TEASERVLET TUTORIAL

MAY 18, 2001

PLEASE SEND ALL COMMENTS TO: OPENSOURCE@DIG.COM

PREFACE

The TeaServlet is a framework that sits on top of the Servlet API and creates a separation between data and presentation. The TeaServlet requires a servlet container to run in.

This tutorial is designed to guide new users through a few tasks that will lead to an understanding of the TeaServlet. We will look at how to build applications that work with it. We will also explore the Tea template language and Kettle, the template editor for Tea.

The tutorial is meant to supplement the other documents that are available on the TeaServlet, Tea, and Kettle. For TeaServlet installation, see the *TeaServlet User Manual*.

The tutorial presumes that you have an understanding of Java programming and an understanding of servlets. If you don't, refer to Appendix A: Servlet Resources for a list of servlet resources.

Visit http://opensource.go.com to find everything related to the TeaServlet, Tea, and Kettle.

DOCUMENT REVISION AND EVOLUTION

Primary Author(s)	Description of Version	Date Completed
Reece Wilton	Initial public release.	July 12, 2000
Reece Wilton	Fixed error in Tomcat installation (section 2.2.1).	July 14, 2000
Reece Wilton	Minor changes to Tomcat installation. Added screen shots to section 2.3.	July 17, 2000
Michael Rathjen	Minor editing corrections. Clustered administration screens added.	August 25, 2000
Michael Rathjen	Edited document to sync content with the TeaServlet User Manual.	May 18, 2001

CONTENTS

1. GETTING STARTED

1.1 opensource.go.com

The TeaServlet's home is Walt Disney Internet Group's (WDIG) Open Source Web Site at http://opensource.go.com. Here you will find documents, source code and additional files related to the TeaServlet and the other open source projects at WDIG. You will also find the message board and a place to sign up for product announcements via email. The message board is a good place to get help and information for the TeaServlet. Feel free to ask for information, post suggestions, or ask any questions that you have because the TeaServlet community monitors this board frequently.

1.2 Installation

The first thing you need to do is download the TeaServlet. Go to the TeaServlet page at http://opensource.go.com/teaservlet/ and download the latest version. While you are here, you should download the other documents that interest you.

The TeaServlet's download instructions, requirements, installation instructions, and administration page descriptions are contained in the *TeaServlet User Manual*.

Once the TeaServlet is installed in the servlet container, the administration pages are available. You should ensure these are configured and working properly, as shown in the user manual, before you continue with this tutorial.

2. CREATING A TEASERVLET APPLICATION

Once TeaServlet is installed, the next step is to create a TeaServlet application. Kettle, an editor for Tea templates, should be installed at this point. Kettle and the *Kettle User Manual* are available for download at http://opensource.go.com.

2.1 Hello World

The following steps will describe how to create a simple "Hello World" application. The application will display a "Hello" greeting to the user based upon the user's name passed in the URL. For example, hitting the URL http://…/HelloWorld?name=Bob will display "Hello Bob".

2.1.1 The Tea Template

The first step is to create a Tea template. Open Kettle and create a new template that looks like this:

```
<% template HelloWorld()
yourName = getName()%>
Hello <%yourName%>
```

Note that the second line calls the function getName. We will define the contents of this function later. The results of the function are stored in a variable called yourName, and then outputted to the web page on the last line, just after the word "Hello".

Save this new template as "HelloWorld.tea" in the directory specified for the templates**.path** property in the properties file. If you save the file elsewhere, the TeaServlet will not be able to find your template.

For more information about writing Tea templates, refer to the Tea User Manual.

2.1.2 The TeaServlet Application

The second step is to create your TeaServlet application class. An application is similar to a servlet in that it has an init method and a destroy method. Instead of having a doGet (or service) method, it has a getContextType method and a createContext method.

The application allows us to add a group of functions to the TeaServlet. All of the functions are available to the Tea templates.

