TEASERVLET USER MANUAL

OCTOBER 18, 2001

PLEASE SEND ALL COMMENTS TO: REECE.WILTON@DIG.COM

PREFACE

This document is a manual of the TeaServlet and is intended for use by developers familiar with Servlet technology. The information provided here focuses on installation, configuration, and administration. Although there is some discussion of design strategy, TeaServlet users may wish to review the *TeaServlet Tutorial* to gain more comprehensive insight into the use of the TeaServlet.

There are a number of other documents complementary to this manual that cover topics associated with Tea and Tea-related tools. For additional Tea-related documentation, such as the *TeaServlet Tutorial*, please visit http://pubsys1.starwave.com:8020/.

DOCUMENT REVISION

Primary Author(s)	Description of Version	Date Completed
Brian S. O'Neill	Initial public release.	06-30-2000
Michael Rathjen	Minor update.	07-12-2000
Michael Rathjen	TeaServlet 1.2 update.	08-25-2000
Michael Rathjen	Exception guardian update.	08-30-2000
Michael Rathjen	TeaServlet properties added.	10-16-2000
Michael Rathjen	New TeaServlet functions added.	02-06-2001
Michael Rathjen	Installation, configuration, and administration added.	03-16-2001
Michael Rathjen	TeaServlet 1.3.2 update.	04-25-2001
Michael Rathjen	RegionCaching added.	05-04-2001
Michael Rathjen	Minor update.	05-14-2001
Michael Rathjen	Update for admin pages.	06-20-2001
Michael Rathjen	Added lazy template loading.	07-05-2001
Michael Rathjen	Added gzip region caching.	08-09-2001
Michael Rathjen	Minor update.	10-18-2001

CONTENTS

1. TEASERVLET INTRODUCTION

The TeaServlet allows dynamic web pages to be created by Tea templates. Tea is a language designed specifically for text formatting, and the TeaServlet uses it to enforce separation between actual web page design and Application logic. Further information on Tea can be found in the *Tea Template Language* document, found at http://pubsys1.starwave.com:8020.

Tea requires a host environment in order to run, and the TeaServlet is just that. Compilation support is also integrated, making it a complete Tea environment. A set of administration pages is provided that allows templates to be recompiled and safely reloaded.

The TeaServlet is designed to work with the Servlet API, version 2.1 or higher, and it requires a Java2 environment. It is easily configured into any servlet engine, just like any other servlet.

Development using TeaServlet works best for projects run by small teams of developers and technical producers. A developer's role is the creation of *Applications*, written in Java, which are installed into the TeaServlet. A producer creates and maintains the final appearance of dynamic web pages by writing Tea templates that call upon *functions* provided by the developer's Applications.

2. GETTING STARTED

2.1 The TeaServlet Website

The TeaServlet's home is the Content Delivery website at http://pubsys1.starwave.com:8020/. Here you will find documents, source code, and additional files related to the TeaServlet, Tea, Barista, and Kettle. This website also provides contact information for joining the Tea Template Authors mailing list. The mailing list is an excellent resource for getting help about the TeaServlet and the other products. Technical producers with real-world Tea experience subscribe to this list and will help answer your questions.

2.2 TeaServlet Installation

2.2.1 Requirements

The TeaServlet requires both Java 1.2 or greater and a servlet container that supports the Servlet API 2.1. The servlet container itself must also support Java 1.2 or greater for the TeaServlet to run in it.

Barista, the internally developed servlet container, includes the TeaServlet. TeaServlet is automatically installed and configured as part of Barista.

2.2.2 Download

To download the latest version of Barista, go to http://pubsys1.starwave.com:8020/Barista/.

2.2.3 Installing TeaServlet

TeaServlet is installed automatically when Barista is installed. To install Barista, *see the Barista User Manual*.

2.3 Testing the Installation

Administration pages are available once Barista is installed properly. You can test your installation by accessing these pages. If you have any problems running the administration pages, look for error messages from TeaServlet in Barista's log file.

See 3 TeaServlet Administration for detailed instructions on accessing the TeaServlet administration pages.

3. TEASERVLET ADMINISTRATION

3.1 Accessing Admin Pages

Administration pages are available once Barista is installed. The URL to the administration page is:

```
http://<YourMachine>/<map>/system/teaservlet/Admin?<admin.key>=<value>
```

where **<YourMachine>** is the name of the server, **<map>** is the URI segment mapped to TeaServlet, and **<admin.key>** and **<value>** are the values set for **admin.key** and **admin.value**.

If the default values from the sample properties file are retained, the URL to the admin application will be:

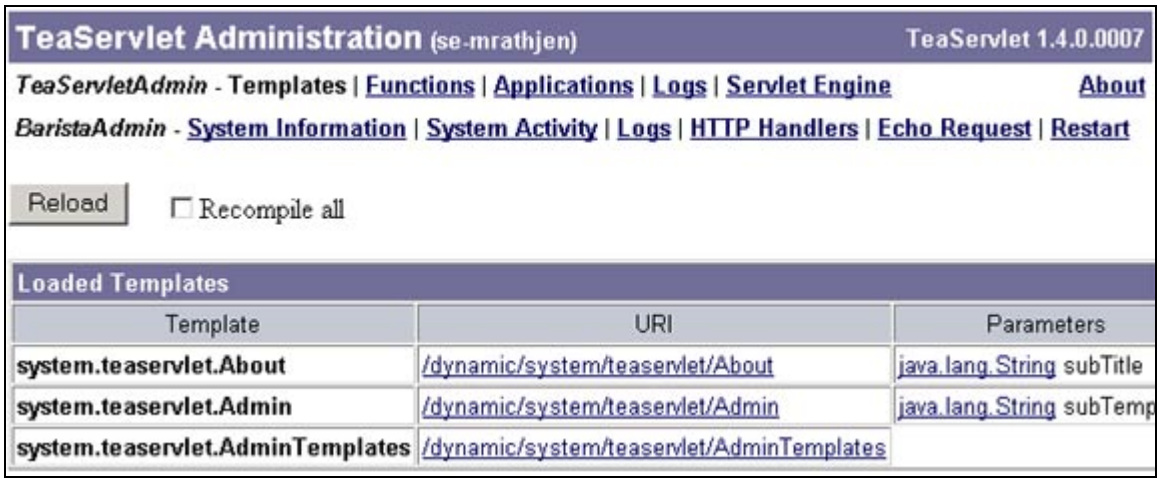
```
http://<YourMachine>/dynamic/system/teaservlet/Admin?magicKey=go
```

