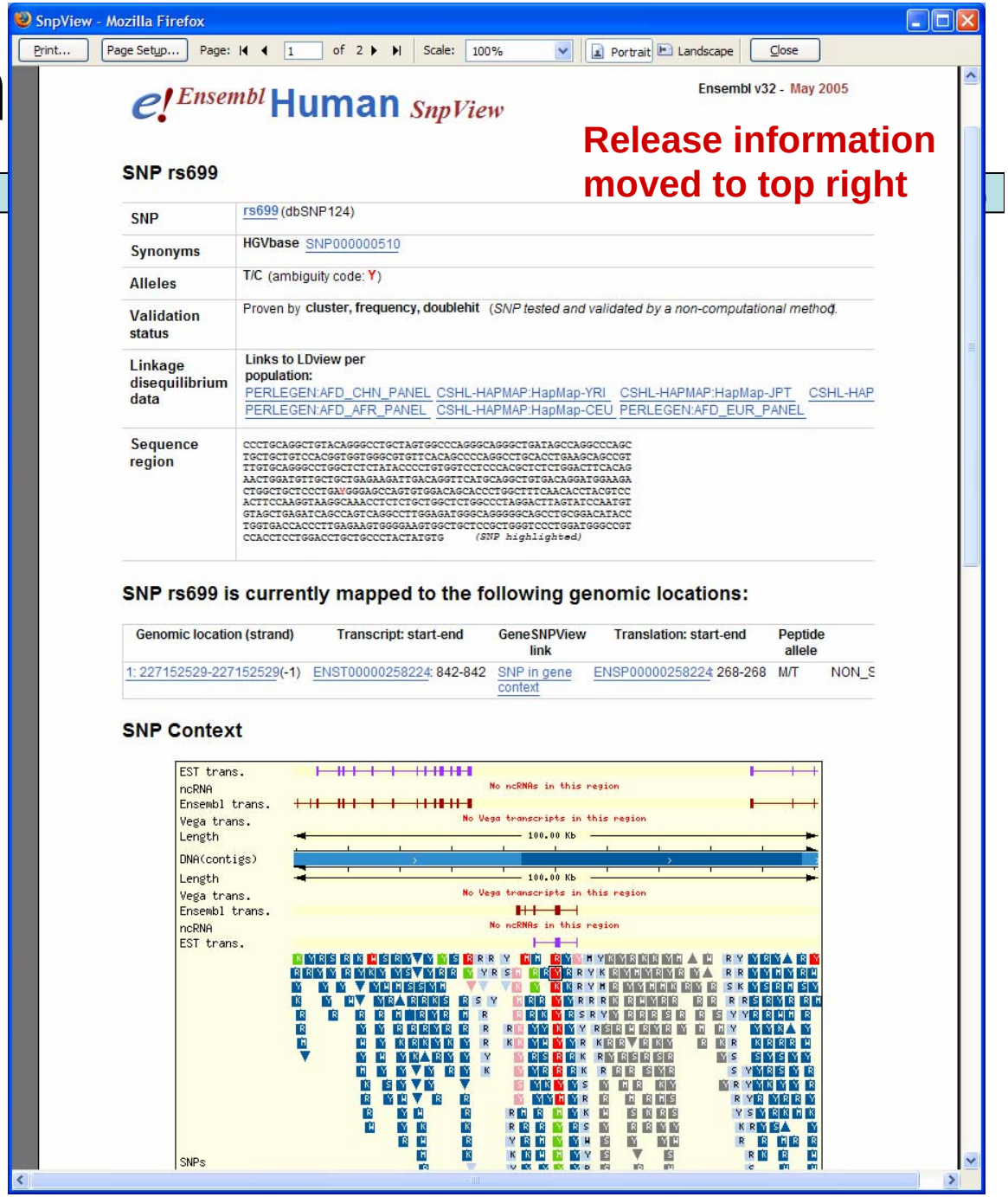
New design

Ensembl website refactoring

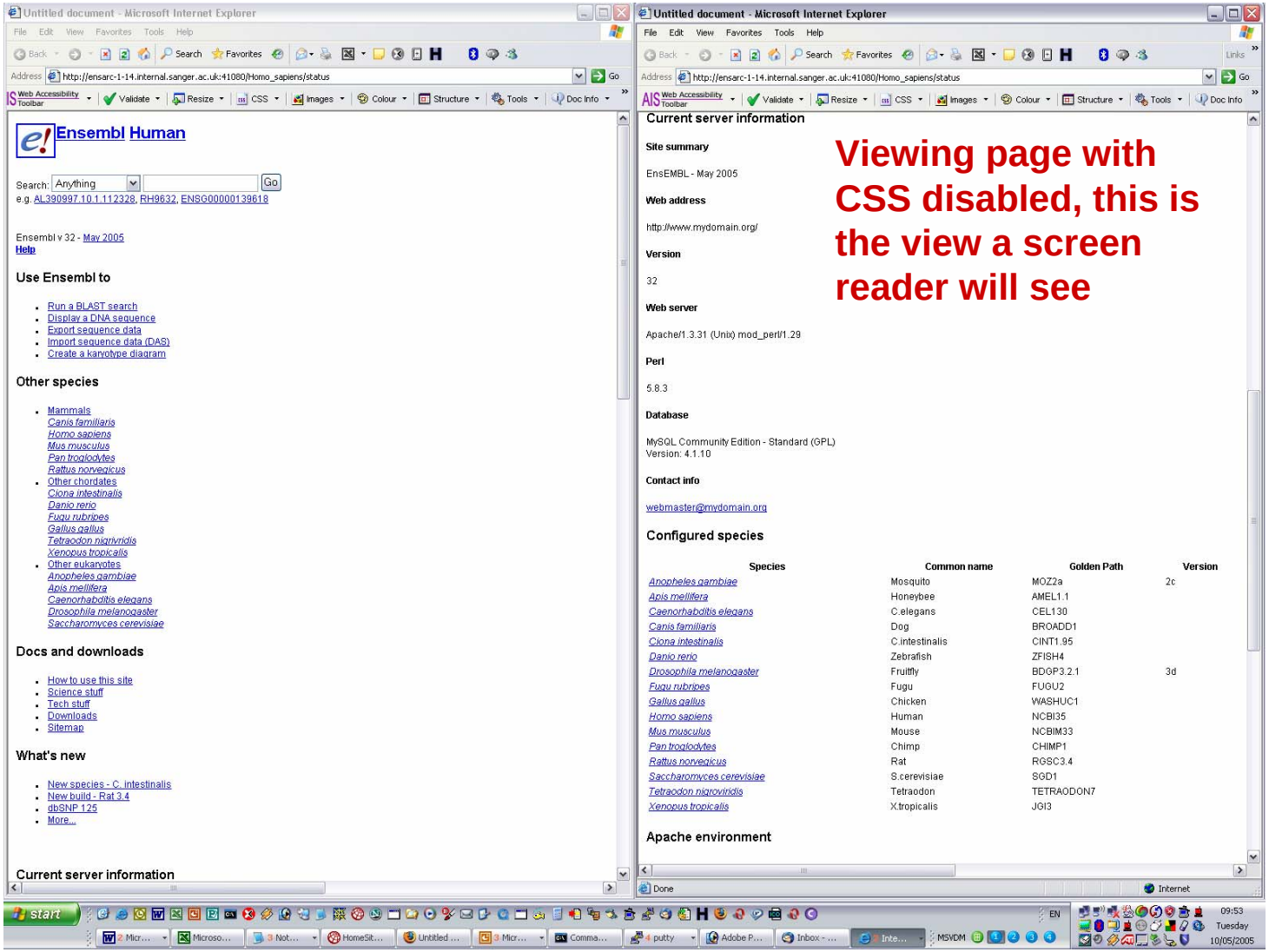
No navigation menu, help link, search.

Collapsed panels not displayed in print mode.

+/- boxes and navigation tools hidden



New design – no CSS??



Plugins

Currently if we wish to have two variants of the website, then we need to have two copies of the whole code – to allow for minor tweaks in a large number of modules. This means:

- Whenever the “core” Ensembl webcode is updated, each variant has to work their modifications back into the main branch – risking missing changes and wasting development time.

In the redevelopment we are developing a plugin system, where:

- Each “variant” has it’s own set of modules – a Plugin – outside the “core” webcode, which add to, replace or modify the action of the “core” modules.

