THE
TORNADO SERVER
BLACKBOOK

- 1.29 -

Puakma Tornado Server BlackBook

Contents

Overview

The Puakma Tornado Server is a cutting edge development platform built from the ground up to power the next generation of Internet based applications.

Audience

This BlackBook is written for administration and development staff.

Development Skills required

- HTML,CSS, JavaScript
- Java
- Some relational Database experience

Architecture

The Puakma server core is written in 100% Java. This means it will run on any platform with a Java 1.4 JVM, allow for it to be used on a vast range of hardware and devices. It has an incredibly small footprint (less than 400Kb for the Puakma core). Both the small footprint and ability to run on a variety of hardware allow the server to scale exceptionally well.

Puakma is designed to be middleware. It acts a little like glue in any environment, binding together legacy systems so that they appear to be a part of one seamless environment. This thinking is currently a little against the grain, since most vendors are pushing for ubiquitous environments. Since the glue analogy allows customers to utilize their existing systems and platforms, Puakma is able to blend in, without causing a major rewrite of these systems.

Tornado Server Anatomy

The Puakma server is really just a common container for many kinds of addins. A server addin may perform any task, maintain its own sockets and clients (such as the HTTP addin), schedule tasks (such as the AGENDA task) or send SMTP mail. This BlackBook will primarily discuss the HTTP addin and its features. This component based architecture means that any developer may write their own addins to extend the server base, and ensures the base server is somewhat "future-proofed" as new and emerging technologies may be easily added.
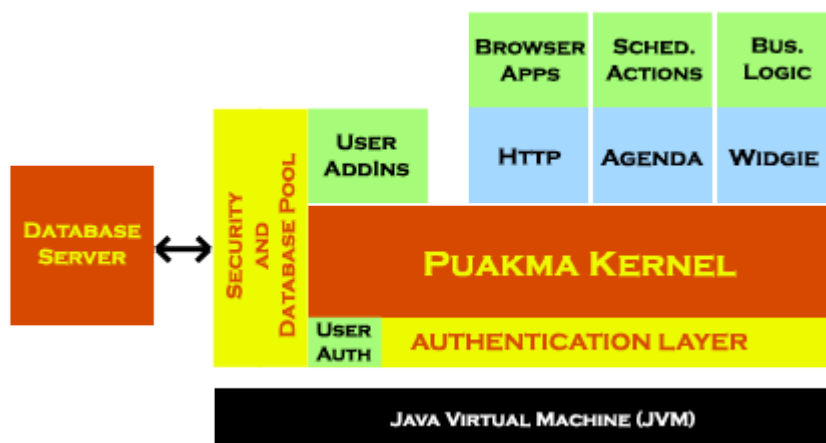
Installation

Installing Tornado Server is simple, before you start, make sure you have the following:

1. Downloaded a copy of the Sun JRE or JDK version 1.4.0 or greater (see: http://java.sun.com/) and installer from http://www.puakma.net
2. Two minutes free for the installation ;-)

For the purpose of this install process, it is assumed that Microsoft Windows NT is the operating system with the database server on the same machine. The basic techniques used here should easily translate across to other operating systems, database servers and hardware platforms.

Architecture and hardware configuration

The design of Puakma is such that almost any hardware/server configuration may be used. You may have your database servers clustered and behind a firewall, with Puakma sitting out front. You can even have multiple Puakma servers utilising the same data source. This allows you to, say, run a HTTP server on two machines, while using yet another for scheduled processing and mail. You are also able to run a multiple copies of Puakma on the same machine. The possible configurations are almost endless. Various architectures will be discussed later.

1. Installing the JDK/JRE

Go to the Sun website and download the latest JDK (Java Development Kit). The JDK is for developers and includes the JRE (You can install just the JRE, but the full SDK is recommended). Please follow Sun's install instructions. When the install has completed, type java -version at the command prompt. You should see something like the following:

```
C:>java -version
java version "1.4.0"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0-b92)
Java HotSpot(TM) Client VM (build 1.4.0-b92, mixed mode)
```

Tornado Server requires 1.4.0 or greater JDK.

2. Installing Tornado Server

Locate TornadoServerInstall.jar and double-click to execute the installation. This file is a self-launching Java application and should work on all GUI platforms. Sometimes .jar files are not associated correctly with Java applications and they will not automatically launch on some machines. If this occurs, open a command shell and type "java –jar TornadoServerInstall.jar" in the folder that contains TornadoServerInstall.jar.

Next follow the installation screens and input your installation choices.

Puakma contains an embedded database called HSQLDB. This is loaded automatically when the server is started and means that you do not need to worry about setting up a database server. If you decide that you would rather use mySQL, PostgreSQL or other RDBMS, then this may be configured. The database setup scripts for other databases are included in the /config/ folder and are named datadef.xxxx (eg datadef.mysql for mySQL). Set up the database as per the vendor's instructions and run the appropriate configuration script to create the Puakma system tables. Next copy the vendor's jdbc driver to /jdbc/, and then update puakma.config with the database connection strings (eg SystemDBURL and SystemDriverClass etc).

Finally, to start Puakma, run pmaStart.bat. You should see screen output similar to the following:

```
*** PUAKMA JAVA APPLICATION SERVER ***
CONFIG: ../config/puakma.config
SYSTEM CLASSES: puakma.jar
ADDIN CLASSES: ../addins/
LIBRARY CLASSES: ../lib/
LOADING JDBC DRIVER: hsqldb.jar
LOADING JDBC DRIVER: mysql-connector-java-3.1.8-bin.jar
LOADING JDBC DRIVER: postgresql-8.0-311.jdbc3.jar
LOADING: /data/puakma/bin/../lib/itext-1.3.jar
LOADING: /data/puakma/bin/../lib/jcifs-1.2.0.jar
LOADING: /data/puakma/bin/../lib/poi-2.5.1-final-20040804.jar
LOADING: /data/puakma/bin/../lib/velocity-1.4.jar


Puakma ENTERPRISE SERVER v3.21 Build:547 - 7 September 2005
Loading configuration from: ../config/puakma.config
Loading language file: ../config/msg_en.lang
Session Timeout: 360 min.
Anonymous Session Timeout: 1 min.


07/09/2005 19:06:24: (I) System hostname 'dolphin.wnc.net.au'  (SYSTEM -
pmaSystem)
07/09/2005 19:06:24: (I) Authenticator:
'puakma.security.pmaDefaultAuthenticator' loaded  (SYSTEM - pmaSystem)

07/09/2005 19:06:24: (E) ###### THIS SERVER HAS NOT BEEN LICENSED ######
(SYSTEM - pmaSystem)
07/09/2005 19:06:24: (E) ######### FOR NON-COMMERCIAL USE ONLY #########
(SYSTEM - pmaSystem)
07/09/2005 19:06:24: (E) #########   http://www.puakma.net   #########
(SYSTEM - pmaSystem)
07/09/2005 19:06:29: (I) PUAKMA Startup (Puakma ENTERPRISE SERVER v3.21
Build:547 - 7 September 2005)  (SYSTEM - pmaServer)
07/09/2005 19:06:29: (I) System console is ready for commands  (SYSTEM -
pmaServer)
07/09/2005 19:06:29: (I) HTTP TORNADO Startup  (SYSTEM - HTTP)
```

The first server startup may be slightly slower than normal since the HSQLDB task will create the system database and tables, then populate this with the design data and login information for SysAdmin. Please check the startup screen carefully for any error messages (E).

At the server console you can execute a variety of commands. Type a ? then enter to get help. You can also 'tell' tasks to do things. eg, tell HTTP cache status

As you can see from the screen above, the configuration for your server is held in /puakma/config/puakma.config Edit this file with your favourite text editor to change your server's behaviour.

Next start your web browser and type the hostname (or localhost if you are on the same machine) into the browser's address bar. Enter the SysAdmin username and the password you entered during the install process. We recommend you create a new user and add them to the Admin group, and delete the default SysAdmin account.

People and groups, logs and server console may be administered from http://your_server/system/admin.pma
Web applications can be created and edited from http://your_server/system/webdesign.pma

If your server does not have a GUI console, the installation can be run on any machine with a GUI console, then the resulting files copied to the non-GUI machine.