The **magicKey=go** parameter is a small security measure designed to stop people from accessing the administration pages. You can change the values of these parameters by modifying the **admin.key** and the **admin.value** parameters in the TeaServlet properties file. For example, changing the **admin.key** parameter to "**open**" and the **admin.value** parameter to "**source**" would change your URL to "**…system/teaservlet/Admin?open=source**". Although you should set those parameters to values that are hard to guess, that alone is not secure enough for a web site. You should also configure your firewall to only allow the administration pages to be hit from IP addresses that are internal to your organization. This will stop external people from accessing your administration pages.

If Barista was installed and configured correctly, the above URL will result in the main administration page. It should look similar to this:

3.1.1 Templates Page

Click on the **Templates** link to view the list of loaded templates.

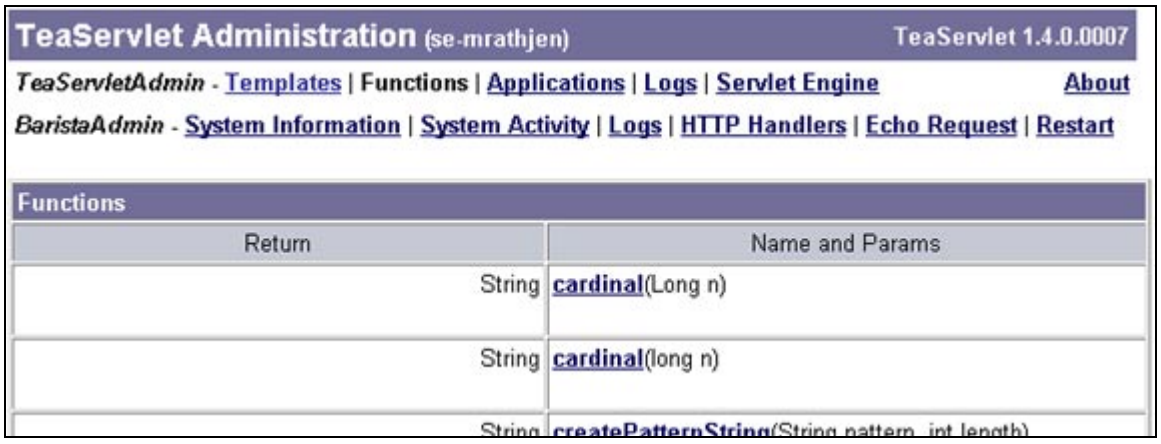


When you have made changes to a template, you can use the **Reload** button on this page to reload your templates. Note there are two checkbox items to the right of the reload button: **Recompile all** and **Apply to cluster**. With these options off, the **Reload** button will recompile any changed templates and reload them into the TeaServlet. If **Recompile All** is checked, the **Reload** button will recompile all templates and reload them into the TeaServlet. If **Apply to Cluster** is checked, the reloading will occur across all servers in the cluster. If you have a server cluster setup in your TeaServlet properties, the servers will be listed to the right of the **Apply to** cluster checkbox. (Apply to cluster will not appear if there are no clustered servers setup in the Barista properties file.) If there are any compilation errors in your templates, a list of errors will display at the top of this page.

The templates table shows the names of all the templates that are currently in memory. Other templates may exist on your file system but they haven't been loaded yet. The URI column shows you the URI that you would use to execute the template. You can click on a URI to execute the template. The parameters column shows any parameters that the templates accept.

3.1.2 Functions Page

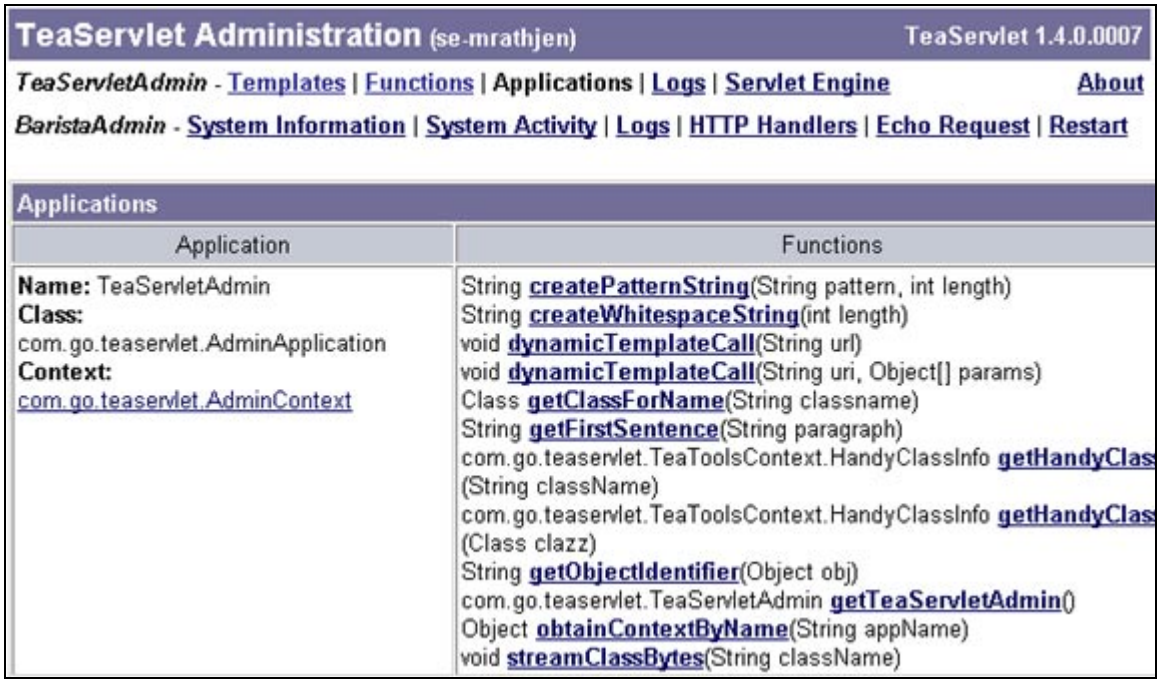
Click on the **Functions** link to view the list of available functions.