Plugins – changes requried

Plugins will allow us to:

- add additional scripts;
- add/modify panels within scripts.

Changes to the code will be extensive (most will need re-writing) but relatively simple:

- E::W::Renderer::*{ObjectType}*::HTML objects split into E::W::Component::*{ObjectType}* (for rendering) and E::W::Configuration::*{ObjectType}* (for configuration)
- E::W::DataFactory::*{ObjectType}*Factory modules will be replaced by E::W::Factory::*{ObjectType}*
- E::W::Data::*{ObjectType}* modules will be replaced by E::W::Object::*{ObjectType}* modules.

Extending the Ensembl Website – Part 1: Writing a new script.

James Smith
Js5@sanger.ac.uk

Ensembl Web Team Project Leader

Writing a new script

To add a new script we need to:

- Write a new script in perl/default

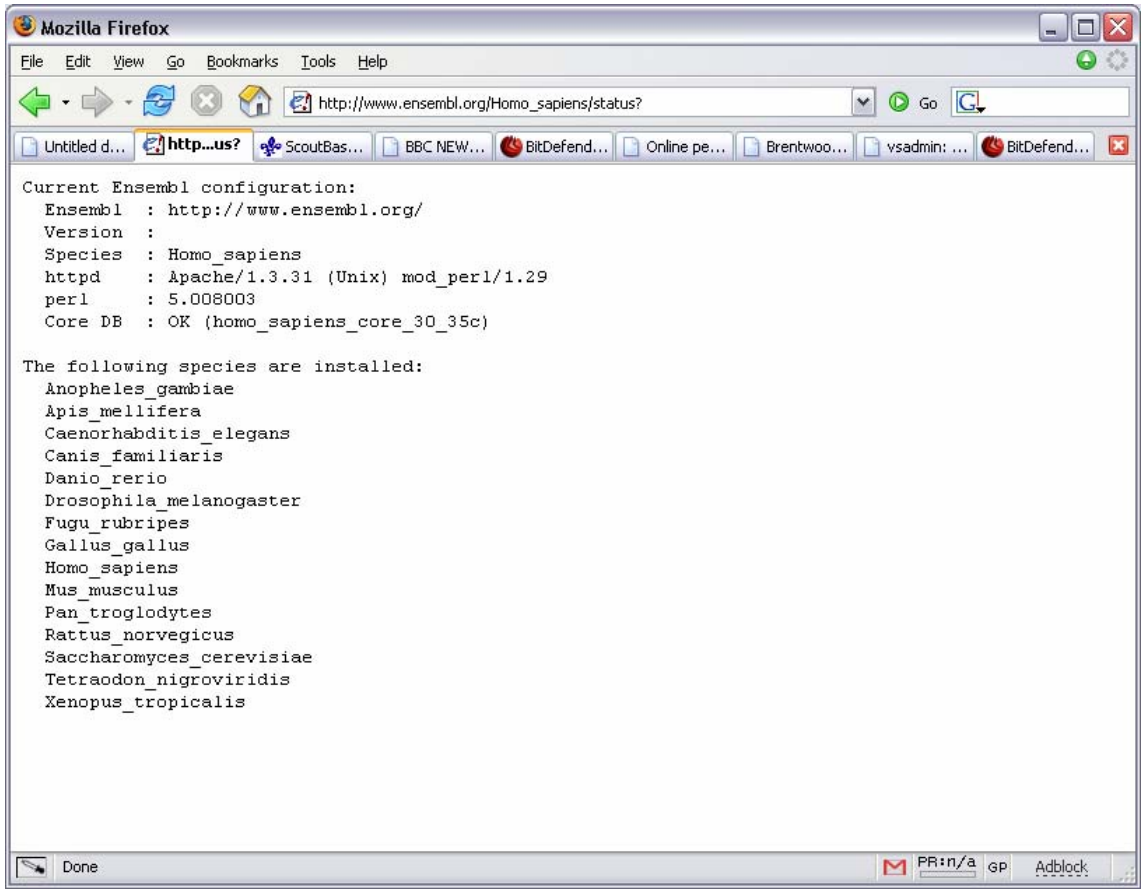
And one or more of the following:

- new ScriptConfig module;
- new Factory module and/or functions;
- new Object module and/or functions;
- new Configuration module and/or functions;
- new Component module and/or functions

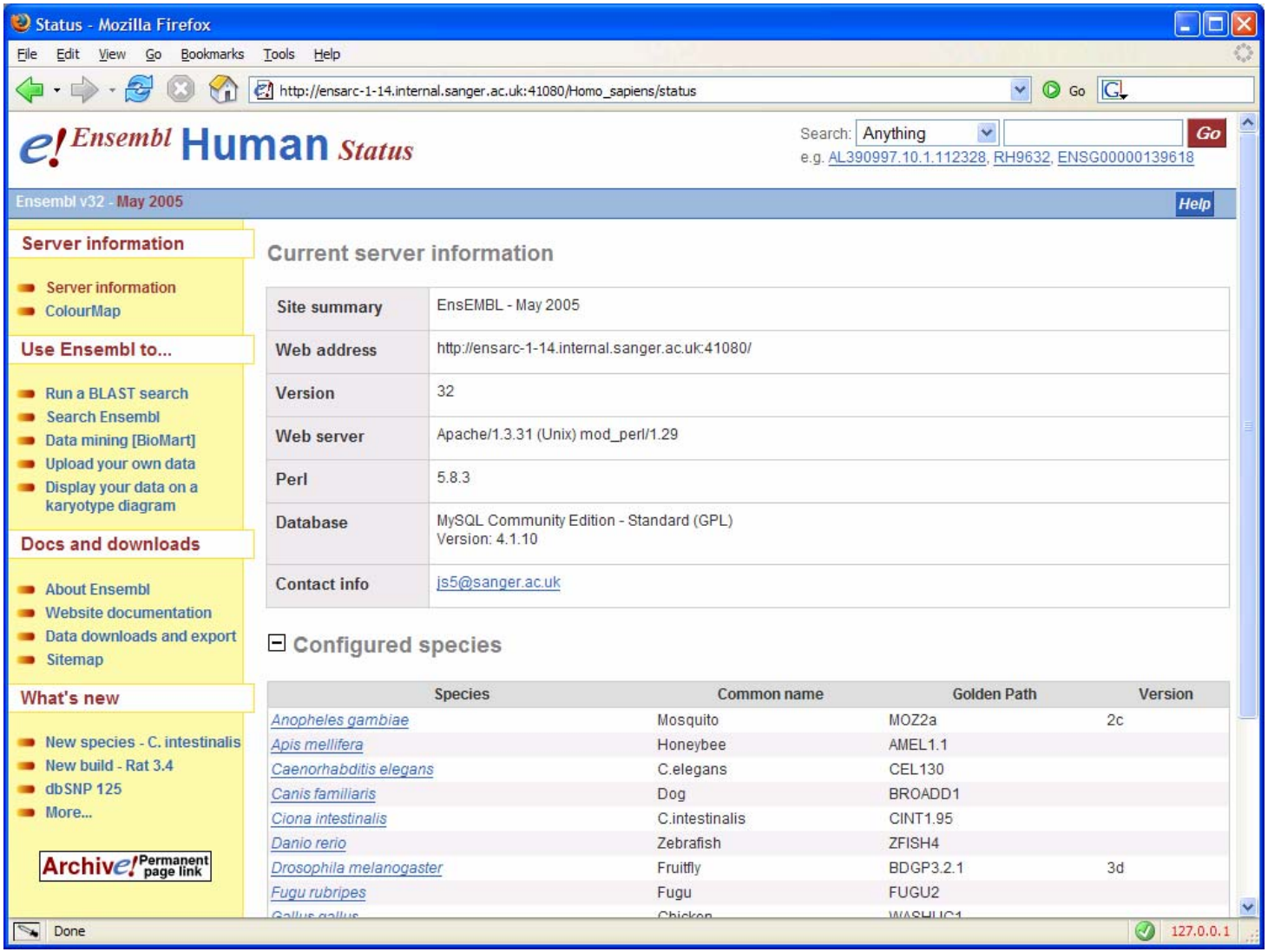
We will look at writing a status page

Current status module…

Current status script produces a plain text page:



New status display



The Script

The script needs to:

