



Table of Contents

Table of Contents	1
Introduction	2
Skins	2
Introduction to Skins.....	2
Creating Skins	2
Basics – Technology	3
Basics – The components	4
Assembling the components	4
Diagram 1. – Assembling the components.	5
Specific Components	5
CSS	5
pagelayout.....	6
Diagram 2 – Sample screen with the different elements outlined.	7
pagelayoutmini.....	8
navbar and admin_navbar	8
Help Text	8
A Brief Summary of Zope’s ptl.....	9



Introduction

This document serves to describe the key components used in the Jet system that affect its visual appearance. This document should provide enough information that any reasonably savvy web designer can make medium complexity visual changes to the interface.

When discussing the visual appearance of Jet's interface screens we use the term skins to describe a group of files that together form an individual look.

For a client to be able to change the look/feel of the Jet interface, they will have to have either the **customised branding module** or the **multiple skins module**. The ability to edit your own skin for branding reasons, or to have multiple skins under Jet is not normally included with the base Jet package.

As the Jet system evolves further due to ongoing development, this documentation will become outdated fairly rapidly. Please ensure you have the most recent copy of this document.

Skins

Introduction to Skins

A **skin** defines the overall visual appearance of the Jet interface screens. A number of files combine to create a skin. These files are grouped together in the skins subsection of the Jet source tree.

If you have purchased one of the skins modules, you should have access (generally via a CVS repository set up for you, but arrangements can vary depending upon your needs) to a copy of the skins directory and subdirectories. By manipulating these files you can change the look and feel of Jet to a very high degree.

There will always be some limitations in very small details, but these are unavoidable if the product is to be maintainable in an ongoing basis.

Creating Skins



The default skin files are found in: `ptl/<realm name>/skins/default/`. If you have access to the entire source package then you may need the full path:

`Jet\modules\zope\ptl\<realm name>\skins\default`.

Hereafter we will only use the shortened path.

For example, if your realm was named "xone" the location would be:
`ptl/xone/skins/default/`.

Any additional skins live in other directories at this location: `ptl/<realm name>/skins/new_skin/` for example. New skins will inherit files from the default skin if needed.

For Example:

Georgina wants to create a new skin for a hotel chain (the Hilton).

She makes a new directory:

`ptl/<realm name>/skins/hilton`

Inside that subdirectory Georgina puts a copy of the `style.css` file. She then edits this file to make the colours match the Hiltons Corporate colours.

She also puts a copy of the `pagelayout` file in the Hilton directory as well as a gif of the Hilton's logo. She edits the `pagelayout` file to display the Hilton logo instead of the default logo at the top of the page.

The `ptl/<realm name>/skins/hilton` directory now contains three files: `pagelayout`, `logo.gif`, and `style.css`. It will inherit any other files it needs from the default set in the `ptl/<realm name>/skins/default` directory.

Basics – Technology

The Jet interface is based on the following technologies –

- Html (xhtml to be specific - therefore the documents have to be well formed).
- CSS (Cascading Style Sheets) - currently only css1 elements are used for compatibility reasons, css2 or later could be used if desired.
- Zope technologies – ptl, (metal and tal).



Knowledge of the Zope technologies shouldn't be required to edit the interface appearance, or for minor textual changes.

Basics – The components

There are six main files that affect Jets layout and visual appearance:

CSS (cascading style sheet) file - This controls colours, font size, table styles etc. The CSS is referenced from the pagelayout template.

pagelayout – this is the main template used to control the layout of the Jet interface, the location of the different elements on the screen.

pagelayoutmini – this template serves a similar purpose as pagelayout, however it is a simplified and smaller (visually) version designed for use in the popup monitor.

table_render – this template is used when presenting tabular data – generally drawn from a database. This defines the opening and closing tags used in tables.

table_heading_render – this template is used to render the heading rows on tables. Currently this template assumes that it is being called from within an already opened table and that the table has 3 columns.

group_render – In parts of the Jet interface, tabular information is drawn from a database iteratively and placed on the screen. This occurs on most screens that have input fields or tabular data – the register screen for example. group_render is used to generate those table rows.

line_render – creates a single table row containing three cells – name, content, helptext.

navbar – creates the navigation bar. Allows access to the basic tools a normal user would use.

admin_navbar – creates the admin navbar, allows access to the more powerful administrators tools.