Your templates can call any of the listed functions. You can click on any of the functions on this page to view the description of the object. Any of these properties are accessible to you in the template.

3.1.3 Applications Page

Click on the Applications link to view the list of installed applications.

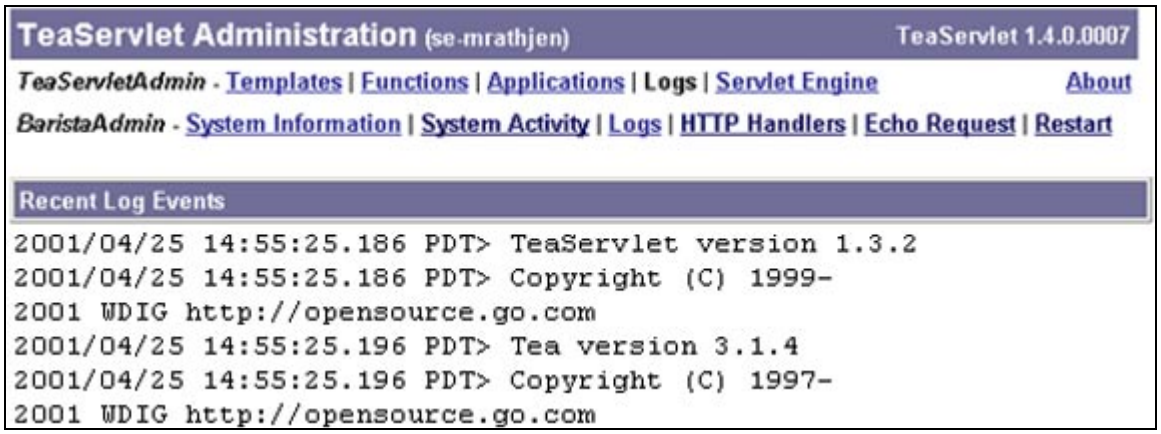


All applications that are configured in the properties file will show up here. The context and the list of functions that the applications provide are also displayed. As you configure more applications, you will be able to view them on this page.

3.1.4 Logs Page

Click on the **Logs** link to view the log entries that are currently in memory.

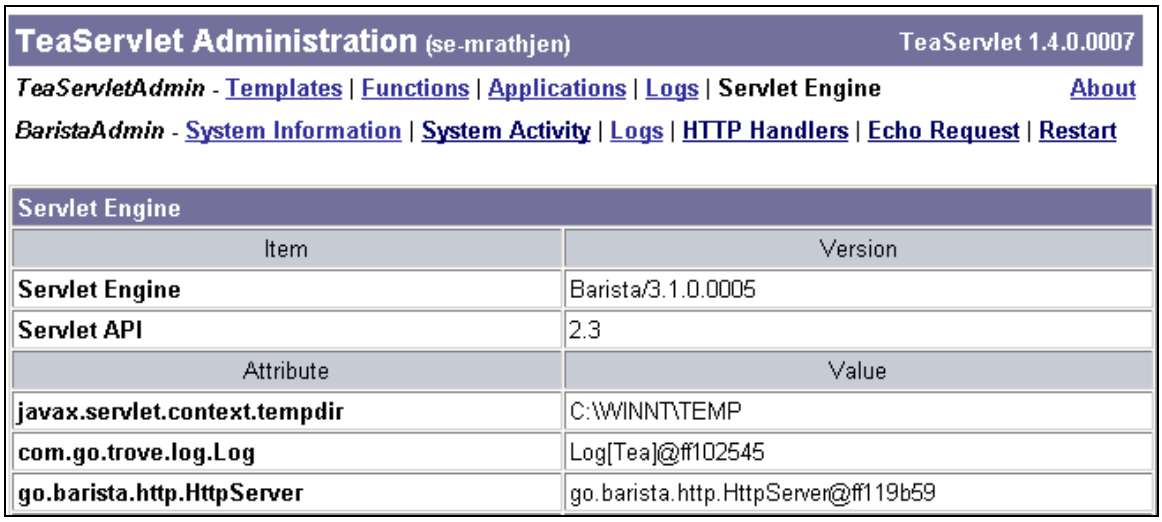
Barista and TeaServlet log information are accessed through the TeaServlet's log admin page. This log page displays recent Barista and TeaServlet log events. Additional information can be displayed by enabling the **Debug**, **Full Timestamp**, **Threads**, and **Log Names** switches.



The **log.max** property in the properties file controls the number of log lines to keep in memory.