You need to create the application class and call it "HelloWorldApplication". This is how the class will look:

```
package sample;

import com.go.teaservlet.*;
import javax.servlet.ServletException;

public class HelloWorldApplication implements Application {

	// initialize the application
	public void init(ApplicationConfig config)
	throws ServletException {
	}

	// destroy the application
	public void destroy() {
	}

	// return the class that contains the functions
	public Class getContextType() {
		return HelloWorldContext.class;
	}

	// return an instance of the functions class
	public Object createContext(ApplicationRequest request,
				ApplicationResponse response) {
		return new HelloWorldContext(request);
	}
}
```

The init method and destroy method are empty in this example. You can ignore these methods for now. The important methods are getContextType and createContext.

The getContextType method tells the TeaServlet which class contains our functions. The class that contains the functions is called the context. In our example, a class called "HelloWorldContext" is our context.

The **createContext** method is used to create an instance of the context class. This instance must be of the same class that your getContextType method returns, HelloWorldContext.

The createContext method is very similar to the doGet method of a standard servlet. Both of these methods are passed a request and a response object. We can use the request object to get information about the user's request.

2.1.3 The Context Class

Once you have an application, the next step is to build a context class. This is how the context class will look:

```
package sample;

import javax.servlet.http.*;

public class HelloWorldContext {

	HttpServletRequest mRequest;

	public HelloWorldContext(HttpServletRequest request) {

		// save the user's request object
		mRequest = request;
	}

	public String getName() {

		// get the "name" parameter from the URL
		String name = mRequest.getParameter("name");
		if (name == null) {
			name = "World";
		}

		return name;
	}
}
```

The HelloWorldContext class contains all the functions that you want to be available to the Tea templates. In this case, there is only one function that you are making available, the getName function. The getName function needs to get information from the user's request. To handle this, you are passing the request into the constructor of this class.

The getName method tries to get the "name" parameter from the URL. If it doesn't exist, it returns the default value of "World".

2.1.4 Running The HelloWorld Sample

The last step before running HelloWorld is to add the HelloWorld application in the TeaServlet properties file. Find the applications branch in the TeaServlet properties file and add the class as shown below:

```
applications {

	"HelloWorld" {
		class = sample.HelloWorldApplication
	}
}
```

Other applications such as the administration application for the TeaServlet will also be in the applications branch. Do not remove the administration application or you won't be able to access the administration pages.

Once the TeaServlet properties file is saved, the TeaServlet will load the HelloWorld application class when it starts up. If your servlet container supports servlet reloads then reload the TeaServlet. If not, restart your servlet container. HelloWorld is now ready to run.

To run HelloWorld, go to the administration pages (if needed, refer to the *TeaServlet User Manual*) and click on the **Applications** link. The HelloWorldApplication is listed there. Click on the **Templates** link and you will see the HelloWorld template listed. Click on the **HelloWorld** template. The template will execute and you will see "Hello World". Adding a name parameter to

the end of a URL, such as "HelloWorld?name=Bob", will cause the template to display "Hello Bob" instead.

If you see the display as expected, you have correctly created a TeaServlet application with functions that can be called from a template.

Because this web application cleanly separates logic from presentation, more templates can be added that use the getName function without changing any Java code. Each new template is simply a new presentation of the same data.

2.1.5 Adding a Second Template

To demonstrate multiple templates, we will create a "BonjourWorld" template. (Bonjour is how you say "hello" in French.) Open Kettle and create a template that looks like this:

```
<% template BonjourWorld()
yourName = getName()%>
Bonjour <%yourName%>
```

Once the template is appropriately saved, go back to the administration pages and click on the **Templates** link. Click on the **Reload Changes** button. The administration page will indicate that the TeaServlet compiled and loaded the BonjourWorld template. Hit the URL http://…/BonjourWorld?name=Pierre and you will see: "Bonjour Pierre".

You have created a new template and made it available to your users without stopping the TeaServlet and without any loss of page serving. Templates can be reloaded while the TeaServlet is still running and serving your web site. New templates can be added and existing templates can be modified all while the web site is live and serving users.