The Administration Interface

The server may be controlled with a web browser. An administration application is included to make the process of managing and monitoring your Puakma server very easy. The administration application may be found here: http://your_server/system/admin.pma Remember that you will be prompted for a username and password. This will initially be sysadmin, with the password allocated during the install process.
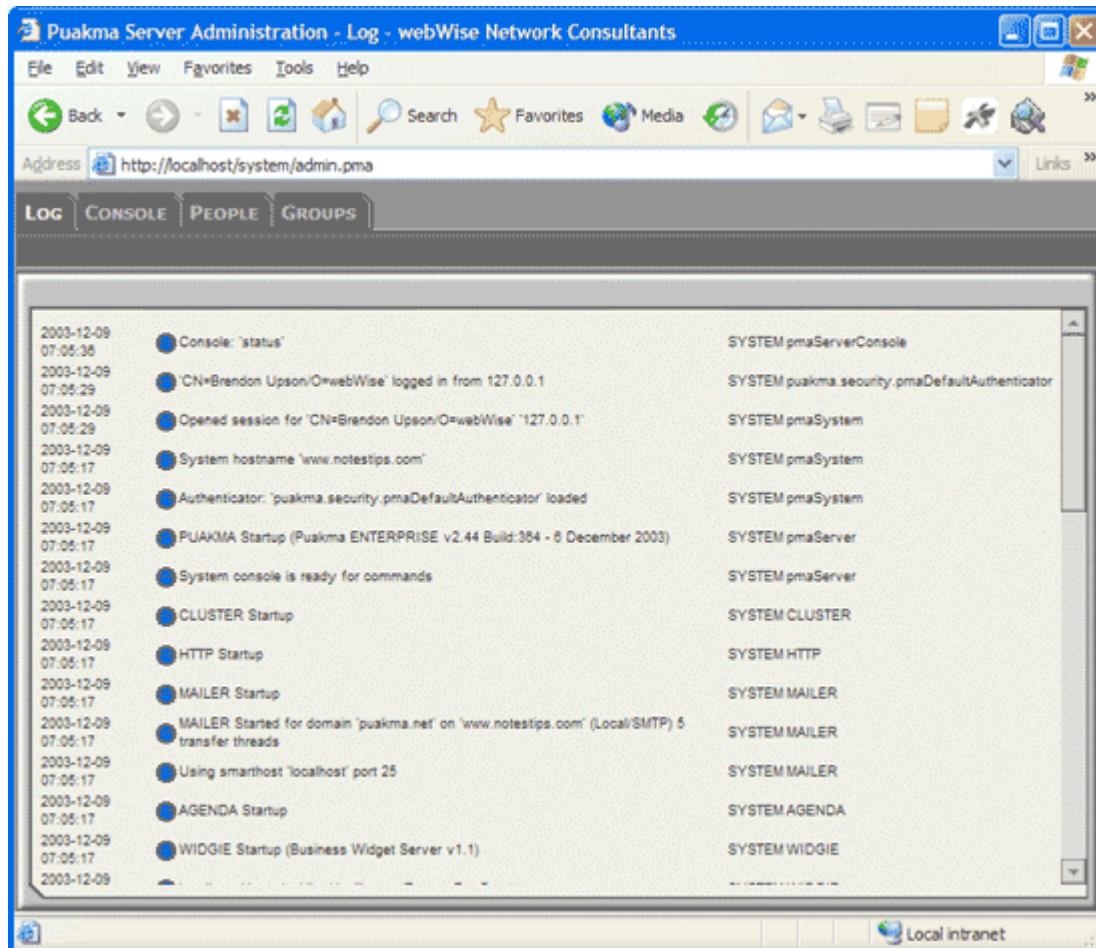
Informational messages are displayed with a blue spot, errors in red and debug messages in yellow. This makes it very easy to troubleshoot problems.

Remote access to the server console is also provided through the web browser interface.
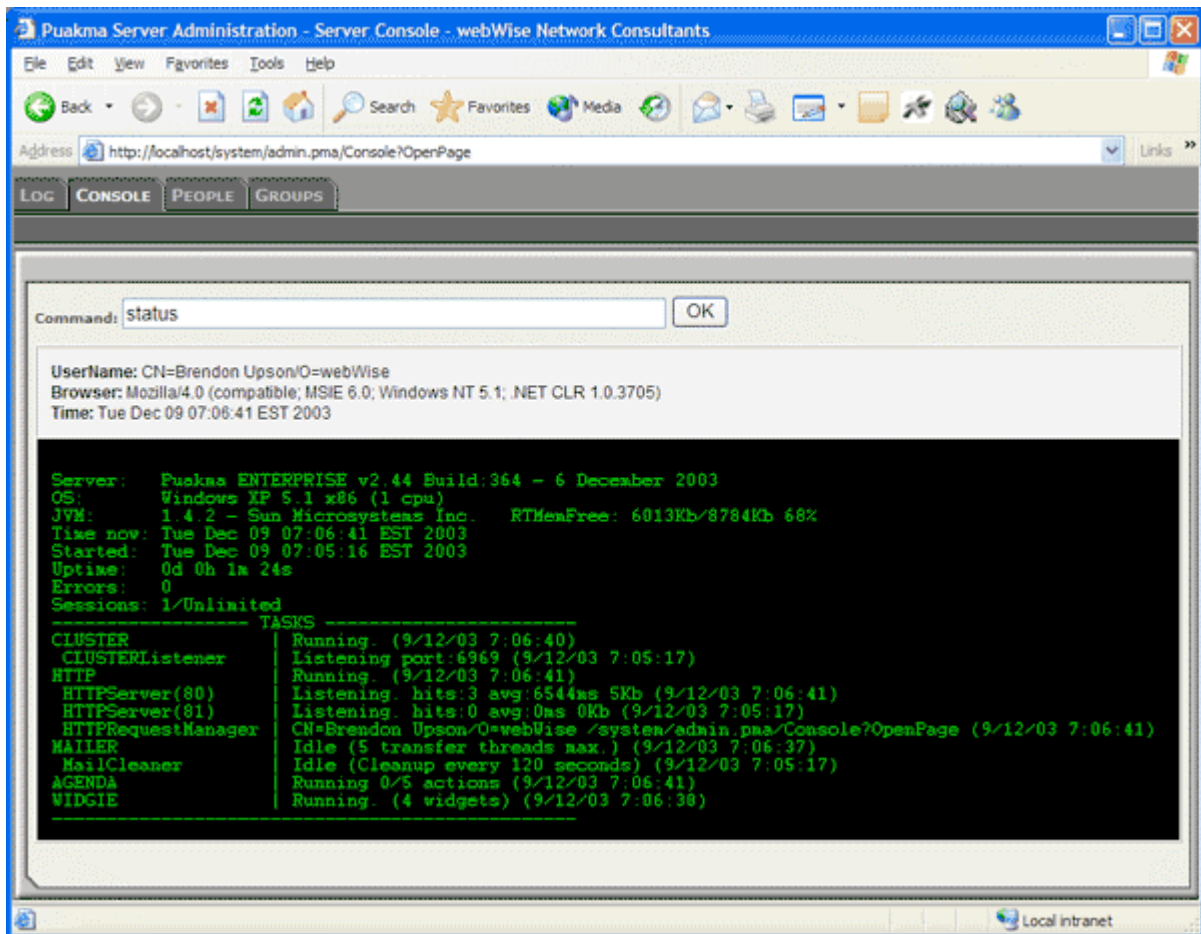
Puakma as a Service

Most modern operating systems support the running of programs as a daemon. Puakma may be run as a service, consult your platform documentation or the http://www.puakma.net website for operating system specific service information.

Multiple instances of the server may also be run on the same machine.

Anatomy of a Web Application

All application design data is stored in a relational database, as part of the "Puakma" system database. The main advantage of storing application design information in a database is the ability to store metadata along with it. Although many web platforms simply store files on the filesystem, storing them in a relational database creates the possibility to manipulate them programmatically and do simple yet powerful queries to update, interrogate or view many facets of the application, even create new elements dynamically.

Tornado Server web applications have the ".pma" (PuakMa Application) URL extention, eg: http://www.server.com/system/admin.pma . As the entire design is served from a relational database, the URL maybe somewhat confusing, because humans naturally associate a URL with a file path. In the case of Puakma, the URL is a reference to a set of design data stored in the database. A single server can hold many of these applications and run them all simultaneously.

Application specific data is stored separately to the design information. The database connection records within the application provide a simple way to reference the data storage area. It is incredibly easy to point your application at an existing data store, or to have multiple applications referencing the same datastore.