3.1.5 Servlet Engine Page

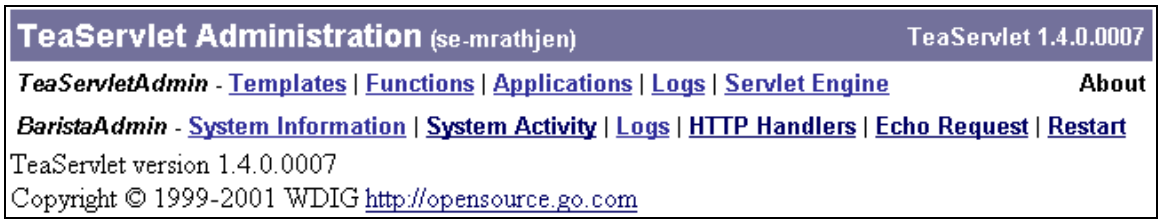
Click on the **Servlet Engine** link to view information that the TeaServlet has about the servlet container in which it is running.



You can see the servlet container name and version as well as the version of the Servlet API that it supports. The attribute table displays the list of attributes that are accessible through the ServletContext class. The ServletContext class is part of the Servlet API. You can also see the initialization parameters that the servlet container has passed to the TeaServlet from the Barista properties file.

3.1.6 About Page

Click on the **About** link to view information about the TeaServlet.



3.2 Extensible Admin Pages

TeaServlet admin pages are extensible. To include your own admin links, implement the AdminApp interface. (For more information, see the Java Docs at http://opensource.go.com/JavaDocs/com/go/teaservlet/AdminApp.html.)

The following screen shot illustrates the TeaServlet admin page with a few additional admin applications.

4. TEASERVLET PROPERTIES

TeaServlet is configured using order-dependent name value pairs, passed in by a Servlet Engine as servlet initialization parameters. Most Servlet Engines don't preserve the ordering of initialization parameters because they are not required to. For this reason, just one initialization parameter is required, "properties.file", which refers to a properties file.

4.1 Overview

The TeaServlet accepts properties in a format very similar to that of java.util.Properties, with the following exceptions:

- Ordering of properties is preserved.
- Values have trailing white space trimmed.
- Quotation marks (" or ') can be used to define keys and values that have embedded spaces.
- Quotation marks can also be used to define multi-line keys and values without having to use continuation characters.
- Properties may be nested using braces '{' and '}'.

Just like properties, comment lines start with optional white space followed by a '#' or '!'. Keys and values may have an optional '=' or ':' as a separator, unicode escapes are supported as well as other common escapes. A line may end in a backslash so that it continues to the next line. Escapes for brace characters '{' and '}' are also supported. Example:

```
# TeaServlet properties.

# Path to locate Tea templates, in .tea files.
template.path = ./templates

# Optional path to write compiled templates, as .class files.
template.classes = ./templates

# Applications to load into TeaServlet.
applications {
    "MyApp" {
        # Application class.
        class = pkg.SimpleApp

        # Application specific initialization.
        init {
            driver = sqldriver
            url = jdbc://somehost
            user = username
            password = secret
        }
    }

    "Admin" {
        class = com.go.teaservlet.AdminApplication

        init {
            # The security key for the Admin page.
            admin.key = admin
            admin.value = true
```

```
            }
        }
    }
```

is equivalent to:

```
# TeaServlet properties.

# Path to locate Tea templates, in .tea files.
template.path = ./templates

# Optional path to write compiled templates, as .class files.
template.classes = ./templates

# Application class.
applications.MyApp.class = pkg.MyApplication

# Application specific initialization.
applications.MyApp.init.driver = sqldriver
applications.MyApp.init.url = jdbc://somehost
applications.MyApp.init.user = username
applications.MyApp.init.password = secret

applications.Admin.class = com.go.teaservlet.AdminApplication

# The security key for the Admin page.
applications.Admin.init.admin.key = admin
applications.Admin.init.admin.value = true
```

4.2 TeaServlet Properties

The Barista properties file includes the TeaServlet properties. These properties are provided with default values to give users a starting point in customizing their own properties. The default property settings are also the settings that typical users might prefer. These properties are identified and described in the following subsections.

4.2.1 Default Template

TeaServlet can be configured to support a default template, which is very similar to the way in which index.html is defaulted in most web servers. Any request to TeaServlet that doesn't refer to a specific template will cause a lookup against the default template.

```
template.default = index
```

If **template.default** is set to "index" and a request is made to /somepath/section, but no such template exists, TeaServlet looks for /somepath/section/index. If the default template is found, but the original request didn't end in a '/', TeaServlet redirects to the original path with a trailing slash. This way, relative hyperlinks still work.

4.2.2 Path Properties

```
template.path = ./templates
template.classes = ./templateClasses
```

The **template.path** property identifies to TeaServlet the location to look for Tea templates (.tea files). The **template.classes** property identifies a location to write compiled files (.class files).

4.2.3 'Lazy' Template Loading

```
template.preload = false
```

The **template.preload** property, when set to **false**, prevents templates from preloading. TeaServlet will preload templates by default if this property is missing or set to any other value.

Lazy template loading is primarily used when a small memory footprint is desirable, yet there are a large number of templates (thousands) available. Templates will not be loaded until they are actually needed.

4.2.4 Application Properties

The applications section of the TeaServlet properties file is used to identify and configure applications. For instruction on creating and configuring additional applications, see section 6 Applications.

The default properties file includes generic settings for the TeaServlet application.

```
applications {
    "System" {
        # The SystemApplication provides TeaServlet/Barista
        # administration support.
        class = com.go.teaservlet.AdminApplication

        init {
            # The security key for the Admin page.
            admin.key = admin
            admin.value = true
        }

        log {
            debug = true
            info = true
            warn = true
            error = true
        }
    }
}
```

The **class** property identifies the application's class. The **class** property is required for all applications. However, properties containing configuration settings will differ from application to application. In the case of the TeaServlet AdminApplication, the **init** and **log** sections contain the configuration settings.

The **init** section for the AdminApplication contains properties that provide access to the administration pages. The use of this key and value to access the administration pages is described in the *TeaServlet Tutorial* document. For security reasons, both of these values should be changed from the defaults soon after installation.