- Handle user input, either directly from the web-page parameters – or through web-page cookies
- Set up any objects required
- Handle errors that are generated
- Configure the display
- Render the display

```
$webpage = new EnsEMBL::Web::Webpage( ... )
```

```
$webpage = new EnsEMBL::Web::Webpage( ... )
```

```
$webpage->has_a_problem;
$webpage->render_error_page;
```

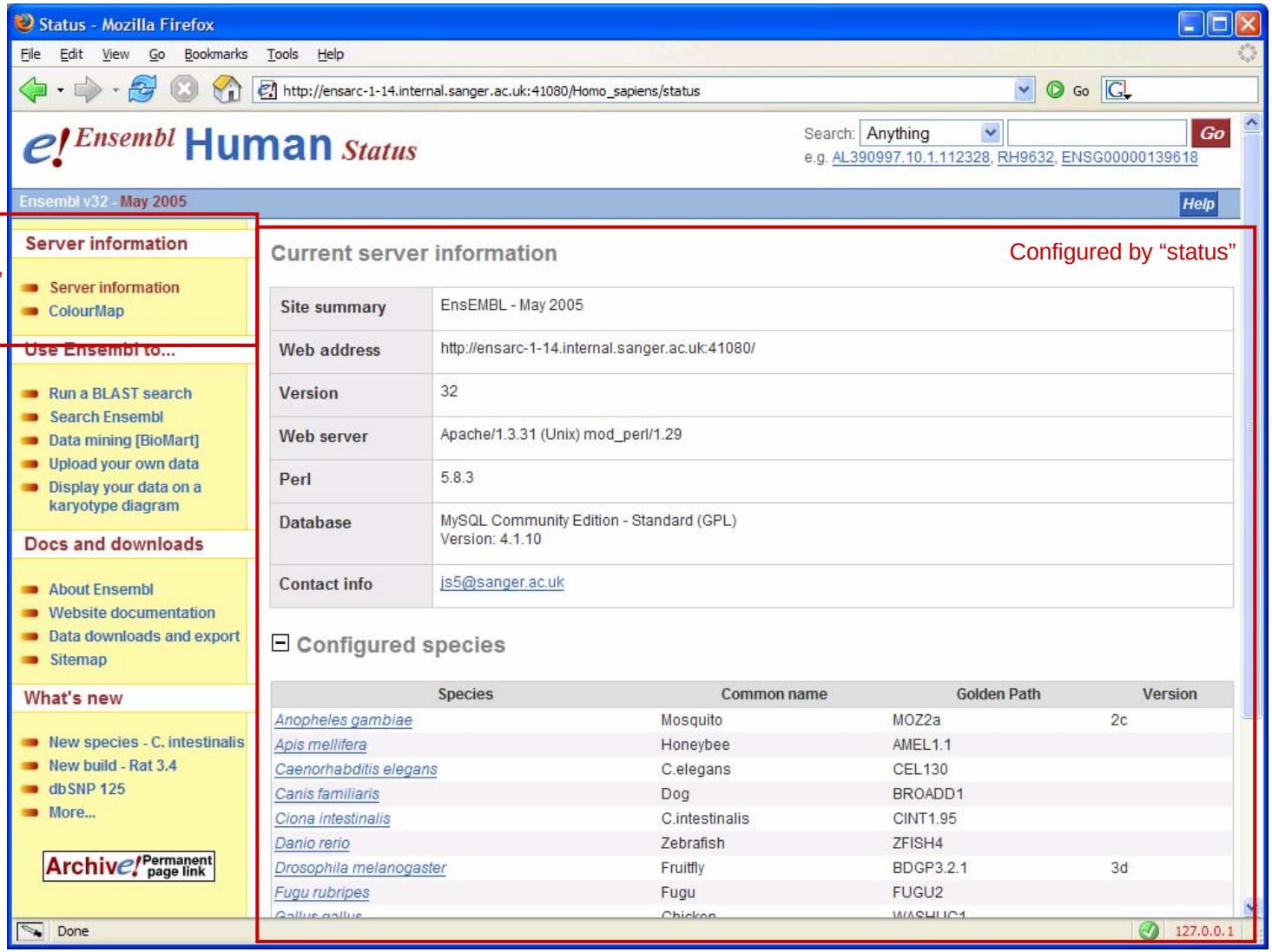
```
$webpage->configure( ... );
```

```
$webpage->render;
```

Sample webpage script


```
#!/usr/local/bin/perl

package status;

use strict;
use warnings;
no warnings "uninitialized";
use EnsEMBL::Web::WebPage;

# Initialize the page
#    EnsEMBL::Web::WebPage is effectively a webpage factory, which “creates” elements
#    the elements of the webpage – including the renderer, factory, objects, etc.
# Parameters
#   'scriptname‘ => [opt] set if want to use alt. scripts input checks
#   'objecttype‘ => [req] type of object to use (B::E::Object::Gene,...
#   'renderer‘   => [opt] defaults to Apache, where to send output
#   'outputtype‘ => [res] in future will may allow rendering in Text, XML, PDF, Excel…

my $webpage= new EnsEMBL::Web::WebPage( 'objecttype' => 'Server‘ );

if( $webpage->has_a_problem ) {
# Handle errors
# Show an error page if there is a problem
  $webpage->render_error_page;
} else {
# Configure the page
# Now configure the page, we need to pass configure an object so that it can get information
# and modules from  it, the usual approach here is just to loop over all objects (usually only 1!)
# Parameters to configure are a E::W::ProxyObject and a list of function calls to make on
# E::W::Configuration::objecttype
  $webpage->configure( $_, qw(status context_menu) ) foreach @{$webpage->dataObjects};

# Finally render the page
  $webpage->render();
}
```

New status display



Configuring the display

To configure the display we have to add elements to the webpage:

- The main “content” of the webpage is made up one or more panels, which we have to configure at this stage.
- The left-hand navigation menu is made up of one or more input blocks.

The configuration functions can create or extend these panels or blocks.

- To add a panel… `$self->{page}->content->add_panel( ... );`
- To add a section to a panel… `$self->{page}->content->panel( ... )->add_component_first(); etc`
- To add a menu block… `$self->{page}->menu->add_block( ... );`
- To add a menu entry… `$self->{page}->menu->add_entry( ... );`

Sample Configuration module


```
package EnsEMBL::Web::Configuration::Server;
use strict;
use EnsEMBL::Web::Configuration;
our @ISA = qw( EnsEMBL::Web::Configuration );

sub status {
  my $self = shift;
  my $content = $self->{page}->content;
  # First panel, “two-column style” displaying the server information.
  if( my $panel1 = $self->new_panel( "Information",
    'code'    => "info",                        ## Name this one “info”
    'caption' => 'Current server information',  ## This is it’s caption
    'object'  => $self->{object}                ## Finally pass it the “Web::Object” object.
  ) ) {
    $panel1->add_components(qw(
      name       EnsEMBL::Web::Component::Server::name
      url        EnsEMBL::Web::Component::Server::url
      version    EnsEMBL::Web::Component::Server::version
      webserver  EnsEMBL::Web::Component::Server::webserver
      perl       EnsEMBL::Web::Component::Server::perl
      database   EnsEMBL::Web::Component::Server::database
      contact    EnsEMBL::Web::Component::Server::contact
    ));
    $content->add_panel( $panel1 );
  }
  # Second panel, “spreadsheet table style” displaying the configured species.
  if( my $panel2 = $self->new_panel( "SpreadSheet“,
    'code'    => 'species',                     ## Name this one “species”
    'caption' => 'Configured species',          ## And set it’s caption
    'object'  => $self->{object},               ## Pass it the “Web::Object”
    'status'  => 'Species‘                      ## Make this panel “collapsable”
  ) ) {
    $panel2->add_components( qw(species EnsEMBL::Web::Component::Server::spreadsheet_Species));
    $content->add_panel( $panel2 );
  }
}

1;
```

```
use EnsEMBL::Web::WebPage;
my $webpage= new EnsEMBL::Web::WebPage(
  'objecttype' => 'Server‘
);
foreach @{$webpage->dataObjects} {
  $webpage->configure( $_, qw(status context_menu) );
}

$webpage->render();
```

This function creates the two panels used to generate the right hand side of the status, one based on the “two-column” format, and one based on a “spreadsheet” layout.