Assembling the components

Lets have a look at how these components get used and called to create a finished Jet page in a browser.

A user makes a call to a **file** via a URL – they have gone into jet and are calling up the login screen for example.

This **file** contains content organised into "slots".

The **file** makes a call to **pagelayout** and passes it the content. In some cases **pagelayoutmini** is called instead, for the purposes of this example **pagelayout** and **pagelayoutmini** are interchangeable.

pagelayout decides which template layout to use (standard or pda) and then inserts the content into the appropriate slots in that template.

pagelayout also links in the appropriate **CSS** file, which is used by the users browser to style the final content.

pagelayout calls **navbar** and **admin_navbar** and puts them in their normal position. **navbar** and **admin_navbar** themselves check to see if they should appear (is the user logged in? are they an administrator? etc).

If the content contains further calls to templates (like **table_data**) this is now processed.

The finished page is then output to the users browser.

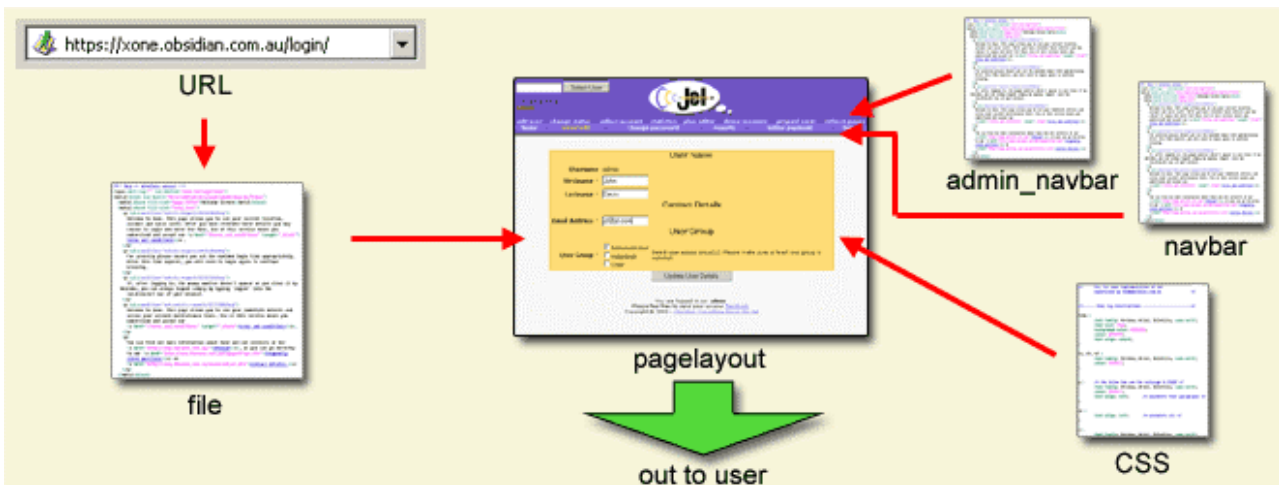


Diagram 1. – Assembling the components.

Specific Components

CSS



Default Location: `ptl/<realm name>/skins/<skin name>/style.css`

The CSS file is used as cascading style sheets are normally used – to control the style and appearance of content – specifically font sizes, font types, margins, colours and so on.