The **log** section for the AdminApplication contains properties that determine what type of information will be recorded from the AdminApplication to the servlet container's log. For example, a setting of "true" will record that type of information in the servlet container's log. **Debug** information can be lengthy, so it is best to switch that property value to "false" once testing is complete. The **debug**, **info**, **warn**, and **error** settings are described in 4.2.5 Logs.

4.2.5 Logs

TeaServlet contains a special set of classes for manipulating logs, which provide additional functionality not provided to a standard servlet. These logs are intended for recording operational information, not for counting requests. Logs are grouped into four categories, debug, info, warn, and error. Each can be enabled or disabled separately for TeaServlet as a whole or for an individual Application. These logs can also be controlled via the administration pages.

In the current implementation of TeaServlet, all log output is directed to the standard servlet logging methods. Log output is also saved in memory, viewable from the administration pages. The memory log is limited in size, and old entries are cleared. The following log settings example can be placed in the main TeaServlet properties area or within an individual Application. Settings not provided by an Application inherit TeaServlet settings. An Application that enables a certain log forces TeaServlet to enable its log.

The **log** section for TeaServlet contains properties that determine what type of information will be recorded from TeaServlet to the servlet container's log.

```
#teaServlet.log.max = 100
log.debug = true
log.info = true
log.warn = true
log.error = true
```

The **teaservlet.log.max** property controls how many log entries are stored in temporary memory for display on the admin page. If this property is absent or commented-out, TeaServlet defaults to 100 entries. The **debug**, **info**, **warn**, and **error** properties determine which types of information will be logged. For example, **error=true** will cause all error messages to be logged and **debug=false** will prevent debug messages from being logged. If these properties are absent in the properties file, Barista defaults to true.

4.2.6 Exception Guardian

TeaServlet 1.2 adds a new feature called "exception guardian". In certain cases, exception guardian prevents the server from returning an error page due to a null pointer exception in the data. When exception guardian is enabled, the web page can still be displayed although the missing data may cause an incorrect presentation. Nevertheless, a presentation that is slightly off is usually preferred over an error page.

To enable exception guardian, add the following line to the TeaServlet properties file:

```
template.exception.guardian = true
```

Using exception guardian can cause compile times to noticeably increase.

4.2.7 "Spiderable" URLs

Many search engine spiders reject indexing of URLs that contain a query string. The assumption is that a page with a query string is likely to be very dynamic, and the indexes generated for that page will become stale very quickly. This is often an invalid assumption. To allow query strings for a page and allow it to still be indexed, TeaServlet supports additional characters for query string separators.

The following configuration example, placed into the properties file, allows TeaServlet to process additional patterns for the query string separator characters of '?', '=', and '&'.

```
separator.query = .html/
separator.value = -
separator.parameter = /
```

A request to something like /path/directory?user=9999&show=address can also be requested as /path/directory.html/user-9999/show-address.

4.2.8 Setting up a Server Cluster

Servers can be clustered together to allow servlet and template reloads from one location. To setup a server cluster, add a **cluster.servers** property to the init section of the TeaServlet. List the server names or IPs separated by commas, as shown in the following example:

```
servlets {
        "Tea" {
            class = com.go.teaservlet.TeaServlet

            init {
              cluster.servers = server0,server1,server2
```

Once a server cluster is setup, reloading servlets on your computer will reload those same servlets on the other servers in the cluster. (The servlet properties section must have the same names.) See 3.1.1 Templates Page for more information.

5. CREATING TEMPLATES

Tea templates output web pages in TeaServlet, and the template files reside in a special directory configured using the template.path parameter. Template files can appear in the root template path or may be in any subfolder. All template files must have a ".tea" file extension. Refer to the *Tea User's Guide* for help on writing templates.

Templates are matched using the servlet request's path info property. Other server-side page-generation technologies, like JSP or shtml, rely on a special file extension to let the server know that the page has a special handler. With TeaServlet, the ".tea" file extension is not needed and doesn't appear in the URI.

Consider a TeaServlet configured in a Servlet Engine such that all URIs beginning with /tea/ are mapped to it. Any request to the URI /tea/SimplePage causes TeaServlet to execute the template defined in SimplePage.tea, located at the root of the template path. If the template does not exist, a "404 Not Found" error is returned.

Here is a possible definition for SimplePage:

```
<% template SimplePage() %>
This is a simple page that does nothing special.
```

The page can be made to accept query parameters:

```
<% template SimplePage(String name) %>
This is a simple page that does nothing special
except show a name: <% name %>
```

The page can be requested as /tea/SimplePage?name=Waldo. If the name parameter is not specified, null is passed to template. A template can directly accept any number of request parameters, and they can be Strings, Numbers (or a subclass, like Integer), or arrays. Request parameters can also be retrieved using the getRequest function:

```
<% template SimplePage()
name = getRequest().parameters["name"]
%>
This is a simple page that does nothing special
except show a name: <% name %>
```

The getRequest function is built into TeaServlet, but other functions can be defined by creating Applications, which are discussed in the next section.

Changes to templates are not seen until the servlet is restarted or the reload button is hit from the administration page. A special Application that comes with TeaServlet needs to be configured in order for the administration pages to operate. Its class is com.go.teaservlet.AdminApplication. The templates that create the pages are pre-compiled and bundled with TeaServlet.

The administration pages are at /tea/system/teaservlet/Admin. See 3 TeaServlet Administration for more information about accessing and using the administration pages. Template reloads via the admin page are safe because it doesn't affect the behavior of any templates currently running, and it rejects all changes if any templates fail to compile.

If the optional template.classes property is configured, compiled template classes are written out as files. This feature allows for faster compilation and startup because only templates that have changed are recompiled.