A Tornado Server application is a complete application package. It contains all design elements and security information required to move the application between server instances. The way the framework is designed allows for applications to be self-documenting, which make supporting a Puakma application much easier.
Developers use their web browsers to create and work with web applications. The web design application can be found at http://your_server/system/webdesign.pma All the source code is included with the webdesign application so developers can modify or extend the base version to include new features (for example "check-in/check-out")
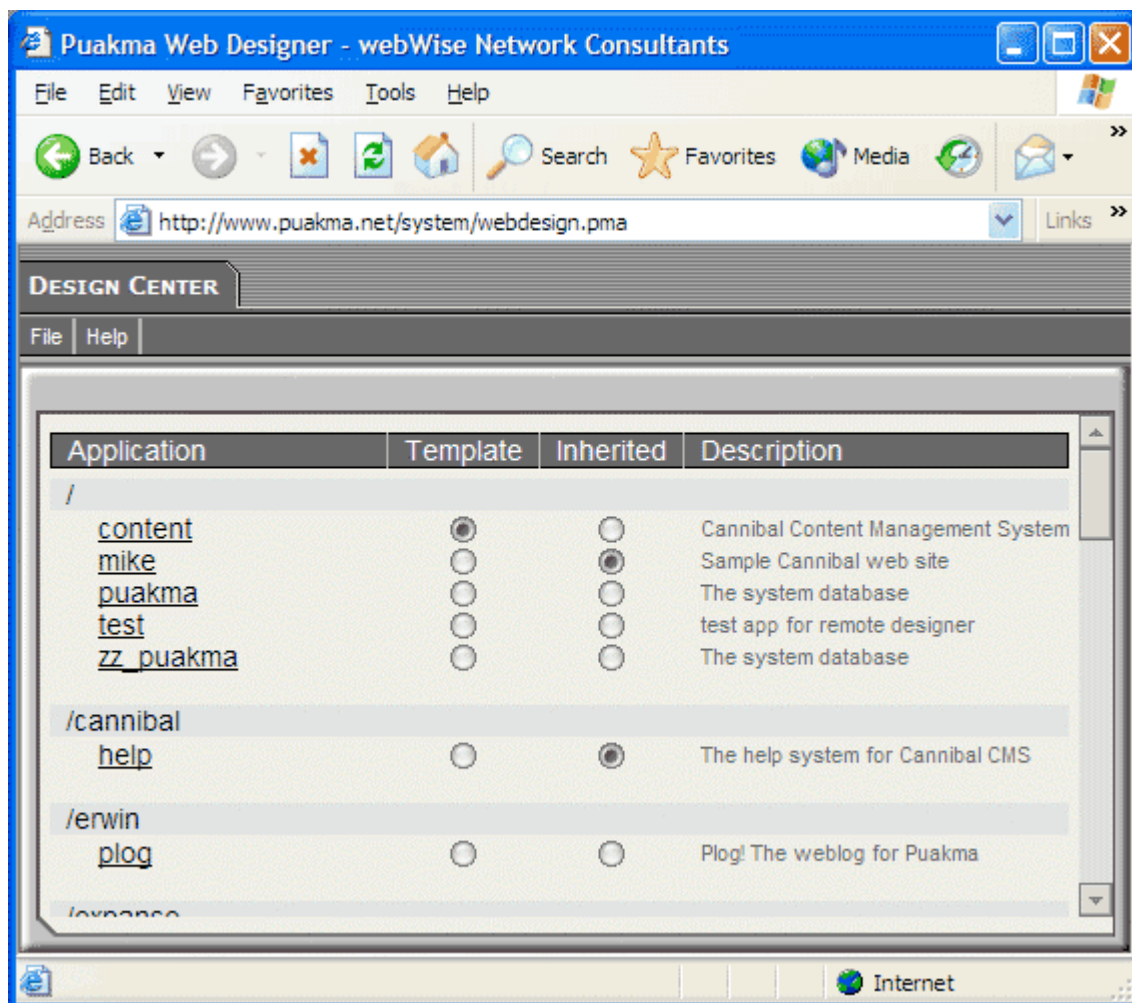
Web applications may be moved easily from server to server through by exporting them to a single ".pmx" file. A pmx file contains all the design data and design metadata to allow the application to be installed in another server instance.

Tornado Server clearly separates the User Interface (UI) from the logic. This makes applications more robust and less prone to being broken when graphic designers make slight alterations to the UI.

We recommend using the Puakma Vortex IDE for developing Tornado Server applications. Please see the website for more information.

Application Settings

The application settings section provides the basic information about the application, including the URL that will be used to reference the application. You can record comments about what the application's purpose is, it's default start page and any global open and save actions (see the Actions section for more information on global actions).

An application specific login page may also be defined here. This is automatically displayed when a user is requested to log in. If no specific login page is specified, the default login screen is displayed.
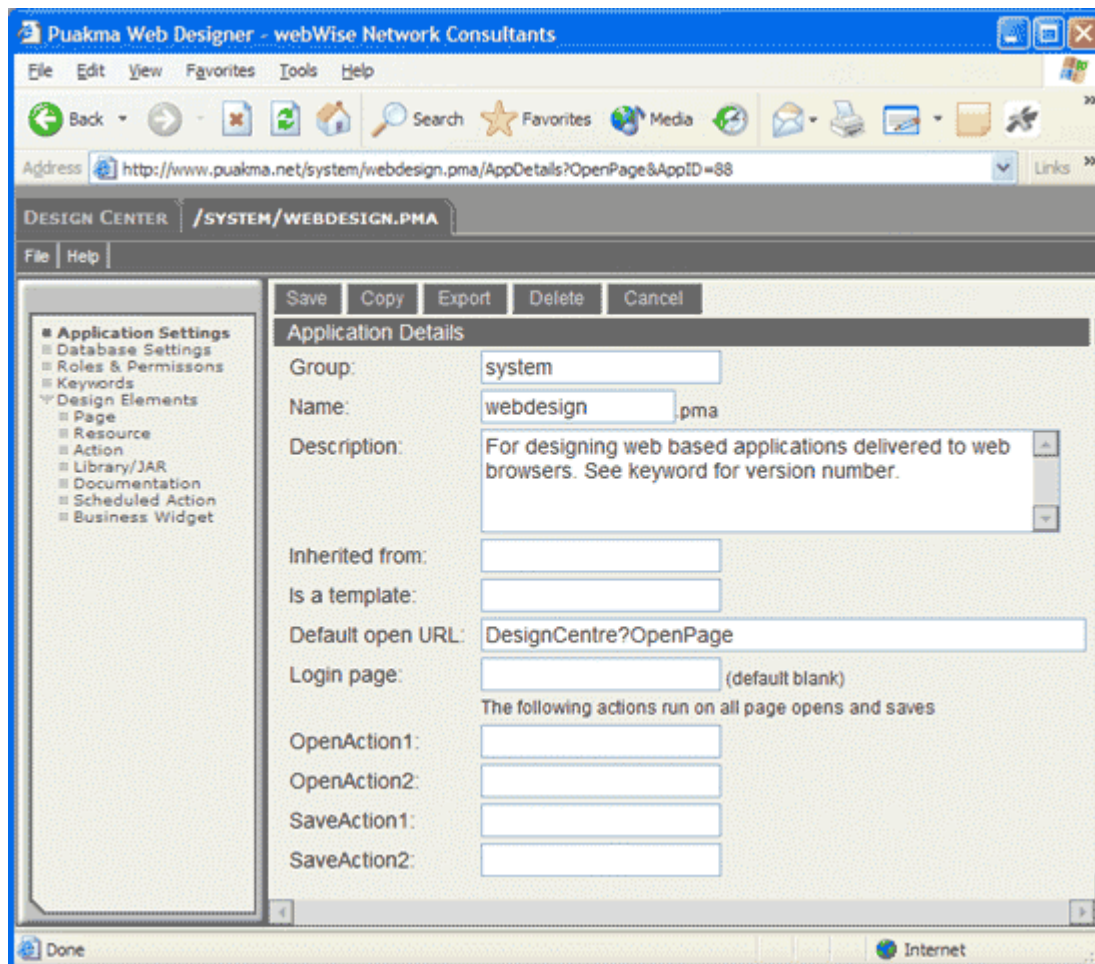
Database Settings

The database settings section is where the references to the application data are stored. This works much like a live data dictionary. Changes can be made quickly to the table definitions then reapplied to the database by clicking the "Build Table" button. Note that this causes the existing table to be dropped and recreated from the definition.

When creating a new application, developers may create a data connection, then table definitions beneath that connection, then attribute definitions beneath the table definition. When the data configuration is correct, simply click the "Create Database" button to build the database, tables and attributes.
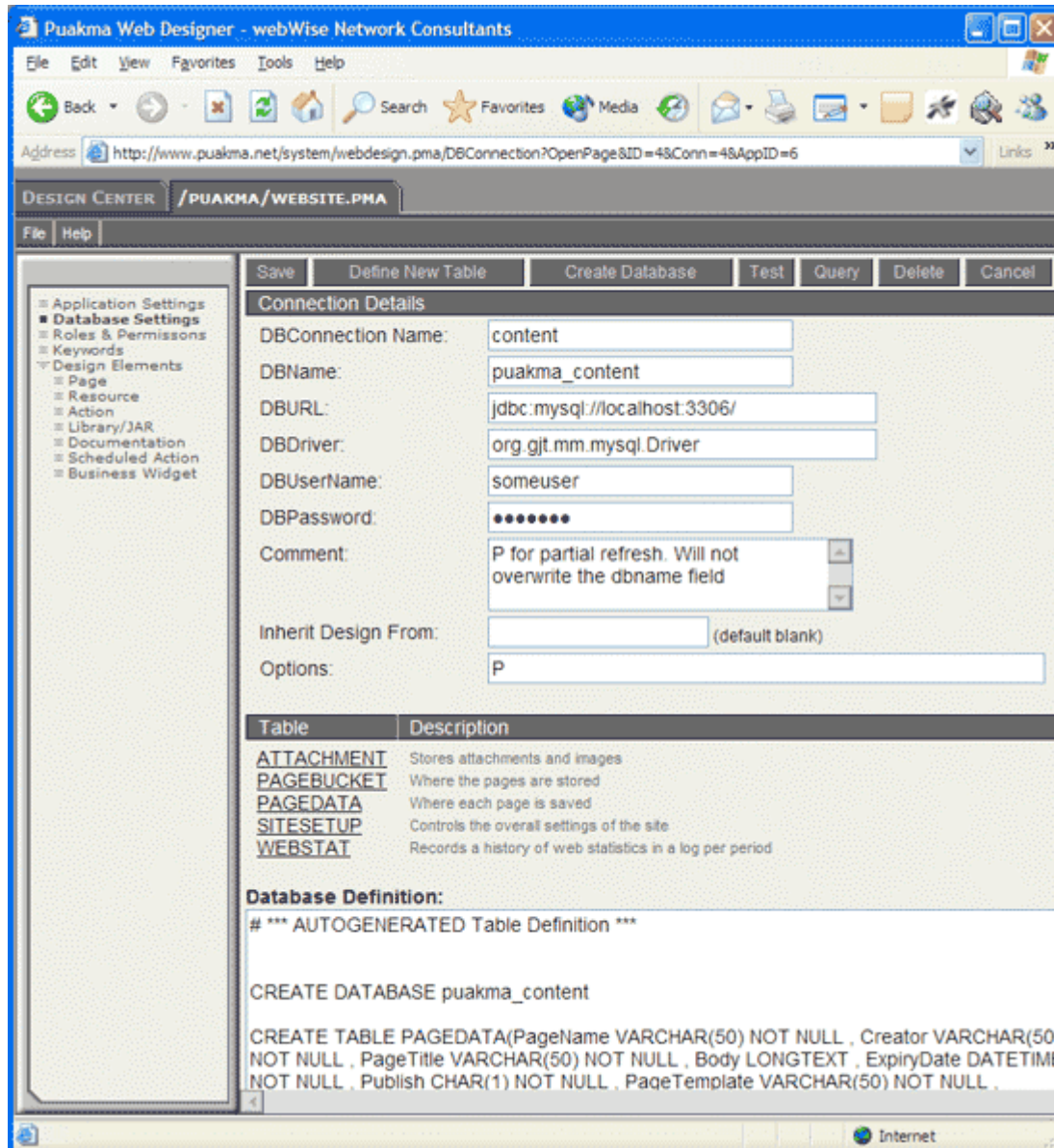
Security