The CSS contains both style definitions for vanilla html elements (h1, h2, p, body etc) as well as some specifically defined classes. The CSS contains inline documentation that describes what each class or group of classes is used for, however any changes should obviously be carefully tested, especially for cross browser conformance and to ensure the content and styling degrades gracefully with earlier versions of browsers (a notoriously difficult task to do well).

pagelayout

Default Location: `ptl/<realm name>/skins/<skin name>/pagelayout`

The pagelayout is the core of the appearance of Jet. It lays out where objects appear and contains links to images and logos that appear on every page. It also links to the stylesheet for the page and contains the footer.

The pagelayout templates each contain four *slots*. When a file calls pagelayout it passes it content for these slots. The slots generally have a default value that gets used if no content for that slot is passed along. The slots defined are as follows:

page_title – This is used for the `<title>` element on pages in the form: `Xone – {page_title}`.

script – This is used for passing javascript around and shouldn't be changed.

help_text – This is where help text that relates to the page as a whole goes. In the standard skin this appears just below the navbars and above any other content in the page.

primary_content – This is the main slot used. Most of the content of a page goes here.

Diagram 2 shows a sample page with the slots labelled.

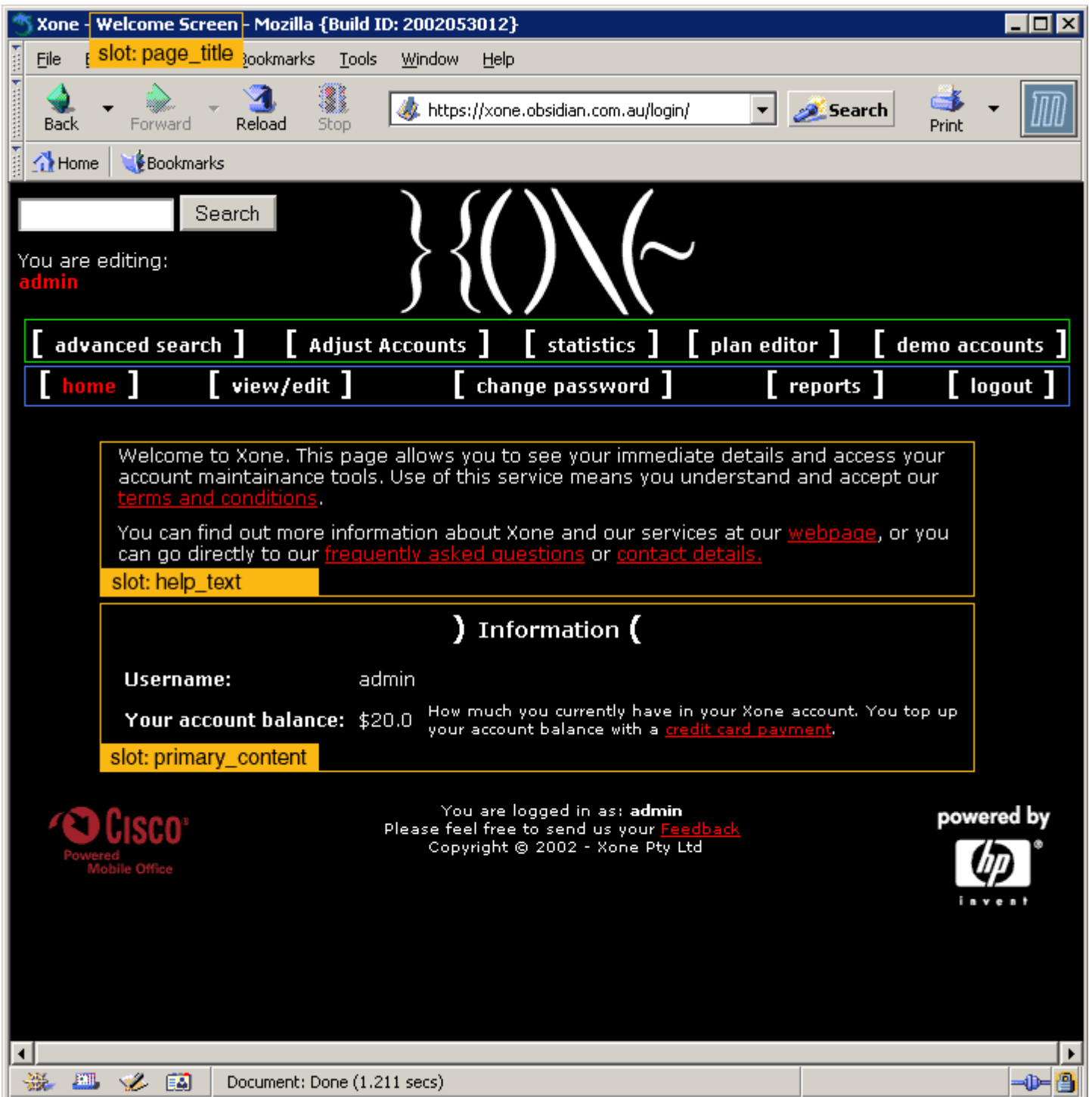


Diagram 2 – Sample screen with the different elements outlined.

- The yellow elements are all drawn from the original page (login in this case) and are labelled as to which slot they are.
- The green element is called from the admin_navbar.
- The blue element is called from the navbar.



pagelayoutmini

Default Location: `ptl/<realm name>/skins/<skin name>/pagelayoutmini`

pagelayoutmini is an alternate version of **pagelayout**. It is designed for use within the popup monitor and other similarly small windows. It has the same slots as **pagelayout**. Unlike **pagelayout**, it doesn't call **navbar** or **admin_navbar**.

navbar and admin_navbar

Default Location: `ptl/<realm name>/skins/<skin name>/navbar`
`ptl/<realm name>/skins/<skin name>/admin_navbar`

These two files are called from pagelayout and create the two navigation bars that may appear on pages (depending upon if you are logged in and who you are logged in as). They check if they are required and should be drawn. The layout (currently single row of links) of the navbars is contained in these two files. Like pagelayout they have pda and standard versions of the navbar layout.