3. UNDERSTANDING THE TEASERVLET

3.1 Application Life Cycle

The life cycle of a TeaServlet application includes calls to four methods: the init method, the getContextType method, the createContext method, and the destroy method.

3.1.1 Init Method

The init method of the application is called after the application instance is constructed. Note that the init method of the application looks very similar to the init method of a servlet. The only difference between these init methods is the TeaServlet application uses the ApplicationConfig class instead of the Servlet API's ServletConfig class. Since the ApplicationConfig class extends the ServletConfig class, the TeaServlet application's init method can do everything a servlet's init method can.

Use the application init method to initialize your application. You can read initialization parameters in the init method or initialize global resources. If the application accesses a database, create a database connection pool in the application method so a new connection to the database does not have to be created for every request.

3.1.2 GetContextType Method

The getContextType method is called after the init method exits. This method tells the TeaServlet which context class your application will be providing.

3.1.3 CreateContext Method

The createContext method is called for every request to a template, even if your application is not used by the template. For this reason, minimize the amount of work in the method and leave the logic within the functions in your context.

The createContext method is very similar to the doGet method of a servlet. Both methods take a request and a response object. The ApplicationRequest extends the Servlet API's HttpServletRequest. This means that any standard servlet request method can be called on the ApplicationRequest. Likewise, the ApplicationResponse extends the Servlet API's HttpServletResponse. Therefore, you have access to everything that you have in a regular servlet.

If multiple requests are made to the TeaServlet, it is possible for multiple threads to be in this method simultaneously. For this reason, it is very important that code in the createContext method is thread-safe, also recommended for a doGet method.

3.1.4 Destroy Method

The application's destroy method is used to free resources that have been allocated. This method will be called whenever the servlet container calls the TeaServlet's destroy method. However,

the destroy method cannot always be relied upon. For example, if the servlet container crashes, the TeaServlet's destroy method will not be invoked.

3.2 Comparing TeaServlet Applications To Servlets

There is little difference between building a TeaServlet application and creating a standard servlet. The primary difference is the separation that the TeaServlet enforces between data and presentation. The results of this separation are explained in the following sections.

3.2.1 The HelloWorld Sample

The HelloWorld sample, described earlier in the document, could easily be written as a servlet. It would look something like this:

```
package sample;

import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

        public void init(ServletConfig config)
        throws ServletException {
        }

        public void destroy() {
        }

        public void doGet(HttpServletRequest request,
                          HttpServletResponse response) {

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String yourName = getName(request);
                out.println("Hello " + yourName);
        }

        public String getName(HttpRequest request) {

                // get the "name" parameter from the URL
                String name = request.getParameter("name");
                if (name == null) {
                        name = "World";
                }

                return name;
        }
}
```

3.2.2 Init And Destroy Methods

The init and destroy methods in the two HelloWorld examples are identical except for the parameters. Although the methods in the HelloWorld examples are not doing anything, they would be implemented the same way in both the TeaServlet application and the servlet if there were some initialization code.

3.2.3 DoGet And GetName Methods

The doGet method in the servlet is comparable to the createContext method in the TeaServlet application. The primary differences are between methods in the TeaServlet application and the servlet are with these two methods.

In the servlet's doGet method, the data is retrieved from the getName function and the presentation is sent to the PrintWriter's output stream. The getName method in the servlet is passed in the user's request.

In the applications createContext method, an instance of the HelloWorldContext class is returned. The getName method is defined in the context and the user's request is passed into the context instead of the getName method.

With the exception of those differences, the getName method is the same as the createContext method. The createContext method and the context only define the logic piece. The presentation is completely separate and resides in the Tea template file.

3.2.4 Summary

Creating a TeaServlet application is similar to creating a servlet because the TeaServlet is only a thin layer on top of the Servlet API. Due to this similarity, converting a servlet to a TeaServlet application is easily accomplished in most cases.

The main difference between the TeaServlet application and a servlet is the separation between data and presentation. The TeaServlet's main goal is to enforce a separation between data and presentation so that is why the differences are in that area only.