Sample Configuration module


```
package EnsEMBL::Web::Configuration::Server;

......

sub context_menu {
  my $self = shift;
$self->add_block( 'server', 'bulleted', 'Server information' );
  $self->add_entry( ‘server',
    'text'    => 'Server information',
    'href'    => "/$ENV{'ENSEMBL_SPECIES'}/status“
  );
  my $colourmap_URL = "/$ENV{'ENSEMBL_SPECIES'}/colourmap";
  my @colourmap_OPT = (
    [ 'Sorted by Red, Green'              => “$colourmap_URL?sort=rg
    [ 'Sorted by Red, Blue'               => “$colourmap_URL?sort=rb
    [ 'Sorted by Green, Red'              => “$colourmap_URL?sort=gr
    [ 'Sorted by Green, Blue'             => “$colourmap_URL?sort=gb
    [ 'Sorted by Blue, Red'               => “$colourmap_URL?sort=br
    [ 'Sorted by Blue, Green'             => “$colourmap_URL?sort=bg
    [ 'Sorted by Hue, Saturation'         => “$colourmap_URL?hls=hs
    [ 'Sorted by Hue, Luminosity'         => “$colourmap_URL?hls=hl
    [ 'Sorted by Luminosity, Hue'         => “$colourmap_URL?hls=lhs”  ],
    [ 'Sorted by Luminosity, Saturation'  => “$colourmap_URL?hls=lsh”  ],
    [ 'Sorted by Saturation, Hue'         => “$colourmap_URL?hls=shl”  ],
    [ 'Sorted by Saturation, Luminosity'  => “$colourmap_URL?hls=slh”  ],
  );
  $self->add_entry( 'server',
    'text'    => 'ColourMap',
    'href'    => $colourmap_URL,
    ‘options' => [map {{ 'href' => $_->[1], 'text' => $_->[0] }} @colourmap_OPT ]
  );
}

1;
```

```
use EnsEMBL::Web::WebPage;
my $webpage= new EnsEMBL::Web::WebPage(
  'objecttype' => 'Server‘
);
foreach @{$webpage->dataObjects} {
  $webpage->configure( $_, qw(status context_menu) );
}

$webpage->render();
```

This function creates a “left-hand-side” navigation menu block, and adds entries for the status page, and also the colour-map script.
This menu is shared with the colour map script.

Handling persistent user info

- There are two types of user configuration stored on the server based on a users cookie. You will already be familiar with the EnsEMBL::Web::UserConfig::… objects for configuring the images on the pages – these will be kept.
- We are now adding a second type, EnsEMBL::Web::ScriptConfig::, these hold script specific information. You create a EnsEMBL::Web::ScriptConfig object which is configured by the package EnsEMBL::Web::ScriptConfig::*{scriptname}*, and/or “plugin” packages.
- This package contains a single function “**init**” which acts on the E::W::SC object when it is created and sets the default values for the ScriptConfig object.
- The storage of the cookie is “transparent” to the script – it is embedded in the “new” call for the EnsEMBL::Web::WebPage object, so you don’t have to worry about calling it.

Sample ScriptConfig module


```
package EnsEMBL::Web::ScriptConfig::status;

use strict;

sub init {
  my ($script_config) = @_;

  $script_config->_set_defaults(qw(
    panel_species on
  ));
}
1;
```

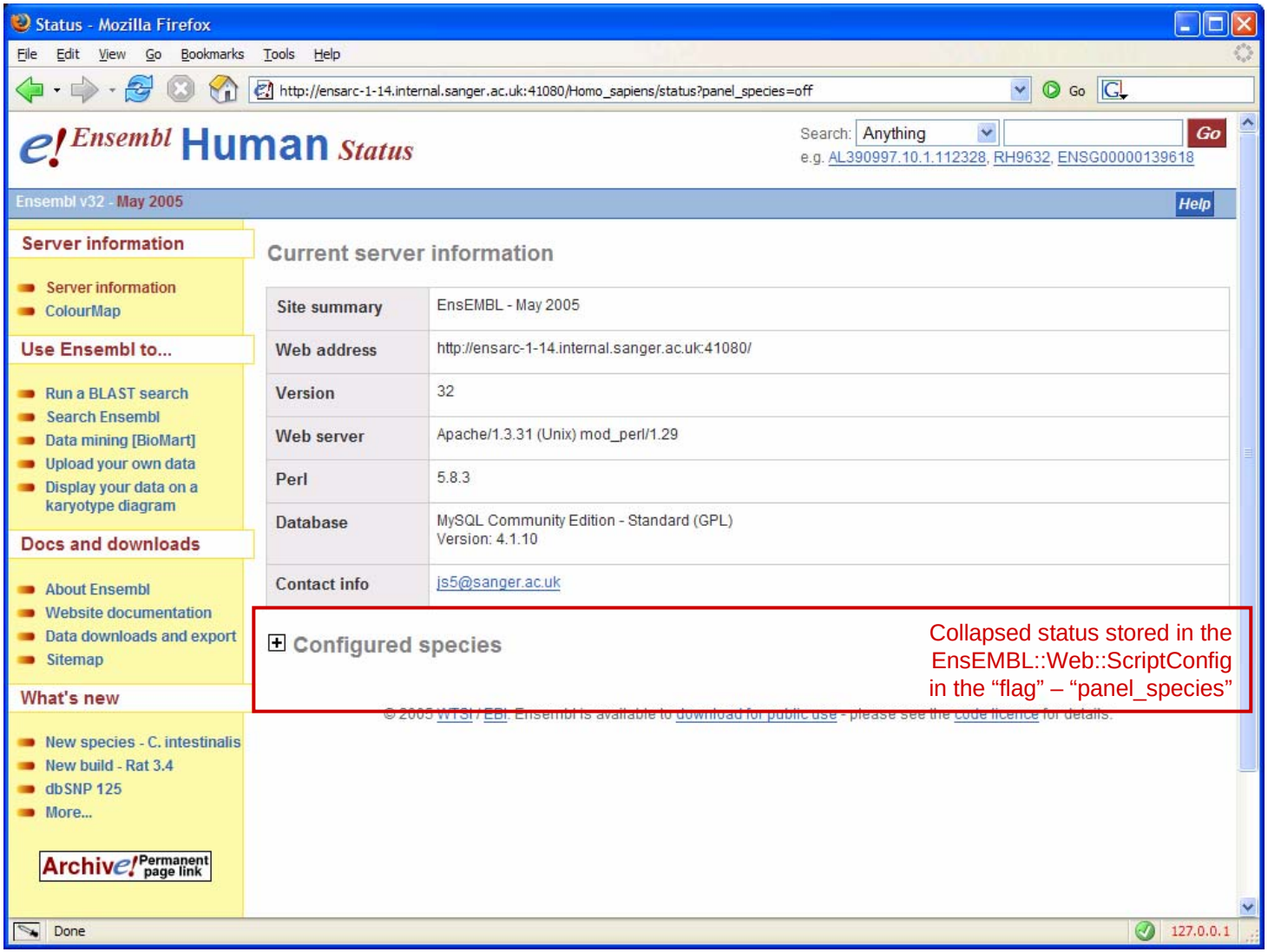
```
use EnsEMBL::Web::WebPage;
my $webpage= new EnsEMBL::Web::WebPage(
  'objecttype' => 'Server'
);
foreach @{$webpage->dataObjects} {
  $webpage->configure( $_, qw(status context_menu) );
}

$webpage->render();
```

ScriptConfig's are not "objects" they contain just a single function "init" which is called with the ScriptConfig object passed to it as the first parameter, and act upon this ScriptConfig object.