Because Tea templates are compiled to Java classes, debugging templates involves similar steps as for debugging Java code. The following template generates a NullPointerException when the name parameter isn't passed to it.

```
<% template SimplePage(String name) %>
This is a simple page that does nothing special
except show a name: <% name %> and its
length: <% name.length %>
```

The Servlet Engine will likely generate an error page, "500 Internal Server Error", and log the exception. The following stack trace helps identify where the problem is, line 4 of the SimplePage template. You may notice the com.go.teaservlet.template package prefix for the template. All templates used with TeaServlet are compiled to this package, which is reserved for compiled templates. This guards against the rare possibility of a template class overriding a Java class.

```
java.lang.NullPointerException
    at com.go.teaservlet.template.SimplePage.execute(SimplePage.tea:4)
    at java.lang.reflect.Method.invoke(Native Method)
    at com.go.tea.runtime.TemplateLoader$TemplateImpl.execute
        (TemplateLoader.java:228)
    at com.go.teaservlet.TeaServlet.processTemplate(TeaServlet.java:470)
    at com.go.teaservlet.TeaServlet.doGet(TeaServlet.java:238)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:715)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:840)
```

Here's a fixed version that can be reloaded without restarting the servlet by using the admin pages:

```
<% template SimplePage(String name)
if (name == null) {%>
No name provided!
<%} else {%>
This is a simple page that does nothing special
except show a name: <% name %> and its
length: <% name.length %>
<%}%>
```

5.1 Pre-Compiled Templates

Generally, TeaServlet is deployed in an environment where the template sources are available to it and can be freely modified. Sometimes it's appropriate to distribute pre-compiled templates with it, although this is an advanced feature that most users of TeaServlet will not have to deal with.

TeaServlet administration templates are distributed in binary form in order to make it easier to install. Since most users won't wish to modify these templates, having the sources available in the template source directory is not necessary. However, the default pre-compiled templates can be overridden by templates in the template source directory if the newer templates have the same name and path.

When TeaServlet needs to load a template for execution, it doesn't look to the template source directory. Rather, it asks the template class loader to find the template in the com.go.teaservlet.template package. This is why the template source need not be available for running a template.

The TeaServlet jar file contains all the pre-compiled template classes, in the correct base package. In order for those classes to appear there, a special build process is required that invokes a command-line Tea file compiler. This file compiler is com.go.tea.util.FileCompiler. When the jar file is created, the compiled results from both Java and Tea sources are added.

All Tea templates must be compiled against a context class, and there can be only one. Since TeaServlet is capable of running multiple Applications, each with distinct contexts, TeaServlet creates a special auto-generated merged class in order to satisfy the one context class requirement of the Tea compiler. If your pre-compiled templates need to access functions from several different contexts, then you won't be able to use the auto-generated merged class because it only exists at runtime.

All of the functions needed by the pre-compiled templates must be defined in contexts that are interfaces. A single interface must be defined that extends all of these other interfaces, but adds no additional members. The templates may then be compiled against this single context interface. An example of this is com.go.teaservlet.AdminHttpContext, which is used for pre-compiling the admin templates.

When the pre-compiled template is loaded, TeaServlet will auto-generate an adapter class that implements the single context interface. The adapter wraps all calls to the auto-generated merged class, and then the pre-compiled templates can be executed.

If any context required by a pre-compiled template is not provided by any configured Applications, then calls to those missing functions will generate a NullPointerException from within the adapter class.

Here is a sample invocation of the FileCompiler to pre-compile the admin templates:

```
java com.go.tea.util.FileCompiler -context
com.go.teaservlet.AdminHttpContext -dest classes/com/go/teaservlet/template
-force -package com.go.teaservlet.template TeaSourceDir
```

6. APPLICATIONS

Java development in TeaServlet is focused around the creation of Applications, which are loaded into TeaServlet in a manner similar to the way Servlets are loaded into a Servlet Engine. An Application provides a *context*, which is just an object whose methods are callable from Tea templates as functions. Multiple Applications may be loaded into a single TeaServlet, and templates may access functions from all of the Applications.

Applications must implement the Application interface, which has init and destroy methods similar to Servlets, as well as methods for managing contexts. The init method receives an ApplicationConfig object, which extends the regular ServletConfig.

For managing contexts, an Application has the getContextType and createContext methods. The getContextType method is only called during TeaServlet initialization, and it returns a Class describing what kind of object will be returned by createContext. Returning null indicates that the Application does not create a context, and so createContext will never be called.

All public methods in the context object that were not defined in java.lang.Object are available to be called by Tea templates as functions. No special interface needs to be implemented by a context.

Since most Applications will define functions, they will have a context type. On every request made to a template, TeaServlet asks each Application that has a context type to provide a context object via the createContext method. Here's a very simple Application:

```
public class SimpleApp implements Application {
    public void init(ApplicationConfig config) throws ServletException {
    }

    public void destroy() {
    }

    public Class getContextType() {
        return SimpleContext.class;
    }

    public Object createContext(ApplicationRequest request,
                                ApplicationResponse response) {
        return new SimpleContext(request);
    }
}

public class SimpleContext {
    private ApplicationRequest mRequest;

    public SimpleContext(ApplicationRequest request) {
        mRequest = request;
    }

    public boolean isCompatibleBrowser() {
        String userAgent = mRequest.getHeader("User-Agent");
        if (userAgent != null) {
            return userAgent.indexOf("Mozilla") >= 0;
        }
        else {
            return false;
        }
    }
}
```

This Application provides just one function, isCompatibleBrowser. Templates can call this and decide on what kind of formatting can be used or redirect the user to a download page. The Application is configured to run in TeaServlet by adding an entry into the properties file. Because this simple Application does not need any initialization data, no init section needs to be provided in the configuration.

```
applications {
    "MyApp" {
        class = pkg.SimpleApp
    }
    ...
}
```