The following table shows how functionality in a servlet directly maps to functionality in a TeaServlet application:

Functionality	Servlet	TeaServlet application
Initialization	HttpServlet's init method	Application's init method
Clean up	HttpServlet's destroy method	Application's destroy method
Business logic	HttpServlet's doGet method and data functions	Application's createContext method and data functions in context class
Presentation logic	HttpServlet's doGet method	Tea template

3.2.5 Why Use the TeaServlet?

With so many similarities and so little difference, users may wonder why they should use the TeaServlet instead of a standard servlet. The main benefit to using the TeaServlet is that you don't have to change your Java code every time you want to change a view or make a new view of the existing data.

The HelloWorld sample for the TeaServlet illustrates this capability. By adding the BonjourWorld template, we were able to create an additional view on top of the existing functionality.

In contrast, the HelloWorld sample servlet is more difficult. To say "Bonjour Pierre" as we did earlier with the TeaServlet, you would have to modify your Java code and include a flag that specified which view you wanted.

4. SAMPLE APPLICATIONS

The following sections illustrate additional sample applications.

4.1 The News Sample

You will start by creating the NewsApplication. This application will return local news story objects based on the location of news that is requested. For this sample, you will read the news stories from the properties file.

4.1.1 The NewsApplication Class

Create the NewsApplication class. It should look like this:

```
package sample;

import com.go.teaservlet.*;
import java.util.*;
import javax.servlet.ServletException;

public class NewsApplication implements Application {

	// the application config
	private ApplicationConfig mConfig;

	// a list of local news stories
	private List mNewsStories;

	// initialize the application
	public void init(ApplicationConfig config)
	throws ServletException {

		// save the application config
		mConfig = config;

		// create the news stories list
		mNewsStories = new Vector()

		// load the news stories from the properites file
		loadNewsStories();
	}

	// destroy the application
	public void destroy() {
	}

	// return the class that contains the functions
	public Class getContextType() {
		return NewsContext.class;
	}

	// return an instance of the functions class
	public Object createContext(ApplicationRequest request,
				ApplicationResponse response) {
		return new NewsContext(mNewsStories);
	}

	// load the news stories from the properties file
	public void loadNewsStories() {

		// get the number of news stories
```

```
                int newsstories = 0;
                String parameter = mConfig.getInitParameter("newsstories");
                if (parameter == null) {
                        mConfig.getServletContext().log(
                                "Error: newsstories property is missing " +
                                "from the properties file");
                }
                else {
                        try {
                                newsstories = Integer.parseInt(parameter);
                        }
                        catch (NumberFormatException e) {
                                mConfig.getServletContext().log(
                                        "Error: newsstories property is not " +
                                        "a valid number");
                        }
                }

                // read each news story
                for (int story = 1; story <= newsstories; story++) {
                        String prefix = "newsstory" + story + ".";
                        String location = mConfig.getInitParameter(prefix +
                                                "location");
                        String headline = mConfig.getInitParameter(prefix +
                                                "headline");
                        String body = mConfig.getInitParameter(prefix +
                                                "body");

                        // create news story object and add to list of stories
                        NewsStory newsstory = new NewsStory(location,
                                                headline, body);
                        mNewsStories.add(newsstory);
                }
        }
}
```

The NewsApplication has two data members: the application config and the news stories. These are initialized in the init method. The init method then calls the loadNewsStories function.

The loadNewsStories function loads the news stories from the properties file. If you were creating a real application, you would probably be reading these from a database or an Enterprise JavaBeans container instead.

4.1.2 The NewsStory Class

Notice the loadNewsStories function uses a NewsStory class. Create that class now. It will look like this:

```
package sample;

import com.go.teaservlet.*;

public class NewsStory {

	// the location of this news story
	private String mLocation;

	// the headline of this news story
	private String mHeadline;

	// the body of this news story
	private String mBody;

	// constructor for the NewsStory object
	public NewsStory(String location, String headline, String body) {

		mLocation = location;
		mHeadline = headline;
		mBody = body;
	}

	// get the location of this news story
	public String getLocation() {

		return mLocation;
	}

	// get the headline of this news story
	public String getHeadline() {

		return mHeadline;
	}

	// get the body of this news story
	public String getBody() {

		return mBody;
	}
}
```

The NewsStory class is a basic JavaBean. It has some data members and some accessor methods to get at the data.