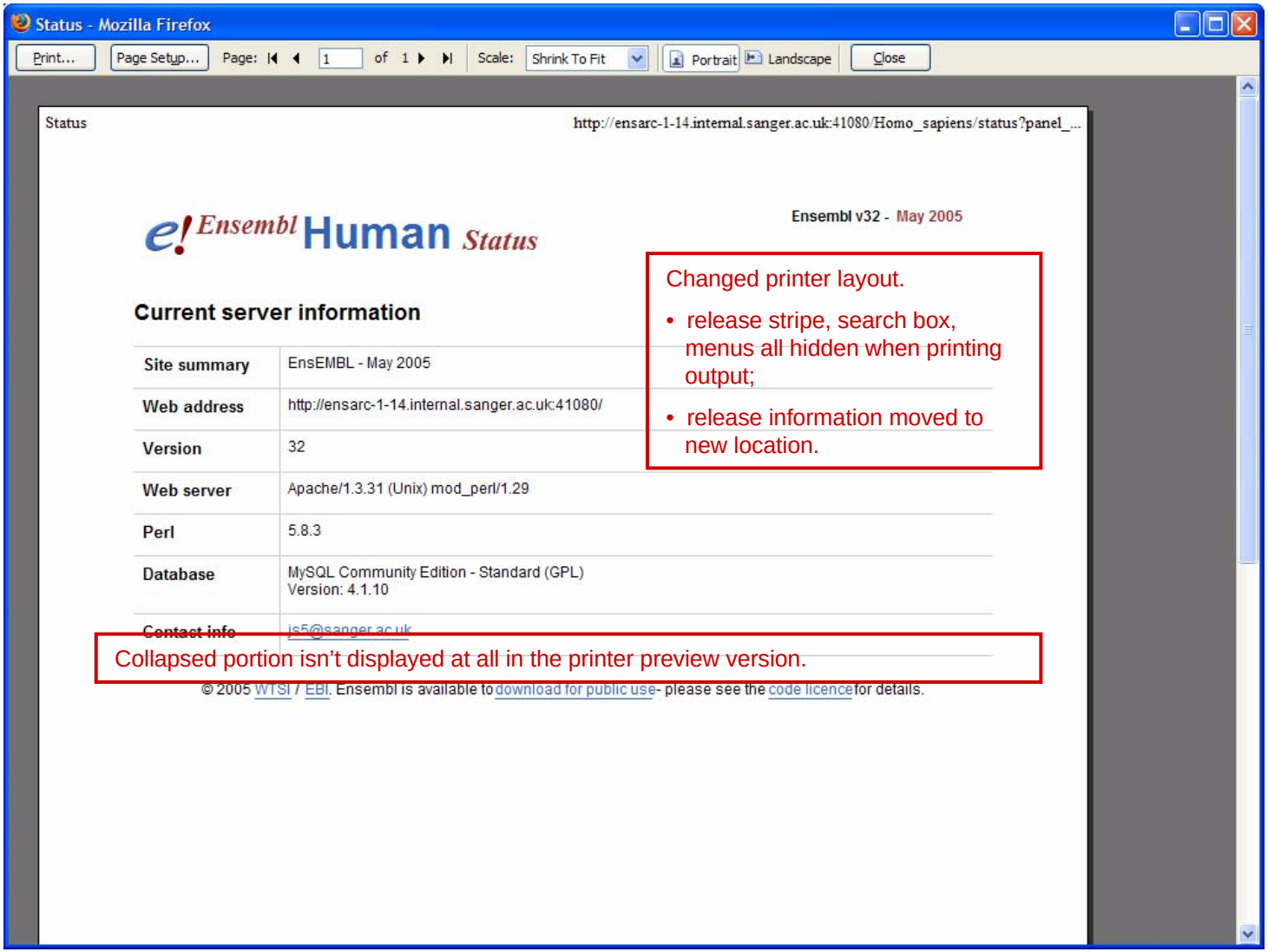
The modules name is based on the name of the script which is being configured.

In this case we are configuring the "species" panel to be collapsible.

Status display with collapsed panel



Printing collapsed panels (or not!)



Sample Component module


```
package EnsEMBL::Web::Component::Server;

use strict;


sub name {
  my( $panel, $object ) = @_;
  (my $DATE = $object->species_defs->ARCHIVE_VERSION ) =~ s/(\d+)/ \1/;
  $panel->add_row( 'Site summary',
    qq(<p>@{[$object->species_defs->ENSEMBL_SITETYPE]} - $DATE</p>)
  );
  1;
}

sub url {
  my($panel, $object) = @_;
  $panel->add_row( 'Web address',
    qq(<p>@{[ $object->full_URL( 'species' => '' ) ]}</p>)
  );
  return 1;
}

sub version {
  my($panel, $object) = @_;
  $panel->add_row( 'Version',
    qq(<p>@{[$object->species_defs->ENSEMBL_VERSION]}</p>)
  );
  return 1;
}

sub webserver {
  my($panel, $object) = @_;
  $panel->add_row(
    'Web server', qq(<p>$ENV{'SERVER_SOFTWARE'}</p>)
  );
  return 1;
}
```

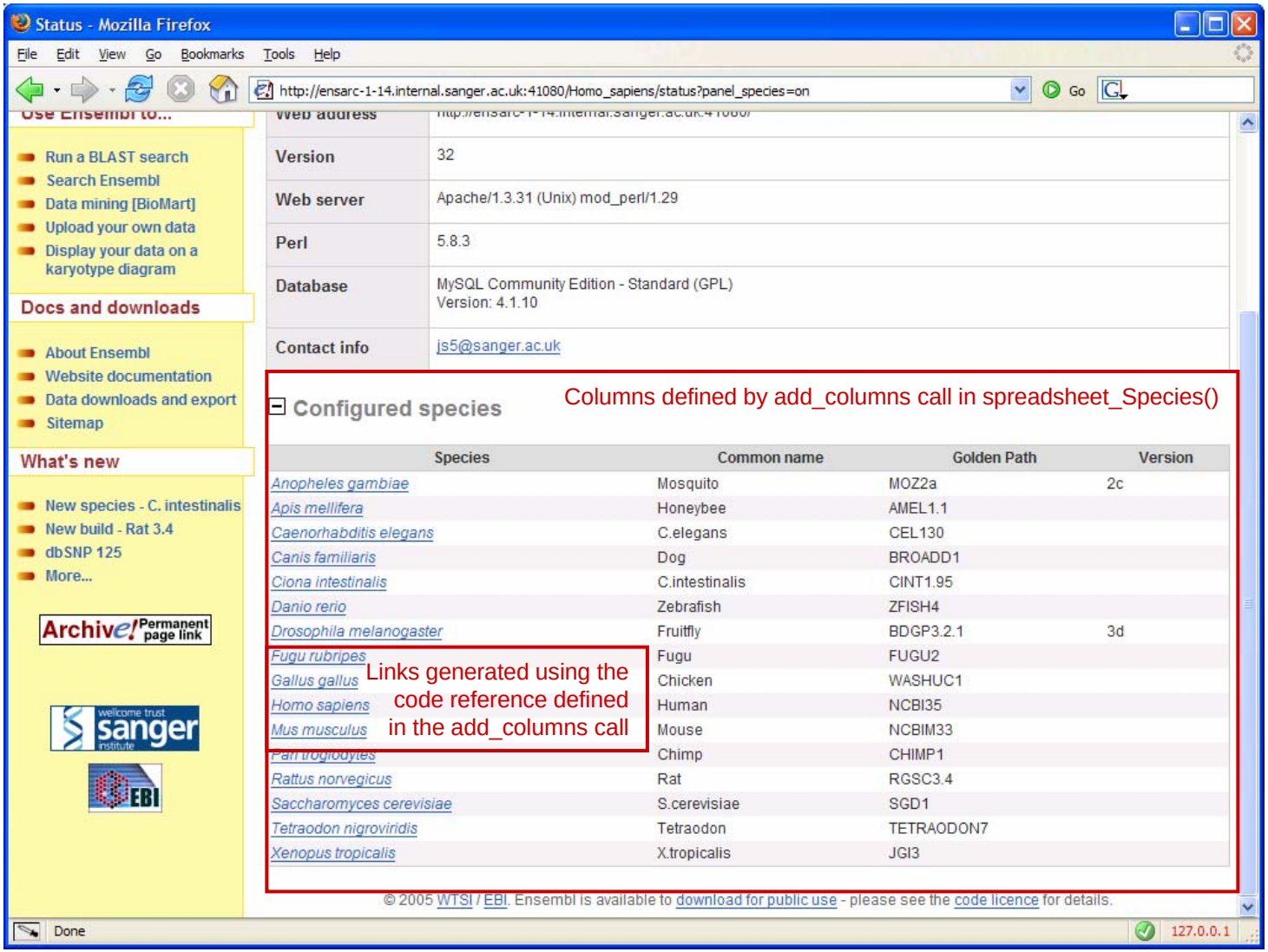
```
$panel1->add_rows(qw(
  name       EnsEMBL::Web::Component::Server::name
  url        EnsEMBL::Web::Component::Server::url
  version    EnsEMBL::Web::Component::Server::version
  webserver  EnsEMBL::Web::Component::Server::webserver
  perl       EnsEMBL::Web::Component::Server::perl
  database   EnsEMBL::Web::Component::Server::database
  contact    EnsEMBL::Web::Component::Server::contact
));
```