Help Text

As well as allowing customers to change the visual appearance of Jet using skins, our software also enables clients to customise the textual content of Jet on a per skin basis. While there are natural limitations to the full extent of changes that can be made here, this is still a powerful tool for changing the textual content in Jet.

In the `ptl/<realm name>/skins/default/` directory, alongside the default skin files, is a directory named `help`. Inside that directory is a file structure that mimics the file structure of the Jet interface code. Each file contains the help text that is automatically inserted into the Jet interface screens as they are called.

For example:

The registration page (`ptl/<realm name>/registration`) when viewed through Jet, contains help text. That help text for the default skin is found in `ptl/<realm name>/skins/default/help/registration`.



Like the files that comprise the appearance of skins, the help text inherits from the default help files. Therefore if a given skin only needed to have the help text for a single page changed that could be easily achieved.

For example:

Suppose Bob needs to change the help text for the login page for a new skin that he is creating for a new airport installation.

Bob creates a new skin directory called airport:

ptl/<realm name>/skins/airport/

inside that directory he makes any visual changes required.

*Bob then copies the current login help text page from
ptl/<realm name>/skins/default/help/login/index_html to
ptl/<realm name>/skins/airport/help/login/index_html*

He then edits the new file to make the required textual changes.

A Brief Summary of Zope's ptl

PTL – page template language, is used throughout Jet. It includes TAL and METAL tags which will often look something like:

```
<metal:block fill-slot="help_text">  
</metal:block>
```

or

```
<metal:block use-macro="here/subroutines/table_data/macros/table_data">  
  <metal:block fill-slot="table_content">  
  
  </metal:block>  
</metal:block>
```

or

```
<span tal:omit-tag="" tal:define="skin here/getSkin">  
</span>
```



Changing the interface of Jet should not require any of these pti tags to be changed, and as a general rule they shouldn't be touched.

Should you wish to learn more about pti, tai, metal or dtmi:

<http://www.zope.org/> is a good place to start.