4.1.3 The NewsContext Class

The NewsContext class is the last class needed before this sample will run. This is how the NewsContext class will look:

```
package sample;

import com.go.teaservlet.*;
import java.util.*;

public class NewsContext {

        // a list of local news stories
        private List mNewsStories;

        // constructor for the NewsContext object
        public NewsContext(List newsStories) {

                mNewsStories = newsStories;
        }

        // get list of stories for the location
        public NewsStory[] getNewsStoriesByLocation(String location) {

                List stories = new Vector();
                for (int count = 0; mNewsStories.size(); count++) {
                        NewsStory story = (NewsStory)mNewsStory.get(count);
                        if ((location == null) ||
                            (location.equals(story.getLocation()) {
                               stories.add(story);
                        }
                }

                // convert the list to an array
                return (NewsStory[])stories.toArray(
                                        new NewsStory[stories.size()]);
        }

        // get list of stories for all locations
        public NewsStory[] getNewsStories() {

                return getNewsStoriesByLocation(null);
        }
}
```

The list of news stories is passed into the NewsContext class. These stories are saved in a data member. You have created two functions that are available to be called by the Tea templates, the getNewsStoriesByLocation function and the getNewsStories function. The getNewsStoriesByLocation function only returns stories that match the specified location.

4.1.4 Setting Up The Properties File

Once you compile your classes, you are ready to setup the NewsApplication in the properties file. Initialization parameters need to be added under the "applications" branch. It will look something like this:

```
applications {

        "MyNewsSample" {

                # class file of the application
                class = sample.NewsApplication

                # initialization parameters for the application
                init {
                        # number of news stories
                        newsstories = 3

                        # first news story
                        newsstory1 {
                                location = seattle
                                headline = Seattle Mariners Win World Series
                                body = The Mariners beat the NY Yankees 4-2.
                        }

                        # second news story
                        newsstory2 {
                                location = seattle
                                headline = Space Needle Falls!
                                body = Seattle's Space Needle fell today /
                                        during a small earthquake.
                        }

                        # third news story
                        newsstory3 {
                                location = losangeles
                                headline = Detective Fired From LAPD
                                body = The LAPD fired Detective Smith today.
                        }
                }
        }
}
```

The initialization parameters setup here are the same ones that the loadNewsStories method will use. The "newsstories" parameter specifies how many news stories there will be. Each news story is then specified.

Instead of embedding the location, headline, and body parameters in the newsstory1 section, you could specify them like this:

```
# first news story
newsstory1.location = seattle
newsstory1.headline = Seattle Mariners Win World Series
newsstory1.body = The Mariners beat the NY Yankees 4-2.
```

Properties can be specified either way. The TeaServlet supports either syntax in the properties file.

4.1.5 The Tea Template

Once the Java coding is finished and the properties file is setup, you need to create a Tea template that uses your functions.

Start up Kettle and create a new project. Go to the **Project** menu and select **Properties**. On the **Context Classes** tab, you will specify your context. Click on **New** and enter the class name of

sample.NewsContext. Make sure that the context class is in the class path that is specified on the **ClassPath** tab.

If you view the Tea Function Browser in Kettle, you will see the NewsContext. You will also see the functions that you can call in your template. These functions can also be viewed from the TeaServlet administration pages by clicking on the **Functions** link.

In Kettle, you should setup all the contexts that are displayed on the **Applications** link of the TeaServlet administration pages. This will allow you to access any of the functions in your templates.

Let's create a template now:

```
<% template NewsPage(String location)

if (location == null) {
        // get all news stories
        newsStories = getNewsStories()
}
else {
        // get local news stories only
        newsStories = getNewsStoriesByLocation(location)
}
%>

<b>Today's News:</b>
<hr>

<% foreach (story in newsStories) { %>
        <i><% story.headline %></i><br>
        <% story.body %><br>
<% } %>

That's all folks!
```

Compile your template in Kettle to make sure there are no errors. Then load the template into the TeaServlet by clicking **Reload Changes** on the **Template** link in the administration pages. You can view your template by hitting http://…/NewsPage in your browser.