Sample Component module


```
package EnsEMBL::Web::Component::Server;

use strict;

sub spreadsheet_Species {
  my( $panel, $object ) = @_;
  $panel->add_columns(
    { 'key'   => 'species',
      'align' => 'left',
      'title' => 'Species',
      'format' => sub { return sprintf( qq(<a href="%s"><i>%s</i></a>), $_[1]{'link'}, $_[0] ) } },
    { 'key' => 'common',
      'align' => 'left',
      'title' => 'Common name' },
    { 'key'   => 'gp',
      'align' => 'left',
      'title' => 'Golden Path' },
    { 'key'   => 'version',
      'align' => 'left',
      'title' => 'Version' }
  );
  foreach( $object->get_all_species ) {
    $panel->add_row( $_ );
  }
  return 1;
}
```

```
$panel2->add_component( qw(
  EnsEMBL::Web::Component::Server::spreadsheet_Species
));
```

New status display



wellcome trust sanger institute

EBI

Sample Object module


```
package EnsEMBL::Web::Object::Server;

use strict;
use warnings;
no warnings "uninitialized";

use EnsEMBL::Web::Object;
our @ISA = qw(EnsEMBL::Web::Object);

sub get_all_species {
  my $self = shift;
  @species = @{ $self->species_defs->ENSEMBL_SPECIES };
  my @data = ();
  foreach my $species (@species) {
    (my $name = $species ) =~ s/_/ /g;
    push @data, {
      'species‘ => $name,
      'common‘  => $self->species_defs->other_species( $species, 'SPECIES_COMMON_NAME' ),
      'link‘    => $self->full_URL( 'species'=>$species ),
      'gp‘      => $self->species_defs->other_species( $species, 'ENSEMBL_GOLDEN_PATH' ),
      'version‘ => $self->species_defs->other_species( $species, 'SPECIES_RELEASE_VERSION' ),
    };
  }
  return @data;
}

1;
```

```
package EnsEMBL::Web::Component::Server;

sub spreadsheet_Species {
  my( $panel, $object ) = @_;
  $panel->add_columns( ... );
  foreach( $object->get_all_species ) {
    $panel->add_row( $_ );
  }
  return 1;
}
```

This object is very simple as it doesn't have "ensembl object" attached, as we can get everything we need from the species_defs call…

The columns in the spreadsheet have keys "species", "common", "gp" and "version". The additional hash element "link" is used code reference:

```
'format' => sub { return sprintf( qq(<a href="%s"><i>%s</i></a>), $_[1]{'link'}, $_[0] ) }
```

This code reference, reformats the "species" column to be wrapped in an web-link. For the code reference `$_[0]` is the appropriate element of the row (in this case `$row{'species'}`) and `$_[1]` is the whole row.

Sample Factory module

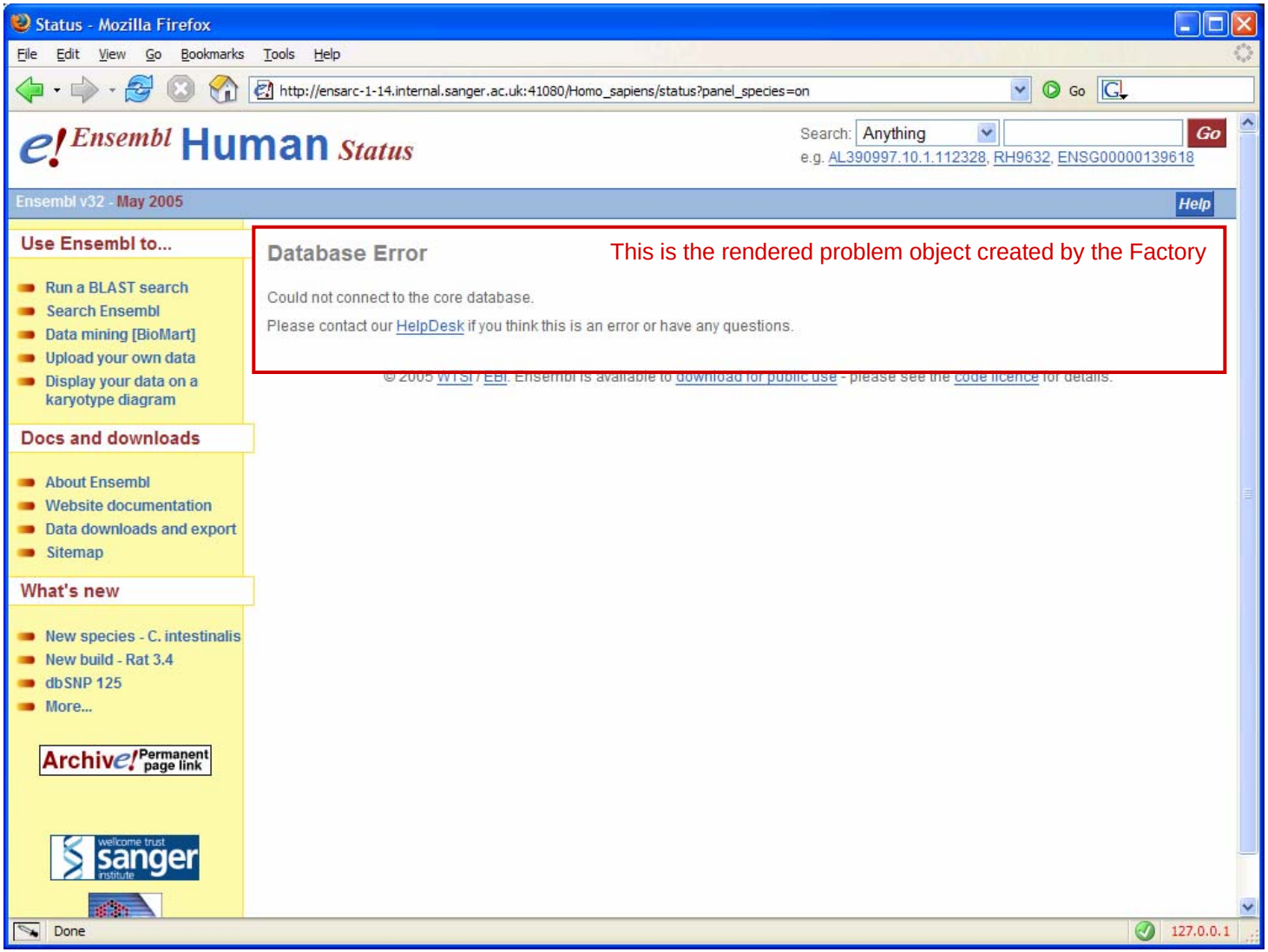

```
package EnsEMBL::Web::Factory::Server;

use strict;
use warnings;
no warnings "uninitialized";

use EnsEMBL::Web::Factory;
use EnsEMBL::Web::ProxyObject;

our @ISA = qw(EnsEMBL::Web::Factory);

sub createObjects {
  my $self = shift;
  return $self->problem( 'fatal', 'Database error', 'Could not connect to core database.' )
    unless $self->database('core');
  $self->DataObjects( EnsEMBL::Web::Proxy->new( 'Server', '', $self->DBConnection, $self->Input );
}

1;
```

```
if( $webpage->has_a_problem ) {
  $webpage->render_error_page;
} else {
  ...
}
```

This factory is very simple as it doesn't have to create any Ensembl objects, usually the second parameter to Proxy is the "Bio::EnsEMBL" object.

The problem object is created to propogate errors back to the web-browser.

New status display



Extending the Ensembl Website – Part 2: Plugins.

James Smith
Js5@sanger.ac.uk
Ensembl Web Team Project Leader

Plugins – why?

We have implemented plugins for four main reasons:

- Makes our lives easier maintaining a number of different Ensembl sites within the web-team… www, dev, pre, archive, vega, personal development environments and the “public” mirror code.
- Will make those who create there own local Ensembl installations lives easier – as the can create a personalised “configuration” file which they can copy into any new installation, and it should just work!
- Will make those who extend their local installation of Ensembl lives easier – as any “additional” code is stored in their own Plugin space.
- Should allow for “external” groups to develop features in Ensembl, which can be incorporated into anyones “Ensembl” mirror without being merged into the main code.

Plugins – using to configure e!

The simplest use of plugins is to store “local” overrides for the configuration.