Most Applications will provide more interesting functions. The following changes cause the SimpleApp to establish a database connection and provide query results.

```
package pkg;

import java.util.*;
import java.sql.*;
import javax.servlet.*;
import com.go.teaservlet.*;
import com.go.trove.util.*;
import com.go.trove.log.*;

public class SimpleApp implements Application {
    private Log mLog;
    private Connection mCon;

    public void init(ApplicationConfig config) throws ServletException {
        mLog = config.getLog();
        PropertyMap properties = config.getProperties();

        // Load the JDBC driver class.
        try {
            Class.forName(properties.getString("driver"));
        }
        catch (ClassNotFoundException e) {
            throw new ServletException(e);
        }

        Properties dbProperties = new Properties();
        dbProperties.putAll(properties);

        try {
            mCon = DriverManager.getConnection
                (properties.getString("url"), dbProperties);
        }
        catch (SQLException e) {
            throw new ServletException(e);
        }
    }

    public void destroy() {
        try {
            mCon.close();
        }
        catch (SQLException e) {
            mLog.error(e);
        }
    }

    public Class getContextType() {
        return SimpleContext.class;
    }
```

```
    public Object createContext(ApplicationRequest request,
                                ApplicationResponse response) {
        return new SimpleContext(this, request);
    }

    synchronized UserRecord getUserRecord(int userId) throws SQLException {
        String sql = "select name, age from users where id=" + userId;
        ResultSet rs = mCon.createStatement().executeQuery(sql);
        try {
            if (rs.next()) {
                return new UserRecord(rs.getString(1), rs.getInt(2));
            }
            else {
                return null;
            }
        }
        finally {
            rs.close();
        }
    }
}

public class SimpleContext {
    private SimpleApp mApp;
    private ApplicationRequest mRequest;

    public SimpleContext(SimpleApp app, ApplicationRequest request) {
        mApp = app;
        mRequest = request;
    }

    // Available as function in Tea templates.
    public boolean isCompatibleBrowser() {
        String userAgent = mRequest.getHeader("User-Agent");
        if (userAgent != null) {
            return userAgent.indexOf("Mozilla") >= 0;
        }
        else {
            return false;
        }
    }

    // Available as function in Tea templates.
    public UserRecord getUserRecord(int userId)
        throws java.sql.SQLException
    {
        return mApp.getUserRecord(userId);
    }
}

public class UserRecord {
    private final String mName;
    private final int mAge;

    public UserRecord(String name, int age) {
        mName = name;
        mAge = age;
    }

    public String getName() {
        return mName;
    }

    public int getAge() {
        return mAge;
    }
}
```

Here's a template that uses the functions defined by SimpleApp:

```
<% template ShowRecord(Integer id) %>
<html><body><%
if (!isCompatibleBrowser()) {
    'Upgrade your browser'
}
else if (id == null) {
    'No id provided'
}
else {
    record = getUserRecord(id)
    '<table border="1">
    '<tr><th>Name</th><th>Age</th></tr>'
    '<tr><td>' record.name '</td><td>' record.age '</td></tr>'
    '</table>'
}
%>
</body></html>
```

TeaServlet allows multiple Applications to be configured into it, even multiple instances of the same type. Each Application is configured with a name that is unique, and each Application may receive private initialization parameters. All the functions provided by the set of configured Applications are merged together so that they are all sharable from a single set of templates.

Because many independently developed Applications may be loaded into a TeaServlet, the possibility of duplicate or conflicting function signatures exists. The order in which Applications are configured is important, and functions defined in Applications configured before other Applications have priority over them.

Functions can be called using a complete name that consists of the configured Application name and a dot prefixing the function name. In the example Application, configured with the name "MyApp", the getUserRecord function can also be called from a template as MyApp.getUserRecord().

6.1 Applications Included with the TeaServlet

6.1.1 AdminApplication

The AdminApplication is configured by default. See sections 3 TeaServlet Administration and 4.2.4 Application Properties for more information.

6.1.2 RegionCachingApplication

The RegionCachingApplication allows template writers to specify code regions to cache within a template. To improve performance, you may want to cache code regions if they contain code that amounts to the same result (based upon a key).

To install the RegionCachingApplication, add the following application properties to the TeaServlet properties file:

```
applications {
    ...
    "RegionCache" {
        class = com.go.teaservlet.RegionCachingApplication
        init {
            cache.size = 500
            default.ttl = 5000
            timeout = 500
            gzip = 6
        }
    }
    ...
}
```

The **cache.size** property sets the minimum guaranteed number of code regions to cache. The **default.ttl** property sets the default time-to-live (in milliseconds) for cached regions. **Timeout** indicates the maximum time in milliseconds to wait for the cache before serving an expired code region.

The **gzip** property determines the level of gzip compression of a cached code region. This property can be set to any integer, 0 to 9. 0 sets no compression and 9 sets the highest level of compression. Choose higher levels of compression to use less bandwidth and lower levels of compression to conserve server processor load. The default setting is 6.

The code region will only be compressed when a non-expired, cached code region is requested. Otherwise, the server would waste time compressing code regions that are expired or that may be only requested once (not yet cached). When compressing a code region, the server will also keep the non-compressed code region available for browsers that do not support gzip compression. This means that setting *any* level of compression (aside from 0) will result in more memory use.

Caching Code Regions

Use the following syntax to call the cache function and cache a region of code:

```
cache( TTL , key ) {
     // Cached template code and text goes here
 }
```

TTL and **key** are optional arguments that may be passed into the function. **TTL** overrides the default time-to-live setting in the properties file.

The **key** argument is used to provide additional information to help identify the cached region's key. Otherwise, regions are keyed on enclosing template, region number, and HTTP query parameters. An array can be used to pass in multiple values for **key**.

The cache() function can be invoked multiple times within a template and the cache calls can be nested within each other.