Let's take a closer look at this template.

Notice the "<%" and "%>" tags throughout the template. These tags specify the start and end of code regions. Anything outside the code block is outputted as text and is referred to as a text region. Anything inside a code region is executed.

The top of every template starts with the template declaration. You must have this as the first line in your template.

Parameters can be passed to templates. In this case, we are getting the "location" parameter from the URL. Viewing http://…/NewsPage?location=seattle will set the location parameter to "seattle". View that URL and you will see the Seattle news. View http://…/NewsPage?location=losangeles to see the Los Angeles news.

The foreach loop is the only loop allowed in Tea. This allows you to loop over an array or collection of objects.

Create a second template now that only shows the Seattle news. It will look like this:

```
<% template SeattleNews()
call NewsPage("seattle")
%>
```

Here you are calling your NewsPage template to do the work. In this case, you are using the NewsPage template like a function. This "function" outputs the news for the selected location. You will find that making presentation functions in templates is a good way to reuse presentation logic. Calling other templates is also a useful way to break up a template into smaller parts.

You could write the SeattleNews template like this:

```
<% template SeattleNews2()

// get seattle news stories only
newsStories = getNewsStoriesByLocation("seattle")
%>

<b>Today's News:</b>
<hr>

<% foreach (story in newsStories) { %>
       <i><% story.headline %></i><br>
       <% story.body %><br>
<% } %>

That's all folks!
```

This SeattleNews2 template would produce the same results as the SeattleNews template. The difference is obvious. You have had to do a lot more work in the second template. If you decide to change the "That's all folks!" message to "The End", you would have to change this in your NewsPage template as well as the SeattleNews2 template. Using the SeattleNews template, the change to the NewsPage template would be all that is required.

For more information about writing Tea templates, see the Tea User Manual.

4.1.6 Changes To The NewsApplication

The NewsApplication creates a new instance of the context for every request. Since the context does not have any request-based data, you could change the application to be slightly more efficient. You would change your class to look like this:

```
public class NewsApplication implements Application {

        ...

        // the context instance
        private NewsContext mContext;

        // initialize the application
        public void init(ApplicationConfig config)
        throws ServletException {

                ...

                // create an instance of the context
                mContext = new NewsContext(mNewsStories);
        }

        // return an instance of the functions class
        public Object createContext(ApplicationRequest request,
                                    ApplicationResponse response) {
                return mContext;
        }
}
```

Note that the HelloWorldApplication class can't be changed to work like this because it uses request-based data. Each user needs a new copy of the context because the "name" parameter is being accessed from the request.

4.1.7 Changing The Data Source

Because the application logic and the presentation logic are separate, you can change your application implementation at any time without having to change your presentation logic in the templates.

Suppose that you have found a better source for your news stories and you can access these stories from an XML file. You would modify your application to work like this:

```
package sample;

import com.go.teaservlet.*;
import java.io.*;
import javax.servlet.*;

public class NewsApplication implements Application {

	// the news file
	private File mNewsFile;

	// initialize the application
	public void init(ApplicationConfig config)
	throws ServletException {

		// make sure the xmlfile parameter is specified
		String filename = config.getInitParameter("xmlfile");
		if (filename == null) {
			throw new ServletException("Missing xmlfile property");
		}

		// make sure the file exists
		mNewsFile = new File(filename);
		if (!mNewsFile.exists()) {
			throw new ServletException("File doesn't exist: " +
							filename);
		}
	}

	// destroy the application
	public void destroy() {
	}

	// return the class that contains the functions
	public Class getContextType() {
		return NewsContext.class;
	}

	// return an instance of the functions class
	public Object createContext(ApplicationRequest request,
				ApplicationResponse response) {
		return new NewsContext(mNewsFile);
	}
}
```

In this new NewsApplication class, you are throwing a ServletException if an error occurs. This allows the servlet container to handle the error for you.