The application server takes care of application security. By configuring roles in the web application the server is able to determine which users may access which resources. Checking of these roles is delegated to an "Authenticator" which allows security to be handled in the same way for all applications residing on the server. Developers may create their own customer authenticators to handle logging in against many different kinds of data sources. An application may handle its own security and login process, but it is recommended to delegate this to the authentication subsystem, since this makes the application more portable between environments.

Tornado Server uses a role-based security model. Developers create roles specific to the application and then associate individual names or groups to those roles. When a name or group is assigned a role, that is referred to as a permission.

There is a special role named "AllowAccess". This role is checked automatically by the server when an application user attempts to access a design element within the design collection. A setting of "*" allows all users basic access to the design elements of the application. When you create a new application, an "AllowAcces" and an "Admin" role are created automatically.

"WebServiceAccess" controls who may call web services.

"ResourceAccess" controls which users without the "AllowAccess" role can access resources. This is useful for applications that don't allow anonymous access but have their own internal login page (that references resources in the same application).

Permissions may be specified in a number of ways:
- A person's full X500 name, eg: "John Smith/Sales/AcmeCorp"
- A group name, eg: "All Sales Staff"
- A wildcard name, eg "*/Sales/AcmeCorp"
- * meaning anyone including anonymous users
- !* anyone except anonymous users (users must be authenticated)

In addition, multiple permissions may be assigned to each role. Permissions are not granular, if a user's name matches one of the permissions, they have the role. It is then up to the developer's code to determine what may be done based on that permission. To check roles in the Action java code:

```
if(pSession.hasRole("DocumentCreator"))
{
        //perform the specific logic
}
```

A useful way of handling security is to utilize a "Global Action" that does role and permission checking in a central spot, outside of the main application logic. The advantage of this approach is that all security rules are in one place and all design elements are catered for. This also means the developer can code the security rules after the bulk of the application has been developed. Following is a sample security Global Action.

```
import puakma.system.*;

/**
 * Called for every page automatically to ensure the correct security is
applied.
 */
public class GlobalSecurity extends ActionRunner
{


   public String execute()
   {
      String LOGIN_PAGE="Login";

      //If anonymous user, plug to the Login Page.
      if(pSession.getUserName().equals(pmaSession.ANONYMOUS_USER))
      {
            if(ActionDocument.designObject.getDesignName().equalsIgnoreCase
      (LOGIN_PAGE)) return "";

      return ActionDocument.rPath.getFullPath() + "&login";
      }

      RequestPath rpPage = ActionDocument.rPath;
      String sPage = ActionDocument.rPath.szDesignElement;

      //admin has access to everything
      if(pSession.hasUserRole("Admin")) return "";


      //if we get to here, we're not an admin so check each page
      if(sPage.equalsIgnoreCase("Administration"))
      {
            return getDBURL()+ "/Unavailable?OpenPage";
      }

      //add more page and role checks here

      return "";
   }
}//end class
```

Keywords

Keywords are a storage area for application specific data that changes infrequently. Common uses for key words are Combo-box values, or specific locations or settings used in the code.

Keywords consist of a keyword item, which is a name that developers use to access the keyword, and a set of one or more data items.

There are two ways to access keywords, either as a part of a page element tag:
```
<P@List name="Status" value="" useKeyword="DocumentStatus" @P>
```
or from within the Action code:
```
Vector vList = pSession.getAllKeywordValues("DocumentStatus", true);
String sFirstValue = pSession.getKeywordValue("DocumentStatus");
```

Pages

Pages are usually html in nature. Pages bind together the UI components and provide an easy means of changing the "look" of an application. Typically pages contain <P@ tags. These tags allow the Actions to insert data dynamically into a web page. This gives pages the ability to interact with Actions and consequently, the database. Pages may also be text/xml, text/xsl, text/plain, or actually any mime-type at all.

A page may have an OpenAction and a SaveAction parameter set in the webdesigner. The OpenAction parameter defines what action is run when a pages is opened, while the SaveAction defines what Action is run when the page is submitted to the server. These parameters are optional.

We recommend using the "Parent Page" functionality discussed later.

Custom Login Pages

Puakma supports a custom login page per application. If no page custom login page is specified, then the default login page ($Login) from within /puakma.pma will be displayed.

The Application Settings section defines the name of the custom login page for the application. This page name must exist as a Page within the application.

A login page must have four specific elements to function correctly:
1. "$LoginPage" set to "1" to tell Puakma this is to be used for login
2. "Username" field with the user's name
3. "Password" field with the user's password
4. "$RedirectTo" field containing the URI of the page to be displayed after a successful login. Note that this field should be of type `<@Hidden name="$RedirectTo" value="" @P>` The server will automatically populate this field with the correct path when the login page is presented.

There are more advanced methods of providing custom login processes, since Actions have almost full system access. This is an advanced topic and will be covered by a later BlackBook.

Custom Error Pages

When a user requests an element from an application that does not exist in the design collection, Puakma will attempt to locate a Page named "404" to serve to the user. A "403" Page may also be served if the user does not have access to the element they are requesting.

Resources

A resource is a static file type, such as an image that makes up part of the application. Typically, CSS stylesheets, JavaScript, XSL stylesheets and images are attached as resources. A resource may be any file type.

Actions

Actions perform the logic or data processing of the application. Actions may be invoked in three ways:
1. Directly from a URL, eg: http://servername/group/app.pma/ActionName?OpenAction
2. When a particular page is opened
3. When a particular page is submitted

Actions are simple Java classes that extend ActionRunner. A simple Action that prints "Hello, world!" to the browser follows:

```
import puakma.system.*;
```

```
/**
 * A simple Puakma action template.
 */
public class SimpleAction extends ActionRunner
{
    public String execute()
    {
        this.writeln("Hello, world!");
        return ""; //no redirect!
    }
}
```

Global Actions

Global Actions are defined as part of the Application Settings section. A global OpenAction is run every time a Page or Action is called. There may be multiple Global Actions which are run sequentially. The first action to output some value halts the chain processing the response from that Action is sent to the browser.

Global Actions are great for either prefilling fields on a Page (such as menu items, user names and the like) or controlling complex security. By placing all your application access logic in one place it makes it very easy to control who has access to which portions of the application without peppering the application logic with security checks.

Scheduled Actions

Scheduled Actions have identical code to a regular Action, but they are not able to output to a browser. Scheduled Actions are run by the AGENDA server task at the time defined by the Scheduled Action's settings in the webdesign application.
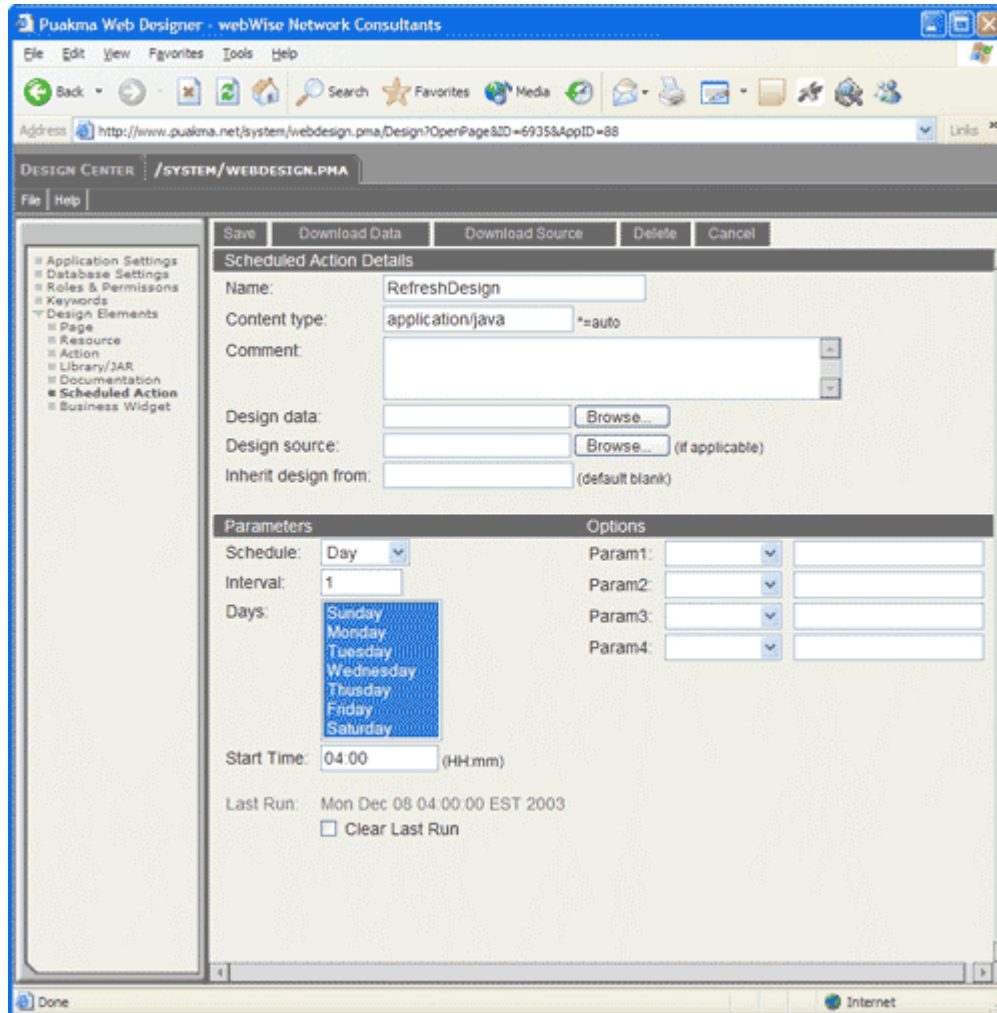
A great way to test a scheduled action during development is to save it into the design collection as a regular action. This way the developer needs only to open the URL for the action for the action logic to be executed.