- First: edit the conf/Plugins.pm file to specify where and in what name space your plugin is stored:

```
## To get standalone scripts to run from anywhere – hard code your server-root here!!
$SiteDefs::ENSEMBL_SERVERROOT = ‘/my_ensembl_install/xtreme';

$SiteDefs::ENSEMBL_PLUGINS = [
## 'MyModule::NameSpace' => 'Directory/root/of/plugin/directory',
  'EnsEMBL::MyMirror'    => $SiteDefs::ENSEMBL_SERVERROOT.'/mymirror/',
];
```

- Second: create your plugin directory and the conf directory within it, in this case
  - /my_ensembl_install/xtreme/mymirror/ and
  - /my_ensembl_install/xtreme/mymirror/conf/
- Third: create your own SiteDefs.pm, DEFAULTS.ini and any other species “.ini” files in this directory to over-ride the standard settings.

Plugins – SiteDefs.pm


```
package EnsEMBL::MyMirror::SiteDefs;

use strict;

## update_conf function called by main SiteDefs.pm
sub update_conf {
  $SiteDefs::ENSEMBL_PORT                   = 80;
  $SiteDefs::ENSEMBL_USER                   = ‘my_ensembl_user';
  $SiteDefs::ENSEMBL_GROUP                  = ‘my_ensembl_group';

  $SiteDefs::ENSEMBL_SERVERADMIN            = ‘me@my_ensembl.mydomain.org';
  $SiteDefs::ENSEMBL_SERVERNAME             = 'my_ensembl.mydomain.org';
  $SiteDefs::ENSEMBL_MAIL_ERRORS            = 1;
  $SiteDefs::ENSEMBL_ERRORS_TO              = ‘me@mydomain.org’;

  $SiteDefs::ENSEMBL_DEBUG_FLAGS            = 24;

  $SiteDefs::ENSEMBL_USERDB_NAME            = 'ensembl_web_user_db';
  $SiteDefs::ENSEMBL_USERDB_HOST            = 'my_mysql.mydomain.org';
  $SiteDefs::ENSEMBL_USERDB_PORT            =  3306;
  $SiteDefs::ENSEMBL_USERDB_USER            = ‘my_mysql_user';
  $SiteDefs::ENSEMBL_USERDB_PASS            = ‘my_mysql_pass';
  $SiteDefs::ENSEMBL_LONGPROCESS_MINTIME    = 20;
}

1;
```

Plugins – DEFAULTS.ini


```
[general]
ENSEMBL_HOST       = my_mysql.mydomain.org
ENSEMBL_HOST_PORT  = 3306
ENSEMBL_WRITE_USER = my_mysql_user
ENSEMBL_WRITE_PASS = my_mysql_pass
ENSEMBL_DBUSER     = my_mysql_ro
ENSEMBL_DBPASS     =

ENSEMBL_BINARIES_PATH   = /usr/local/bin             ; DotterView
ENSEMBL_EMBOSS_PATH     = /usr/local/emboss          ; AlignView
ENSEMBL_WISE2_PATH      = /usr/local/wise2           ; AlignView
ENSEMBL_BLAST_BIN_PATH  = /data/bin/wu_blast         ; BlastView
ENSEMBL_REPEATMASKER    = /data/bin/RepeatMasker     ; BlastView
ENSEMBL_BLAST_FILTER    = /data/bin/wu_blast/filter  ; BlastView
ENSEMBL_BLAST_MATRIX    = /data/bin/wu_blast/matrix  ; BlastView
ENSEMBL_BLAST_DATA_PATH = /data/blastdb/ensembl      ; BlastView

ENSEMBL_SEARCH          = textview

etc ...
```

Plugins – using to configure e!

Here we use a “standard” Sanger plugin, plus local “development” plugins.

```
## To get standalone scripts to run from anywhere – hard code your server-root here!!
$SiteDefs::ENSEMBL_SERVERROOT = ‘/my_ensembl_install/xtreme';

$SiteDefs::ENSEMBL_PLUGINS = [
## 'MyModule::NameSpace' => 'Directory/root/of/plugin/directory',
  'EnsEMBL::MyMirror_js5' => $SiteDefs::ENSEMBL_SERVERROOT.'/mymirror_js5/',
  'EnsEMBL::MyMirror'     => $SiteDefs::ENSEMBL_SERVERROOT.'/mymirror/',
];
```

This extra “plugin” conf contains just the following SiteDefs.pm file:

```
package EnsEMBL::Sanger_js5::SiteDefs;

sub update_conf {
  $SiteDefs::ENSEMBL_PORT                   = 41080;
  $SiteDefs::ENSEMBL_USER                   = 'js5';
  $SiteDefs::ENSEMBL_SERVERADMIN            = 'js5@mydomain.org';
  $SiteDefs::ENSEMBL_SERVERNAME             = ‘my_ensembl.mydomain.org';
  $SiteDefs::ENSEMBL_ERRORS_TO              = 'js5@mydomain.org';
  $SiteDefs::ENSEMBL_DEBUG_FLAGS            = 24;
  $SiteDefs::ENSEMBL_LONGPROCESS_MINTIME    = 5;
}

1;
```

Extending the Ensembl Website – Part 3: Extending the script.

James Smith
Js5@sanger.ac.uk
Ensembl Web Team Project Leader

Extending this display

- Want to add a “development” plugin which displays the Apache webserver environment information.
- Need additional Configuration::Server module to add new panel
- Need additional to Object::Server module to collect “environment information”
- Need additional Component to configure the additional panel
- Required to update Plugins.pm in conf directory to include module.

- Additionally the plugin will add a “rudimentary” sitemap for the static content to the top panel.

Plugins module


```
## To get plugins to work with standalone scripts
## you need to specify the server root here!!
$SiteDefs::ENSEMBL_SERVERROOT = '/ensemblweb/js5/wwwdev/xtreme';

## Add our module, Key is namespace, Value is directory containing plugin.

$SiteDefs::ENSEMBL_PLUGINS = [
  'EnsEMBL::Sanger‘      => $ENSEMBL_SERVERROOT.'/sanger-plugins/sanger',
  'EnsEMBL::Sanger_js5‘  => $ENSEMBL_SERVERROOT.'/sanger-plugins/js5',
  'EnsEMBL::Development‘ => $ENSEMBL_SERVERROOT.'/public-plugins/development',
];

1;
```

Plugin Configuration module


```
package EnsEMBL::Development::Configuration::Server;

use strict;
use EnsEMBL::Web::Configuration;
our @ISA = qw( EnsEMBL::Web::Configuration );

use EnsEMBL::Web::Document::Panel::SpreadSheet;

sub status {
  my $self = shift;
  if( my $panel2 = $sefl->new_panel( 'SpreadSheet',
    'code‘    => 'environment',
    'caption‘ => 'Apache environment',
    'object‘  => $self->{object},
    ‘status’  => ‘panel_apache’
  ) ) {
    $panel2->add_components(qw(apache
    EnsEMBL::Development::Component::Server::spreadsheet_Apache));
    $content->add_panel_last( $panel2 );
  }
  if( my $panel_ref = $self->panel( 'info' ) ) {
    $panel_ref->add_component_last(qw(tree
    EnsEMBL::Development::Component::Server::static_tree));
  }
}

1;
```

Plugin ScriptConfig


```
package EnsEMBL::Development::ScriptConfig::status;

use strict;

sub init {
  $_[0]->_set_defaults(qw(
    panel_apache on)
  );
}

1;
```