You also need to re-implement the NewsContext class. It will look like this:

```
package sample;

import com.go.teaservlet.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;

public class NewsContext {

	// the news file
	private File mNewsFile;

	// constructor for the NewsContext object
	public NewsContext(File newsFile) {

		mNewsFile = newsFile;
	}

	// get list of stories for the location
	public NewsStories[] getNewsStoriesByLocation(String location) {

		// create list to store selected stories in
		ArrayList stories = new ArrayList(children.size());

		// open the file and find the root element
		FileInputStream file = new FileInputStream(mNewsFile);
		Element root = new SAXBuilder().build(file).getRootElement();

		// iterate through the nodes of the xml tree
		List children = root.getChildren();
		Iterator iter = children.iterator();
		while (iter.hasNext()) {
			Element child = (Element)iter.next();
			if ("news".equals(child.getName())) {
				try {
					String locale = child.
							getAttribute("location);
					if ((location == null) ||
					    (location.equals(locale))) {
					   String headline = child.
							getAttribute("headline");
					   String body = child.
							getAttribute("body");
					   NewsStory story = new NewsStory
							(locale, headline, body);
					   stories.add(story);
					}
				}
				catch (Exception e) {
				}
			}
		}

		return (NewsStories[])stories.toArray(
					new NewsStories[stories.size()]);
	}

	// get list of stories for all locations
	public NewsStories[] getNewsStories() {

		return getNewsStoriesByLocation(null);
	}
}
```

Most of the work is performed in the NewsContext class. For every request to the getNewsStories or getNewsStoriesByLocation functions, the XML file is being parsed.

The JDOM library is used to parse the XML file. You will need to download the library before you can get this sample to compile. The library is available from http://www.jdom.org.

You will have to create a sample news file. It will look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<newsfile>

<news location="seattle">
<headline>Seattle Mariners Win World Series</headline>
<body>The Mariners beat the NY Yankees 4-2.</story>
</news>

<news location="seattle">
<headline>Space Needle Falls!</headline>
<body>Seattle's Space Needle fell today during a small earthquake.</story>
</news>

<news location="losangeles">
<headline>Detective Fired From LAPD</headline>
<body>The LAPD fired Detective Smith today.</story>
</news>

</newsfile>
```

Now you are ready to run this application in the TeaServlet. Since you didn't change the names of your functions, there are no changes required to the Tea templates. Your previous Tea templates will work fine. Hit the templates a few times and then add some more news items to the XML file. Your additional news items will be displayed by the templates as soon as the XML file is saved.

5. APPENDIX A: SERVLET RESOURCES

There is a lot of servlet information available on the Internet. Since this tutorial presumes knowledge of servlets, you may find this list of resources useful if you are not already a servlet developer. These resources are not maintained or supported by WDIG and are only provided as links to help you find information.

5.1 General Servlet Information

There are many sites that list general information about servlets. Here are some of them:

Web Page	URL
Sun's Product Page for Java Servlets	http://www.javasoft.com/products/servlet/
The Java Tutorial: Servlets	http://java.sun.com/docs/books/tutorial/servlets/
Story of a Servlet: An Instant Tutorial	http://java.sun.com/products/servlet/articles/tutorial/
Servlet Essentials	http://www.novocode.com/doc/servlet-essentials/

5.2 Servlet API

The JavaDocs for the Servlet API is an excellent resource for servlet programming.

Web Page	URL
Servlet 2.1 API	http://www.javasoft.com/products/servlet/2.1/api/packages.html
Servlet 2.2 API	http://www.javasoft.com/products/servlet/2.2/javadoc/

5.3 Servlet Books

"Java Servlet Programming", published by O'Reilly, is recommended. For information about this book, look at http://www.servlets.com/jsp/about.html.

You might also want to look at JavaWorld's review of servlet books at http://www.javaworld.com/javaworld/jw-03-2000/jw-03-ssj-books_p.html.