A nocache() function is also provided to prevent the caching of a specific region within a cached region. This may be useful in cases where most code needs to be cached and only small regions must not be cached.

```
cache() {
     // Cached template code and text

       nocache() {
              // Non-cached template code and text within cached region
       }

     // Cached template code and text
 }
```

When templates are reloaded, the cached regions are dropped.

7. DESIGN STRATEGIES

Although the example Applications provided in this document are small and simple, in TeaServlet large-scale Applications are typically developed. These tips will help you in the design of your Applications.

Most of the logic for processing functions should reside in the Application, not the context. The context should be treated as a gateway between the templates and your Application. The context should contain a reference to the Application, or some other object(s) that do the real work.

An additional motivation for this design is the fact that new context objects may need to be created on every template hit. This is necessary if any functions need direct access to the HTTP request and response objects. If an open resource needs to be maintained, such as a pool of database connections, that resource only needs to be created upon Application initialization. The context should receive either the connection pool it needs or access to an object that can provide it when needed. Using the Application object as a container of resources is the most convenient solution.

Don't let Applications grow to be too big. If an Application is becoming hard to maintain because of its size or there are several developers on a project, break up the Application into several simpler Applications. This is possible since each Application can be independently configured into TeaServlet.

If all of the Applications need access to the same resources, such as a connection pool, then you can use the servlet attribute passing mechanism to share those resources among Applications. You can even create an Application that defines no functions (returns null for the context type), but it creates and shares resources during initialization. This Application would always need to be configured before dependent Applications.

Write as few functions as possible, and give them descriptive names. Tea is intended to work against JavaBeans that are rich with properties, because it makes it easier to access related information. Template writers can use TeaServlet admin pages to browse the available properties on objects. Having to navigate through a sea of functions makes the process of writing and maintaining templates more difficult.

Using descriptive names also reduces the possibility of function name collisions. Although using the exact Application name can specify a function, this strategy can lead to problems if the configured name of an Application changes.

8. STANDARD FUNCTIONS

TeaServlet provides a standard set of functions in addition to the standard Tea functions. These functions aren't provided by any Application and they have highest priority. Most of the functions aid in accessing request parameters and controlling the response.

8.1 encodeParameter

```
String encodeParameter(String)
```

Encodes the given string so that it can be safely used in a URL. For example, '?' is encoded to '%3f'.

8.2 fileExists

```
boolean fileExists(String path)
```

Tests if a file at the given path exists. If the servlet that is running the template has a root directory, it is used to determine the physical file path. If the servlet does not have a root directory set, it is assumed to be the root of the file system. If the path given to this function does not lead with a slash, the path is relative to the pathInfo variable from the request.

8.3 getRequest

```
HttpContext.Request getRequest()
HttpContext.Request getRequest(String encoding)
```

Returns an object that contains all the request information from the HTTP client. When a character encoding is supplied, all the request parameters are converted against it. The Request object that is returned is not the same as HttpServletRequest in order to control which information should be available to a template and to make it easier to access from Tea.

8.4 insertFile

```
insertFile(String path)
```

Inserts the contents of the given file into the page output unless the file does not exist or cannot be read. If the servlet that is running the template has a root directory, it is used to determine the physical file path. If the servlet does not have a root directory set, it is assumed to be the root of the file system. If the path given to this function does not lead with a slash, the path is relative to the pathInfo variable from the request.

8.5 insertURL

```
insertURL(String URL)
```

Inserts the contents of the resource at the given URL into the page output. If the resource does not exist or cannot be read, nothing is inserted. The given URL can be absolute or relative.

8.6 readFile

```
String readFile(String path)
String readFile(String path, String encoding)
```

Reads and returns the contents of the given file. If the file does not exist or cannot be read, an empty string is returned. This function differs from insertFile in that the character conversion is applied and the file's contents are returned.

If the servlet that is running the template has a root directory, it is used to determine the physical file path. If the servlet does not have a root directory set, it is assumed to be the root of the file system. If the path given to this function does not lead with a slash, the path is relative to the pathInfo variable from the request.

8.7 readURL

```
String readURL(String URL)
String readURL(String URL, String encoding)
```

Reads and returns the contents of the given URL. If the URL does not exist or cannot be read, an empty string is returned. This function differs from insertURL in that the character conversion is applied and the resource's contents are returned.

8.8 sendError

```
sendError(int code)
sendError(int code, String message)
```

Causes an error page to be generated using the given status code and optional message. Further template processing is aborted and any output already provided by the template is not sent.

8.9 sendRedirect

```
sendRedirect(String URL)
```

Creates a response that forces the browser to redirect to the given URL. Further template processing is aborted and any output already provided by the template is not sent.

After calling sendRedirect, the remainder of the response should be aborted by throwing a com.go.teaservlet.AbortTemplateException.

8.10 setContentType

```
setContentType(String)
```

Sets the MIME type of the page, like "text/html" or "text/plain". The default content type for TeaServlet is text/html, and so this function doesn't need to be called for most HTML pages. If another character encoding is desired, such as for internationalization, append charset property like this: "text/html; charset=iso-8859-1".

8.11 setHeader

```
setHeader(String name, String value)
setHeader(String name, int value)
setHeader(String name, Date value)
```

This is an advanced function that sets the value of an arbitrary HTTP response header, like Last-Modified.

8.12 setStatus

```
setStatus(int code)
```

Sets the response's HTTP status code, like "200" for OK or "404" for not found.

8.13 setURLTimeout

```
setURLTimeout(long timeout)
```

Requests to check, insert, or read URLs will timeout if the remote hosts don't respond in time. Call this function to explicitly set the timeout value.

8.14 URLExists

```
boolean URLExists(String URL)
```

Tests if a resource at the given URL exists. If the URL has no protocol specified, then a relative lookup is performed. If the relative URL begins with a slash, the lookup is performed with an absolute path on this server. If the relative URL does not begin with a slash, then the lookup is performed with a relative path.