Libraries

Libraries may be either single java class files or multiple class files in a jar. Libraries provide a way to share reusable code amongst your actions and Business Widgets.

To create a library:
1. Create and compile a new Java class
2. Open the libraries section of the webdesign application and click the "New Library" button
3. Give the library a name
4. Attach the binary class file and optionally the source

All libraries are automatically loaded when an action is executed for the first time.

Documentation

The documentation section is not accessible from any URL, it may only be viewed through the webdesign application. This provides an area within the application to attach notes and information to other programmers that may help with the maintenance of the application. Documentation may be any type of file (MS-Word, Text file, PDF, images).

We have taken great care to ensure a Puakma web application is a self-sustaining container of both application design and documentation.

Business Widgets

Puakma web services are based on SOAP and required the Enterprise Edition to be installed in order to run (the require the WIDGIE server task). Puakma web services are possibly the easiest way of creating and publishing web services. To create a Business Widget web service, create a Java class that extends **BusinessWidget**.

```java
import puakma.addin.widgie.*;

/**
* A sample web service
*/
public class HelloWorld extends BusinessWidget
{

    public String sayHello()
    {
      return "Hello, world!";
    }

}
```

Once the code has been compiled and attached as a Business Widget, it will be automatically loaded and mounted within the application, ready for use. There are no complicated WSDL interfaces to write or configure. Business Widgets are run in their own thread so may also store their own data that persists between calls to the service.

PMX files for moving applications between servers

Puakma provides a simple mechanism for moving web applications between servers. This may be useful when an application is moved from the development server to the production server, or simply to send to customers to evaluate.

A PMX file (PuakMa eXport) is a compressed XML file that contains the application design information flattened into a single file. PMX files are generally very small (<500Kb) and easy to email or provide as a download from a website.

To create a pmx file of an existing application:
1. Open the webdesign application http://your_server/system/webdesign.pma
2. Click on the application to export
3. From designer's menu, select File > Export
4. You will be prompted to save a file. Save the file to your local disk. This file is the pmx.
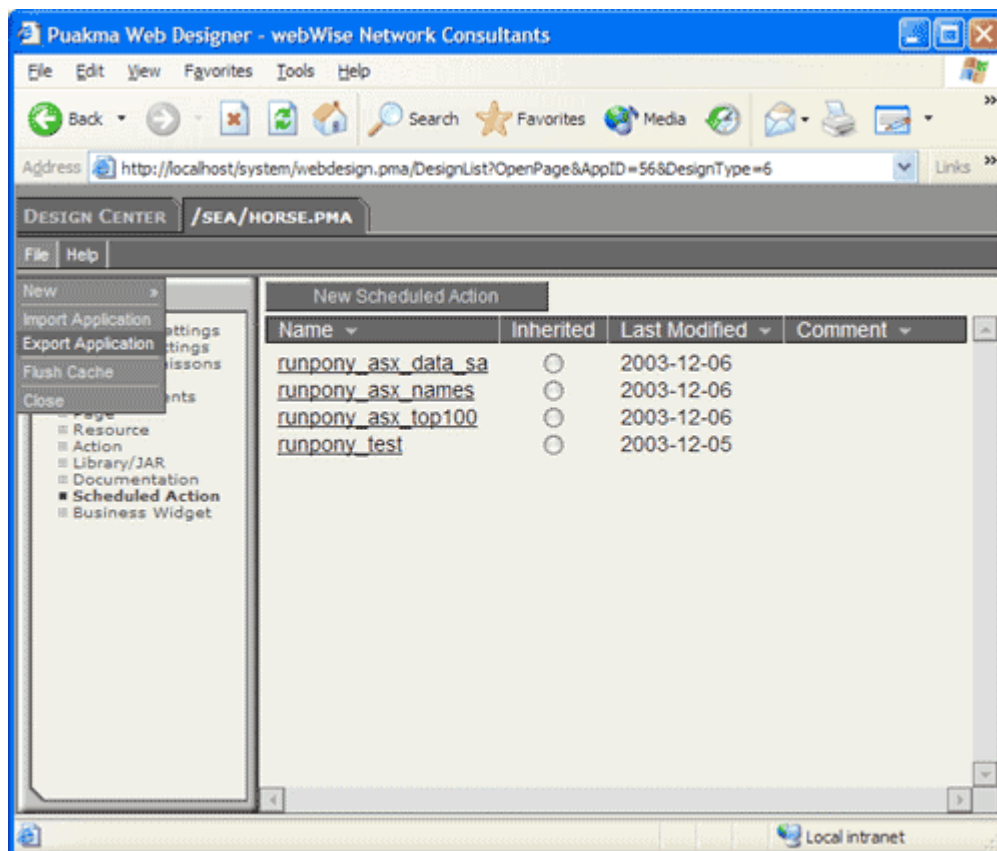
Page Elements using <P@ tags @P>

Puakma specific HTML Page tags

Text	text	`<P@Text name="FieldName" value="FieldValue" ... @P>`
Hidden	?	`<P@Hidden name="FieldName" value="FieldValue" ... @P>`
Checkbox	option1 option2	`<P@Checkbox name="FieldName" cols="1" Choices="option1|1,option2|2,option3|3" Value="FieldValue" ... @P>`
Radiobutton	option1 option2	`<P@Radio Name="FieldName" Cols="1" Choices="radio1|1,radio2|2,radio3|3" value="FieldValue" ... @P>`
Listbox	option1 option2 option3	`<P@List Name="FieldName" Choices="option1|1,option2|2,option3|3" Value="FieldValue" Size="10" MULTIPLE ... @P>`
Combobox	option2	`<P@List Name="FieldName" Choices="option1|1,option2|2,option3|3" Value="FieldValue" ... @P>` Effectively this is the same as a listbox tag, but has no size attribute. You can also include a usekeyword="yourkeyword" tag. This will populate the list with your keyword values created in the webdesign app.
Textarea	Sample...	`<P@textarea Name="FieldName" Value="FieldValue" Cols="40" Rows="4" ... @P>`
Sub-Page	N/A	`<P@Page PageName @P>` Includes the entire contents of another page in the current page. These may be nested 3 deep. This is more efficient than a computed page, but cannot be changed at runtime.
Computed Sub-Page	N/A	`<P@ComputedPage Name="FieldName" Value="PageName" @P>` Includes the page named in the value parameter at that position in the current page. This provides the ability to control what is shown programmatically by setting the value of the item in an OpenAction.
Computed	Plain text	`<P@Computed Name="FieldName" Value="FieldValue" @P>` A computed field may be accessed by an OpenAction action and populated with plain text or formatted HTML.
Path	/puakma.pma	`<P@Path @P>` This is a handy tag useful for making application portable tags. For example you might use this code to reference an image, eg `<img src="<P@Path@P>/myimg.gif?OpenResource">`. The path tag supports a value attribute to return only a part of the URI. For example for the URI "/apps/crm.pma" the tag `<P@Path value="group" @P>` will return "apps". Possible values are: Group, Application, Action, DesignElementName, Parameters, FileExt
File		`<P@File Name="FieldName" ... @P>`
Password	*****	`<P@Password Name="FieldName" Value="FieldValue" ... @P>`

Button		`<P@Button Name="Name" Value="label" ... @P>`
Form Tag	`<form name="page" method="POST">`	`<P@Form @P>` Inserts the system generated form tag at the location this tag appears.
Version String	`<!-- Puakma v2.32 Build:303 - 30 October 2003 -->`	`<P@Version @P>` Inserts a html comment in the source denoting the Puakma server version.
Page Parameter	-	`<P@Parameter Name="ParamName" Value="">` When you pass a parameter to a page, allows you to insert that param in the page WITHOUT writing any page action. ie: if the page is opened with the url "http://server/group/app.pma&fred=17" everywhere in the page this tag appears will show the string "17"
Cookie	-	`<P@Cookie Name="CookieName" Value="">` When the browser sends a cookie up to the server, everywhere on the page that this tag appears will be replaced with the value of the cookie.
Date	31/01/2002	`<P@Date Name="FieldName" Value="FieldValue" Format="dd/MM/yyyy" ... @P>` This will format a date object into the corresponding String value. size parameter is optional as it will use the length of the format string. Possible values are: now, today, yesterday, tomorrow, nextweek, lastweek, nextmonth, lastmonth, nextyear, last year.
ComputedDate	31/01/2002	`<P@ComputedDate Name="FieldName" Value="FieldValue" Format="dd/MM/yyyy" ... @P>` This will format a date object into the corresponding String value. This will ensure the date is always in read mode. Possible values are: now, today, yesterday, tomorrow, nextweek, lastweek, nextmonth, lastmonth, nextyear, last year.
View	N/A	`<P@View Name="ViewName" datasource="SELECT * FROM SOMETABLE ORDER BY SomeColumn" xslstylesheet="browse.xsl" connection="conn_name" nexttext="Next" previoustext="Previous" maxperpage="25" ... @P>` This will render a list of rows from the relational database. The last 4 parameters are optional.
ChildPage	N/A	`<P@ChildPage @P>` Renders the child page into this location. Used for "templating" functionality to simplify page designs and allow complete application rebranding very easily. See below for more information.