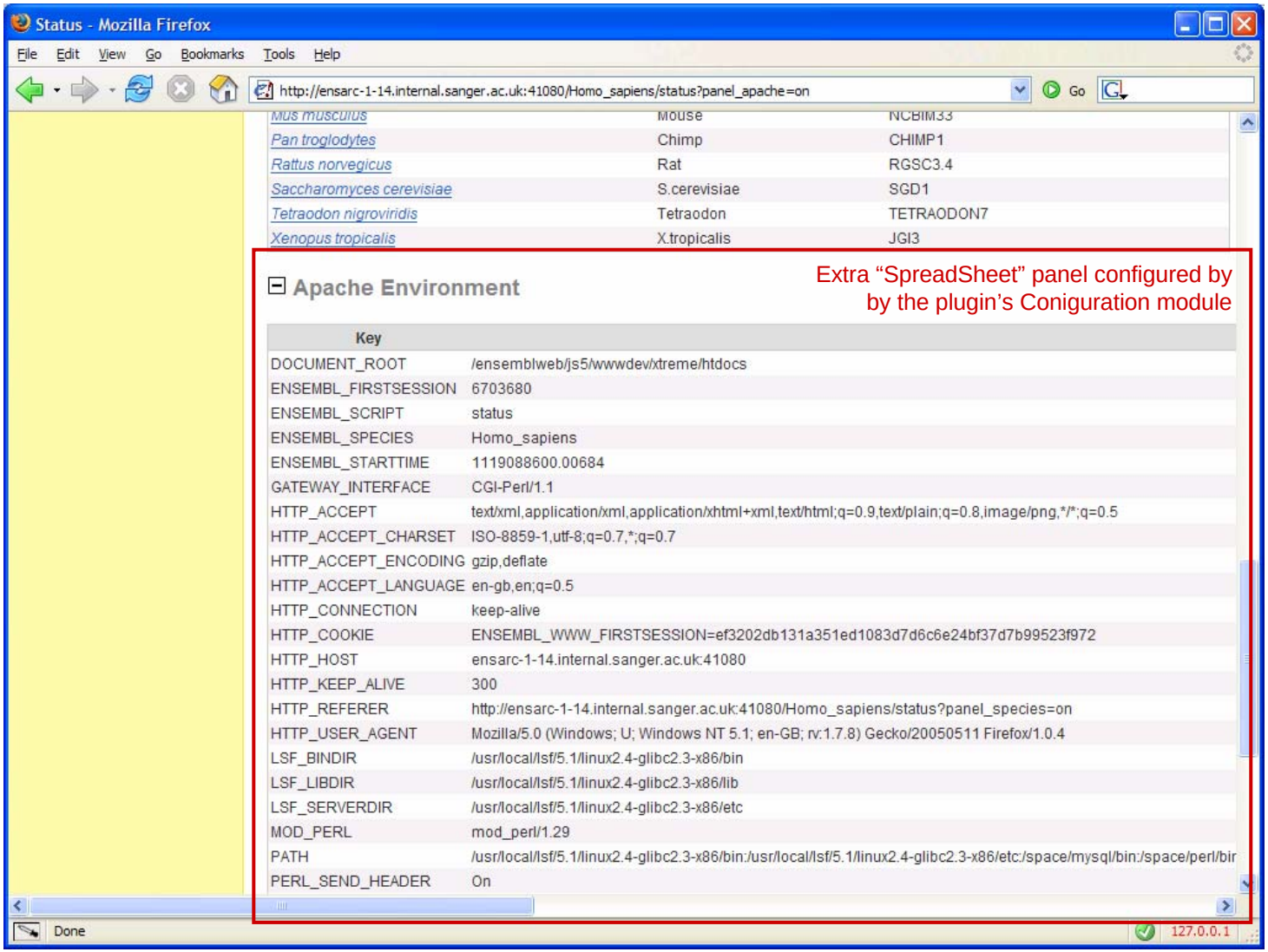
Plugin Component module


```
package EnsEMBL::Development::Component::Server;

use strict;

sub spreadsheet_Apache {
  my( $panel, $object ) = @_;
  $panel->add_columns(
    { 'key' => 'key',    'align' => 'left', 'title' => 'Key' },
    { 'key' => 'value',  'align' => 'left', 'title' => 'Value' },
  );
  foreach (@{$object->get_environment}) {
    $panel->add_row( $_ );
  }
  return 1;
}

sub static_tree {
  my($panel,$object) = @_;
  $panel->add_row( 'Document tree', _sub_tree( $object, '/', 0) );
  return 1;
}

sub _sub_tree {
  my( $object, $K, $i ) = @_;
  my $HTML = qq(@{[' ' x ($i*2)]}<a href="$K">@{[$object->species_defs->ENSEMBL_BREADCRUMBS->{$K}[0]]}</a><br />\n);
  foreach my $C (@{$object->species_defs->ENSEMBL_CHILDREN->{$K}||[]})  {
    $HTML .= _sub_tree( $object, $C, $i+3 );
  }
  return $HTML;
}

1;
```

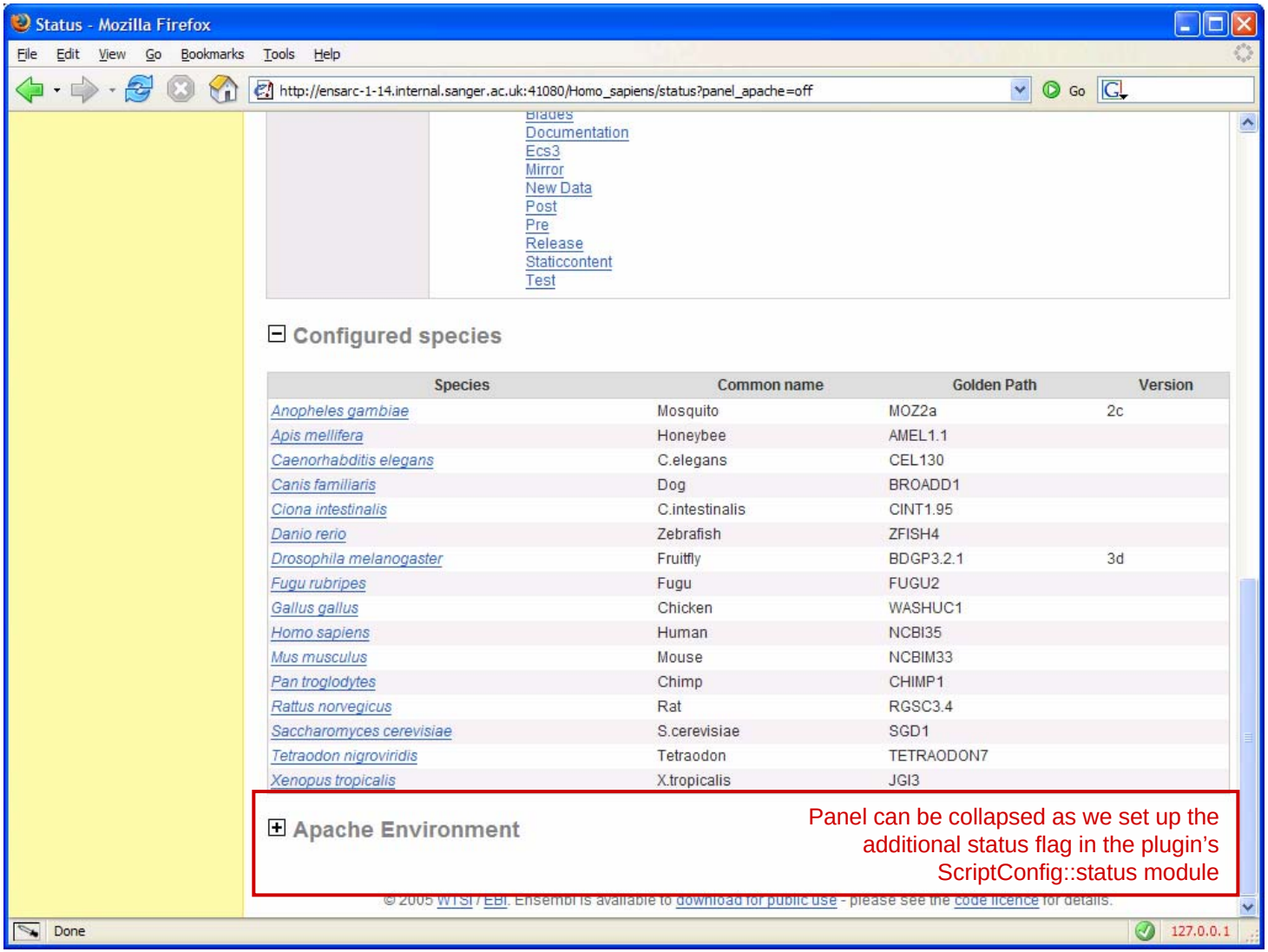
Plugin Object module


```
package EnsEMBL::Development::Object::Server;

use strict;
use warnings;
no warnings "uninitialized";

use EnsEMBL::Web::Object;
our @ISA = qw(EnsEMBL::Web::Object);

sub get_environment {
  my $self = shift;
  return [ map { {'key'=>$_,'value'=>$ENV{$_}} } sort keys %ENV ];
}

1;
```

Additional “Development” panel



Additional “Development” panel



Additional “Development” panel

Additional menu Item added by Sanger plugin