Note: With most fields you can also include a `UseKeyword="yourkeyword"` parameter. This will populate the list with your keyword values created through the Keywords option in the webdesign app.

Combo-boxes, list boxes, radio buttons and check-boxes all have the ability to get their choices from a connection in the application. Add `datasource="SELECT Name,': ',Age,ID FROM PERSON ORDER BY Name"` and `connection="DataConnectionName"`. This will execute the select query against the named connection and populate the choices of the field with the contents of the database. You may also add `firstchoice="- Select an Option -|"` to make the first choice in a list. This follows the same convention as the View tag.

Multiple selection tags also support an additional option of `missingvaluetext="The value is not in the list"` . This tag option allows for the field's value to be added as an option to the list. This is useful in cases where you have a list of choices of, for example, active companies and the record you are editing refers to an inactive company. Ordinarily with HTML, a combo box will default to the first value in the list where no matches are made. Adding this option to a p-tag enables the dynamic checking and adding of the value to the

choicelist. Setting the option value to an empty string will use the value of the item as both the choice and the value.

Puakma p-tags must appear on the same line, a p-tag cannot be broken over two lines.

Parent/Child Pages

Tornado server provides the ability to put the majority of the page's markup and layout in a "parent page". The idea is that a site will have few parent pages and many child pages. This allows the bulk of the layout and markup to be included on the parent page and only the specific markup added to the child. This allows complete rebranding of an application by altering only the parent page(s).

To enable this functionality, create a page in the design collection ensure that the page includes one `<P@ChildPage @P>` tag. This is the spot where the child page will be placed. Next create the child page with the html fields and markup as required. Ensure that the page parameters specify the ParentPage attribute is set to the name of the parent page.

SimpleParentLayout
```
<html>
<head><title><P@Computed     name="title"     value="Your     Tornado     App"
@P></title></head>
<body>
<img src="<P@Path @P>/yourlogo.jpg"><br/>
<P@ChildPage @P>
</body>
</html>
```

ChildPage
```
<P@Form @P>
Name: <P@Text name="Name" @P><br/>
Age: <P@Text name="Age" size="5" @P><br/>
<P@Button name="btnOK" value="Submit" onclick="document.forms[0].submit();"
@P>
</form>
```

Ensure the ChildPage design element has the following parameter set:

When a web user requests the child page http://yourserver/app.pma/ChildPage?OpenPage The child page will be automatically inserted into the parent.

Chain Of Events For HTTP Requests

Each HTTP request to GET a page follows this process:
1. The request enters the server and the http headers are parsed
2. The server determines the user's identity based on the session cookie
3. User security is checked against the application and design elements requested
4. A "document" is created in memory describing the data sent by the client
5. The Page design element is parsed and new items (fields) are created on the document where defined by the Page's `<P@` tags. Page is inserted in its parent where appropriate.
6. Each Global OpenAction is run against the Page, updating the document where applicable
7. The specific OpenAction for the Page is run, further updating the in-memory document
8. The data on the document is rendered with the design of the Page and streamed to the client.

Inheriting Design Elements

Puakma supports design templates. This means developers may have multiple instances of an application on a Puakma server, with each instance set to get its design from a template. Each morning at 4am, the

RefreshDesign Scheduled Action (inside /system/webdesign.pma) checks the template against the applications that inherit from it and make the appropriate changes, such as adding new elements or updating actions that have changed.

An example of where this may be useful is if your server contains five discussion forum applications all sharing an identical design. By updating the master template, all discussion forums will have their design altered to match the master template the next time the RefreshDesign scheduled action is run.

URL Design

A Tornado Server URL is always described in the following form:
http://servername/appgroup/apname.pma/DesignElement?ActionType&Parameter=paramvalue

Valid ActionTypes are as follows:

?OpenPage	Display the named page from the database
?ReadPage	Display the named page from the database, but do not allow the fields to be edited
?OpenAction	Run the named action
?OpenResource	Get the named resource
?WidgetWSDL	Show the WSDL for the named Business Widget (Enterprise Edition web services, SOAP). XML output
?WidgetExecute	Execute the named Business Widget (Enterprise Edition web services, SOAP). Only from a client HTTP POST. XML output
?WidgetList	Displays the available widgets in an application (Enterprise Edition web services, SOAP). XML output

Puakma also supports **&login** and **&logout** parameters to force a user to login or out.

The design of the URL is important to improve the maintainability of web applications. Developers can look at the URL and easily determine where in the application the problem might be.

Advanced Authentication Options

The Puakma server takes care of all security, leaving the developer free to perform the occasional role check. Authentication to the server is managed by one or more "Authenticators" which are configured in Puakma.config. Developers may write their own Authenticators and install them at the server level. All applications on the server use the same authentication chain.

The authentication chain is the list of Authenticators in a specific order used to log users in to the server. Puakma ships with the default authenticator which checks usernames and passwords against a relational database. Also on the http://www.puakma.net website there are some additional authenticators, such as LDAP. This means that if a legacy system such as Domino is used, the user information may be utilized. There is also code available to authenticate against Netware NDS.

Session Authentication

Each user of the server has a Session object that describes their session. This is available within Actions as the object "pSession". The session object is very powerful, containing many methods for work with or determining user credentials. Additionally, ad hoc objects may be stored on the session object providing a volatile storage area for data that needs to persist for the duration of the session.

The session is maintained in the web browser using a session token. When a user first connects to the Puakma server, a token is allocated which the browser sends to the server with each subsequent request. Web browsers must support cookies in order to use Puakma.

Tornado Server supports the `Authorization:` http header. This allows applications such as web services to authenticate with the server with their credentials.

Sessions exist on the server until there has been no activity for the specified server-wide session timeout interval.

A Tornado server generates Ltpa tokens that may also be used for single sign-on against Lotus Domino servers.

Utility Classes

The Puakma.jar file contains many useful classes and methods to speed development time.

puakma.addin.http.document.TableManager

TableManager is designed to provide an abstraction layer between the relational database server and the action. It is used for getting data from the database, XML conversion of resultsets, and insertion and updating of data.

Insert a row into the TRAININGDATA table:
```
TableManager t = new TableManager(pSystem, pSession, "YourConnectionName",
"TRAININGDATA");
t.setField("TrainingCreator", pSession.getUserName());
//save the record
t.insertRow();
```
or
```
t.updateRow("WHERE ID=45");
```

Displaying existing data onto a Page:
```
TableManager t = new TableManager(pSystem, pSession, "YourConnectionName",
"PERSON");
t.populateDocument("SELECT * FROM PERSON WHERE EMPLOYEEID=" + sID,
ActionDocument);
```

Server-side XML/XSL transforms:
```
TableManager t = new TableManager(pSystem, pSession, "YourConnectionName",
"PERSON");
StringBuffer sb = t.getXML("SELECT * FROM PERSON LIMIT 100");
StringBuffer sbResult = pSession.xmlTransform(sb, "personlist.xsl");
if(sbResult!=null) ActionDocument.setItemValue("PersonList",
sbResult.toString());
```


TableManager can also be used to populate the dynamic choices of ListBoxes and ComboBoxes, using very little programmer written code eg:
```
TableManager s = new TableManager(pSystem, pSession, "YourConnectionName",
"SITE");
String szSite[] = s.makeChoicesArray("SELECT SiteName, SiteID from SITE",
"- Select a site -|");
ActionDocument.setItemChoices("SiteList", szSite);
```

puakma.addin.http.document.HTMLView

HTMLView is a generic class for getting data from the database, rendering it with an XSL stylesheet and inserting it in a field in a page. Internally, HTMLView returns a StringBuffer of XML, which is then rendered with the named XSL stylesheet into a computed field on your page. This class sits behind the <P@View tag.

You may provide a null parameter for the dataconnection, next button and previous button. In this case the first available data connection will be used and the next and previous button text will be drawn from the language file (typically msg_en.lang in the /Puakma/config/ directory)

If the XSL stylesheet named in the parameter does not exist as a resource in the design collection, one will be automatically created based on the columns in the XML returned by the query. If no XML data is returned by the query an XSL stylesheet will not be created. Note: BLOB fields from the database are not included in the XML output.

The following code snippet may be copied directly into your Actions.

On the page, add a computed field:
```
<P@Computed name="LogData" @P>
```

Then render the view data into that field:
```
HTMLView hView = new HTMLView(ActionDocument, "logView.xsl", "SELECT * FROM
PMALOG ORDER BY LogDate DESC", "ConnectionName", "Next &gt;", "&lt;
Previous");
hView.setNavigationStyle(HTMLView.NAVIGATION_STYLE_ALPHA);//shows abcd etc
across the top of the page or 0
hView.setRowsPerView(100);
hView.setDocumentViewHTML("LogData");
```

puakma.util.Util

The Util class is full of static utility methods that alleviate common programming chores, such as trimming Strings of excess spaces, adjusting Dates, base64 encoding, gzipping, etc eg:

```
java.util.Date dtYesterday = puakma.util.Util.adjustDate(new
java.util.Date(), 0, 0, -1, 0, 0, 0);
```

puakma.util.RunProgram

This class is designed to simplify the running of an executable program (platform native binary code). eg:

```
RunProgram rp = new RunProgram();
String sCommandLine[] = new String[]{"javac"};
int iReturnVal = rp.execute(sCommandLine, sEnvironmentArray,
fileWorkingDir);
if(iReturnVal!=0)
{
    System.out.println("ERROR: "+new String(rp.getCommandErrorOutput()));
}
else
{
    System.out.println("SUCCESS: "+new String(rp.getCommandOutput()));
}
```

The current thread will wait for program execution to complete. This does not support running programs as a daemon. Programmers will need to wrap this class in their own thread wrapper if required.

puakma.system.X500Name

This class is designed to allow manipulation of X500Names. For example, the name "CN=SysAdmin/OU=Dept/O=Org" can be simply converted to its component parts.

```
X500Name nmUser = new X500Name(pSession.getUserName);
nmUser.getCommonName(); // SysAdmin
nmUser.getAbbreviatedName(); // SysAdmin/Dept/Org
nmUser.getCanonicalName(); // CN=SysAdmin/OU=Dept/O=Org
```

Class Loaders

Tornado Server has two class loaders. The first is created when the server is started and its main job is to simplify the classpath. It actually serves two purposes, first files need only be copied to directories to be available to the class loader (eg /lib/) and secondly it allows new code to be loaded on a server restart without exiting the JVM. Allowing new code to be loaded on a server restart means that overall downtime due to an upgrade is minimized since the administrator can simply copy and prepare all files, then restart the server. The second classloader (SharedActionClassLoader) is responsible for retrieving application specific class data from the database.

SharedActionClassLoader

This classloader performs the bulk of the work. There is one of these classloaders created for each .pma application and one for Actions, Shared Actions and Business Widgets.

When an action is run or widget loaded, the class loader must determine the real class name of the action as it may be different to its name in the design collection. Next it must determine what libraries are to be used with this action (AlwaysLoad or action specific UseLibrary). All these items are added to an internal cache so the action is typically slow to load the first time it is run, then loads very quickly afterward.

The classloader's cache must be purged and the classloader dropped when a design element is changed. This is to ensure subsequent actions correctly pick up the new code. This also means that any stored session objects (pSession.addSessionObject();) must be removed because the server is unable to guarantee that the object stored on the session is still valid or that a library it relies on is available. Further, when sessions are saved to disk on a server restart (eg "restart server save" or "quit save" at the console), any objects loaded with this classloader are discarded.

This classloader should not be used directly by the programmer, we have included information here merely to give developers an understanding of its operation.

Code samples

Working with file uploads

Getting a handle to a file uploaded by a user is handled in the following way. First, create a page with a file upload control, developers can use the inbuilt Tornado Server p-tags or native html:

```
<P@Form @P>
<P@File name="FileData" @P>
<input type="submit" name="btnSubmit" value="Upload file" />
</form>
```

Next create a save action for the page:

```
TableManager t = new TableManager(pSystem, pSession, "YourConnectionName",
"ATTACHMENT_TABLE_NAME");
String sAttach = ActionDocument.getItemValue("FileData"); //get filename
String sMimeType = "";
if(sAttach!=null && sAttach.length()>0)
{
      int iIndex = sAttach.lastIndexOf('/');  //Check if Unix Style
      if (iIndex == -1 ) //Not found
      {
            iIndex = sAttach.lastIndexOf('\\'); //Check if Windows style
      }
      //Extract the last substring, leaving just the filename
      sAttach = sAttach.substring(iIndex+1);
      //get a handle to the file item
      DocumentFileItem dfi = (DocumentFileItem)
ActionDocument.getItem("FileData");
      if(dfi!=null)
      {
            sMimeType = dfi.getMimeType();
            byte btData[] = dfi.getValue();
            if(btData!=null && btData.length>0)
            {
                  //load TableManager with the data
                  t.setField("MimeType", sMimeType);
                  t.setField("FileData", btData);
                  t.setField("FileName", sAttach);
            }

      }//dfi!=null
}
//save the data to the database
t.insertRow();
```

POSTing other types of data

From time to time the server will need to accept other types of data rather than the standard mime encoded or form encoded types from a standard web browser submission. An example may be "text/plain". When one of these content types is POSTed, the server will place all the uploaded data into a single document item called "Data". This item will either be a standard DocumentItem of DocumentFileItem depending on the size of the data POSTed. Data greater than 512Kb will result in a DocumentFileItem being created. Following is an example of how to process the uploaded data from a SaveAction.

```
DocumentItem di = ActionDocument.getItem("Data");
if(di.getType()==DocumentItem.ITEM_TYPE_FILE)
{
      DocumentFileItem dfi = (DocumentFileItem)di; //cast as
DocumentFileItem
      String sType = dfi.getMimeType();
      File f = (File)dfi.getObject();
}
else
{
      byte buf[] = di.getValue();
}
```

Changing the default page

It is possible to display a completely different page to that which the user requested. The following code swaps the page design with that of "ReadersOnly".

```
DesignElement    deAct    =    pSession.getDesignObject("ReadersOnly",
DesignElement.DESIGN_TYPE_PAGE);
if(deAct!=null)
{
      ActionDocument.designObject = deAct;
      ActionDocument.prepare(null);
}
```

The following code retrieves the "NewPage" page from the design collection, prepares (breaks up and parses the p-tags) and renders it (turns the p-tags back into valid HTML).

```
DesignElement    deAct    =    pSession.getDesignObject("NewPage",
DesignElement.DESIGN_TYPE_PAGE);
if(deAct!=null)
{
      try
      {
            HTMLDocument docTemp = (HTMLDocument)ActionDocument.clone();
            docTemp.setContent((byte[])null);
            docTemp.setContent((StringBuffer)null);
            docTemp.designObject = deAct;
            docTemp.prepare(null);
            docTemp.renderDocument(false, false);
            byte buf[] = docTemp.getContent();
            if(buf!=null)
            {
                this.setBuffer(buf);
                this.setContentType(deAct.getContentType());
            }
      }
      catch(Exception e)
      {
```

```
            pSession.getSystemContext().doError("setupReturnHTML():    Error
cloning current document: " + e.toString(), arAction);
     }
}
```

Changing field types

One of the major advantages of p-tags is the ability to change them in an action. For example, under certain conditions perhaps a list box should be removed from the page, or replaced with computed text.

In this example a listbox of project managers is replaced with some plain text:
```
<P@List name="ProjectManagerList" size="10" @P>

HTMLDocumentItem hdi =
ActionDocument.getHTMLDocumentItem("ProjectManagerList");
hdi.setType(HTMLDocumentItem.ITEM_TYPE_COMPUTED);
ActionDocument.setItemValue("ProjectManagerList ", "ALL PROJECT MANAGERS
ARE ASSIGNED");
```

Note: This technique illustrated here does not work with dynamically created fields. For dynamically created fields, the programmer must keep a handle to the dynamic object and manipulate that object.

Creating fields dynamically

In some cases (particularly combo-boxes, list-boxes) programmers need an easy way to create many identical fields, with perhaps the only different being the field name.

This example inserts 30 combo boxes with a slightly different name to the computed field "DynamicData"
```
<P@Computed name="DynamicData" @P>

HTMLDocumentItem   hdi  =  new  HTMLDocumentItem(ActionDocument,  "<P@List
name=\"ListFieldName\" @P>");
String sChoices[] = new String[]{"item1", "item2", "item3"};

StringBuffer sb = new StringBuffer();
int i = 1;
While(i<=30)
{
     sb.append("Add some text….<br/>");
     hdi.setName("DynamicField_"+i);
     ActionDocument.setItemChoices("DynamicField_"+i, sChoices);
     StringBuffer sbCombo = hdi.getHTML(false); //false means edit mode
     Sb.append(sbCombo);
     i++;
}

ActionDocument.setItemValue("DynamicData", sb.toString());
```

Running Actions Programmatically

From time to time it may be necessary to run an action on a document. A common use is for a save action to run the corresponding open action. This saves duplicating code for the populating combo-boxes and adding standard items to the UI.

```
pSession.runActionOnDocument(ActionDocument, "AnActionName");
```

The following example could be a save action:

```
public String execute()
{
      //process the save-specific items here here…

      //now run some actions to refill the UI and lists
      pSession.runActionOnDocument(ActionDocument, "GlobalPrefill");
      pSession.runActionOnDocument(ActionDocument, "OpenSearchJobs");
      return "";
}
```

Sending email

Sending email is very easy. All that is required is to set the fields on a document, then call send() method of the document. The correct sending of email requires that the server is running the MAILER task and that it is configured correctly.

```
//Create a temporary Document
Document docEmail = new Document(pSystem, pSession.getSessionContext());
//set the from field to a value in the keywords collection
docEmail.replaceItem("From",
pSession.getKeywordValue("DefaultEmailSender"));
String sDate = puakma.util.Util.formatDate(new java.util.Date(), "EEEE d
MMMM yyyy");
//set the subject
docEmail.replaceItem("Subject", "Reminder for "+ sDate);
//Add some recipients
String sRecipients = "test@puakma.net,test2@puakma.net";
docEmail.replaceItem("SendTo", sRecipients);
//docEmail.replaceItem("CopyTo", sCCEmailAddress);
//docEmail.replaceItem("BlindCopyTo", sBCCEmailAddress);
docEmail.replaceItem("Body", The bulk of the email message goes here…");

//put the message in the outbound mail queue for processing by mailer
if(!docEmail.send())
{
      //email was not sent…
}
```

Note that the send method does not physically transfer the mail to the recipient's mail server. When the send() method returns true, this simply mean that the message was successfully placed in the outbound queue ready for processing by the MAILER task.

Retrieving files from the database
Once a file has been saved to the database, users will need a way to retrieve the files. Files are typically stored in a binary "BLOB" in the database which is why the mime-type and filename columns are also required. To get the file from the database an action is created to do the work. The HTML page references the action in the following way:

```
<a href="<P@Path @P>/GetFile?OpenAction&ID=17">Download File</a>
```

Next create an action called "GetFile" and add it to the design collection.

```
//retrieve the URL parameter
String sID = ActionDocument.getParameter("ID");
Connection cx=null;
try
{
      cx = pSession.getDataConnection("YourConnectionName");
      Statement Stmt = cx.createStatement();
      ResultSet RS = Stmt.executeQuery("SELECT * FROM ATTACHMENT_TABLE_NAME
WHERE FileID=" + sID);
      if(RS.next()) //only interested in the first row returned
      {
            //get the data from the resultset
            byte[] bufData = RS.getBytes("FileData");
            String sContentType = RS.getString("MimeType");
            String sFileName = RS.getString("FileName");
            //set the HTTP payload to return
            setBuffer(bufData);
            //take any spaces from the filename
            sFileName = sFileName.replaceAll(" ", "_");
            setContentType(sContentType+"\r\nContent-Disposition:
attachment; filename=" + sFileName);
      }
      RS.close();
      Stmt.close();
}
catch(Exception e)
{
      pSystem.doError("Error getting attachment data: " + e.toString(),
this);
}
pSystem.releaseSystemConnection(cx);

// return nothing, so the setBuffer() call takes effect
return "";
```

Custom HTTP Header Processors

As a plugin to the HTTP server or BOOSTER server, Puakma supports "Custom HTTP Header Processors". These processors are able to intercept the request from the browser client before the HTTP server gets to look at it. The Header Processor can the edit and alter the header before passing it on to the HTTP server. This is similar to the URL rewriter in Apache, only more powerful and dynamic due to the ability to edit the entire header. This allows the insertion or interrogation of Single-Sign-On information, automatic logins and many other possibilities.

This is an advanced topic to be covered more fully in a future BlackBook. Once compiled place the SampleHeaderProcessor.class file in the /Puakma/addins/ directory, and add the line:
`HTTPHeaderProcessors=SampleHeaderProcessor`
to the Puakma.config file. Multiple processors may be configured, simply separate with a comma in the config file. Following is an example header processor.

```java
import puakma.util.*;

public class SampleHeaderProcessor extends HTTPHeaderProcessor
{
    public boolean execute()
    {
        if(m_sURI.indexOf(".nsf")>0)
        {
            //this.replaceHeaderValue("Host", "yourdomino.server.com");
        }

        //alter the request type
        //maybe to stop data being posted
        //m_sMethod = "GET"; //or POST, HEAD etc

        //change the request completely
        //m_sURI = "/icons/pma_logo.gif";

        //add a new item to the header
        //m_alHeaders.add("New-Item: Your data");

        //get a value from the header
        //String s = this.getHeaderValue("User-Agent");

        //log a message to the console
        m_pSystem.doInformation(this.getClass().getName()+" is
processing...", this);
        //log an error to the console
        m_pSystem.doError("ERROR MESSAGE", this);

        //return false to allow the next processor in the list work on this
header too
        //if you return true, no more custom header processing will occur
by other
        //loaded header processors
        return false;
    }

}
```
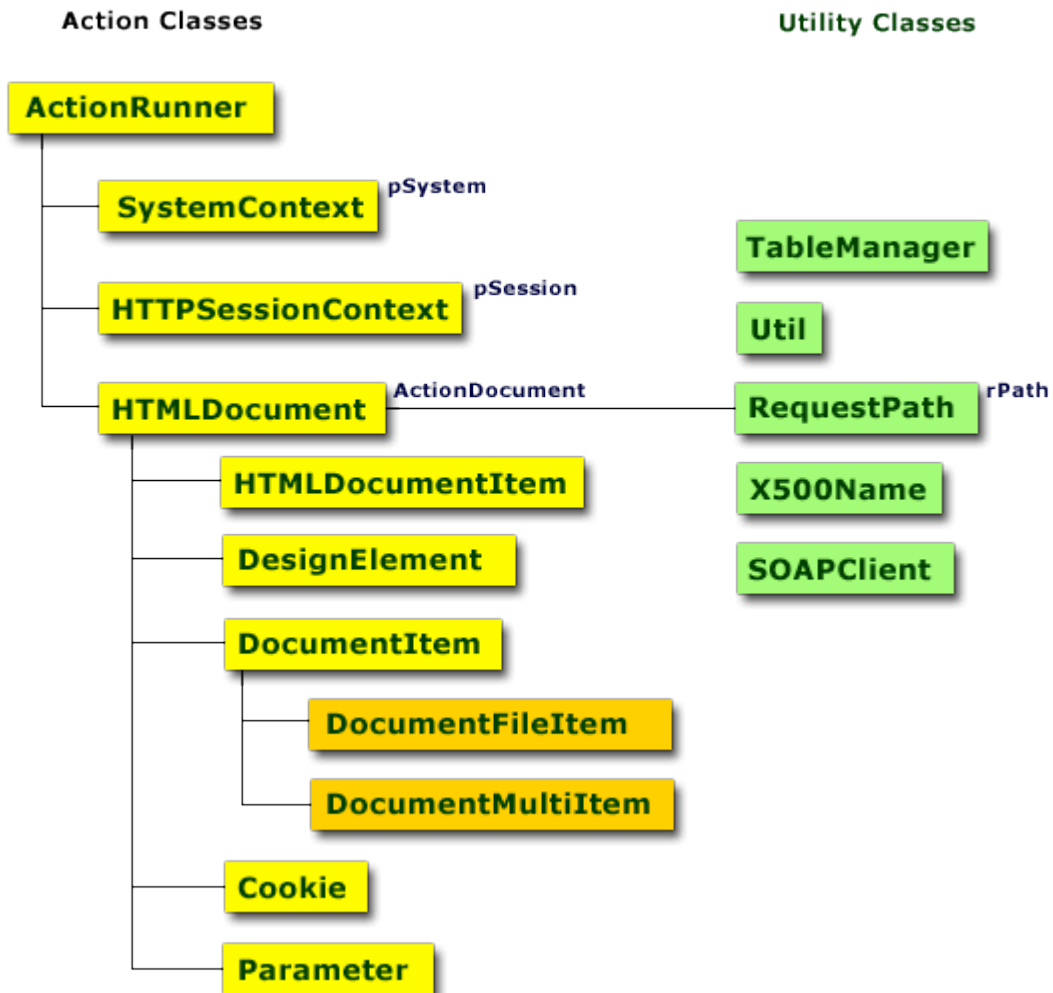
Extending the server with AddIn tasks

Puakma can be easily extended by writing AddIn tasks that are loaded as part of the server startup. An AddIn task may perform any function, some examples of tasks that are shipped as part of the core product are HTTP, BOOSTER, AGENDA, WIDGIE, MAILER, …

AddIns may be packaged as class files or packaged into a .jar file. It is recommended that you place your own addins in the /addins/ folder. By placing jars or classes here, they will be automatically added to the classpath.

The authoring of an AddIn is beyond the scope of this blackbook. For a basic example, please see the SampleAddIn.java file in the /addins/ folder.

Object Model for Tornado Server Actions

Puakma Object Model

Action Classes   **Utility Classes**

ActionRunner

    SystemContext   pSystem

                                  TableManager

    HTTPSessionContext   pSession

                                  Util

    HTMLDocument   ActionDocument   **RequestPath**   rPath

        HTMLDocumentItem

                                X500Name

        DesignElement

                                SOAPClient

        DocumentItem

             DocumentFileItem

             DocumentMultiItem

        Cookie

        Parameter

Additional Resources and Further Reading

HTTP
http://www.w3.org/Protocols/rfc2616/rfc2616.html

Web Services, SOAP
http://www.javaworld.com/javaworld/jw-03-2001/jw-0330-soap.html
http://www.w3.org/TR/SOAP/

Java
http://java.sun.com

SQL
http://www.mysql.com
http://hsqldb.sourceforge.net

HTML
http://www.davesite.com/webstation/html/
http://validator.w3.org/